بسمه تعالی

گزارش کار پروژه اول

آیدین کاظمی 810101561

بخش اول

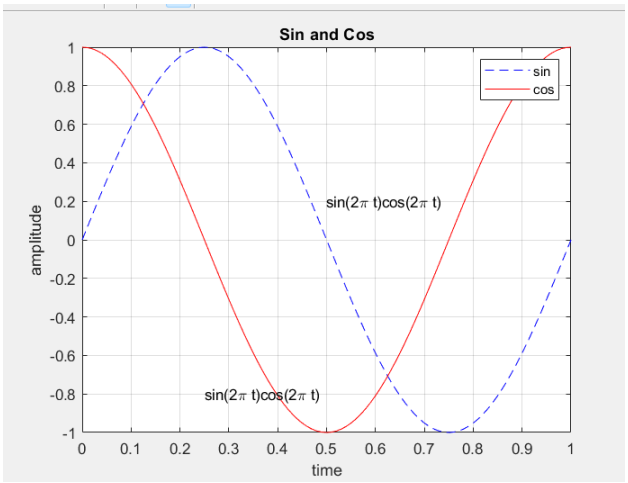
1-1)

در خط اول متغیر مستقل t به شکل یک بردار بین 0 و 1 با گام های 0.01 ساخته شده و سپس در خط های دو و سه توابع sin(2*pi*t) , cos(2*pi*t) به ترتیب ساخته شده و در دو بردار z1 و z2 ذخیره میشوند.

در بخش بعدی دستور figure صفحه رسم را باز کرده و با دستور های plot، دو تابع z1 و z2 بر حسب t رسم میشوند. نقش hold on در خط 7 نگه داشتن نمودار کشیده شده مربوط به z1 در صفحه رسم است و در صورتی که این خط حذف شود، فقط نمودار دوم در صفحه باقی میماند و نمودار اول پاک میشود. دو آرگومان b-- و r به ترتیب به معنای خط چین آبی و رنگ قرمز نمودار رسم شده هستند.

در مرحله بعد دو متغیر x0 و y0 را تعریف میکنیم که دو بردار حاوی به ترتیب طول و عرض های نقاط مد نظر برای قرار دادن متن های متناظر با نمودار های سینوس و کسینوس هستند، که این متن ها نیز در متغیر s تعریف و نگه داری شده و در نهایت با دستور text در صفحه نمودار ها در مکان تعین شده نوشته میشوند.

در بخش پایانی ابتدا title نام نمودار را مشخص کرده و سپس دستور legend نام مربوط به هر نمودار را در گوشه صفحه همراه با شکل آن نمودار رسم میکند. نهایتا xlabel و ylabel محور های متناظر خود را نام گذاری کرده و دستور grid on صفحه را به بخش های مربعی تقسیم میکند. نمودار نهایی:

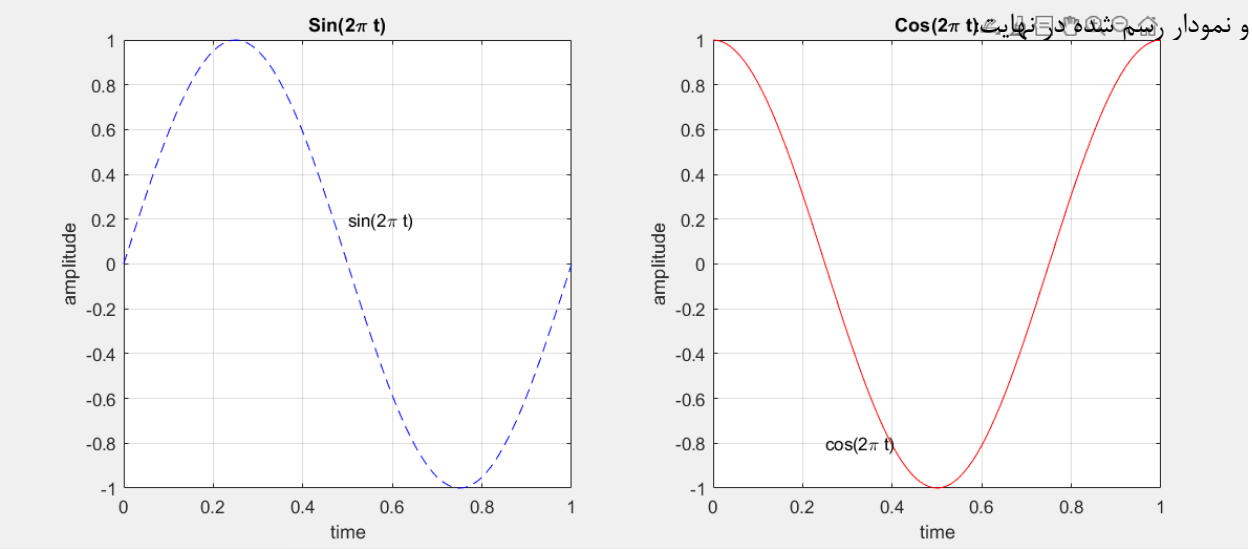


2-1)تابع subplot(m,n,x) به این صورت کار میکند که m تعداد سطر های حاوی عکس ها، n تعداد ستون های عکس و x شماره خانه مد نظر است و از 1 شروع شده و تا m * n ادامه دارد:

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

figure;
subplot(1,2,1);
plot(t, z1, '--b');

x0=0.5;
y0=0.2;
s_sin='sin(2\pi t)';
text(x0,y0,s_sin);

title('Sin(2\pi t)');
xlabel('time');
ylabel('amplitude');
grid on;
```

```
subplot(1,2,2);
plot(t, z2, 'r');

x0=0.25;
y0=-0.8;
s_cos='cos(2\pi t)';
text(x0,y0,s_cos);

title('Cos(2\pi t)');
xlabel('time');
ylabel('amplitude');
grid on;
```

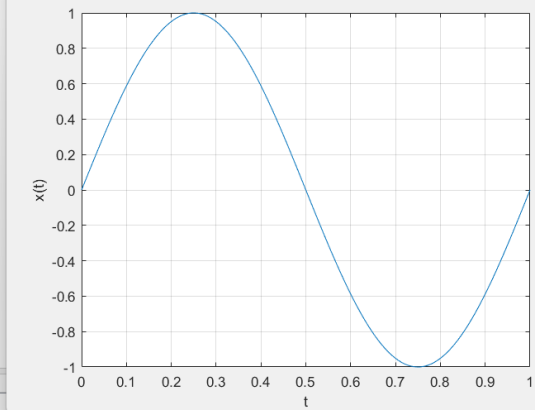
و نمودار رسم شده در نهایت:



بخش دوم

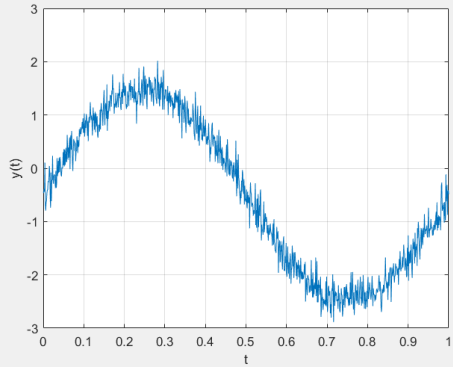
2-1) قطعه کد استفاده شده و شکل نمودار:

```
load p2.mat
plot(t,x)
xlabel('t')
ylabel('x(t)')
grid on
```



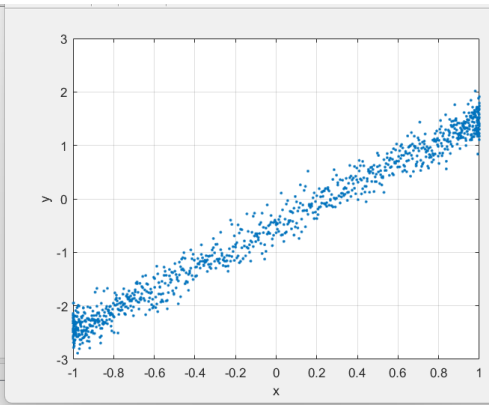
2-2) قطعه کد استفاده شده و شکل نمودار:

```
load p2.mat
plot(t,y)
xlabel('t')
ylabel('y(t)')
grid on
```



2-3) قطعه کد استفاده شده و شکل نمودار:

```
load p2.mat
plot(x,y,'.')
xlabel('x')
ylabel('y')
grid on
```



شیب خط برابر مقدار آلفا و عرض از مبدا برابر با مقدار بتا میباشد.

2-4) ابتدا از راهنمایی داده شده استفاده کرده و از معادله یک بار نسبت به آلفا و بار دیگر نسبت به بتا مشتق میگیریم، برابر با صفر قرار داده و مقادیر را برای آلفا و بتا حل میکنیم. توضیحات هوش مصنوعی برای جزئیات حل معادله به صورت زیر میباشد:

1. **Residual (error) for each data point:**

$$\text{Residual}_i = y_i - (ax_i + b)$$

2. **Total sum of squared residuals:**

$$S(a,b) = \sum_{i=1}^{n} (y_i - (ax_i + b))^2$$

3. **Partial derivative of $S(a,b)$ with respect to a:**

$$\frac{\partial}{\partial a} S(a,b) = -2 \sum_{i=1}^{n} x_i \left(y_i - (ax_i + b)\right)$$

Set to zero:

$$\sum_{i=1}^{n} x_i y_i = a \sum_{i=1}^{n} x_i^2 + b \sum_{i=1}^{n} x_i$$

4. **Partial derivative of $S(a,b)$ with respect to b:**

$$\frac{\partial}{\partial b} S(a,b) = -2 \sum_{i=1}^{n} (y_i - (ax_i + b))$$

Set to zero:

$$\sum_{i=1}^{n} y_i = a \sum_{i=1}^{n} x_i + nb$$

5. **Solving the system of equations:**

From:

$$\sum_{i=1}^{n} y_i = a \sum_{i=1}^{n} x_i + nb$$

Solve for b:

$$b = \frac{\sum_{i=1}^{n} y_i - a \sum_{i=1}^{n} x_i}{n}$$

6. **Substitute b into the first equation:**

$$\sum_{i=1}^{n} x_i y_i = a \sum_{i=1}^{n} x_i^2 + \left(\frac{\sum_{i=1}^{n} y_i - a \sum_{i=1}^{n} x_i}{n} \right) \sum_{i=1}^{n} x_i$$

Simplify to:

$$a = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

7. **Final expressions:**

$$a = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$$

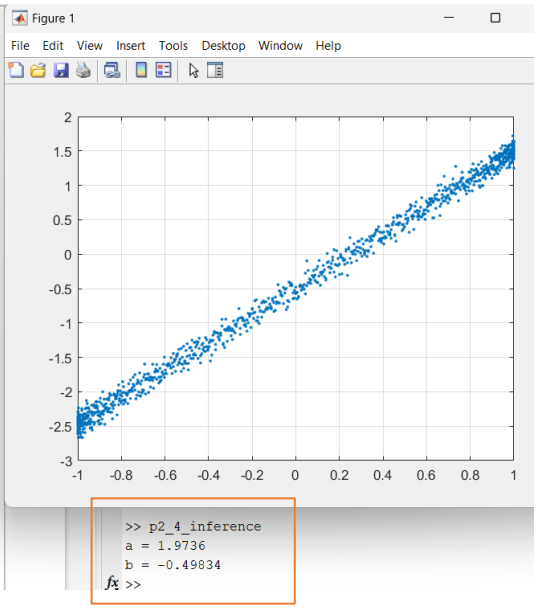
$$b = \frac{\sum y \sum x^2 - \sum x \sum xy}{n \sum x^2 - (\sum x)^2}$$

بنابراین و با توجه به توضیحات داده شده، اسکریپت زیر را مینویسیم (کامنت های مناسب برای توضیح گذاشته شده است):

```
function [a, b] = p2_4(x, y)
    sum_xy = sum(x .* y); % element wise mult istead of vector mult
    sum_xx = sum(x .^ 2);
    sum_y = sum(y);
    sum_x = sum(x);

    sum_power_2 = sum_x^2;
    n = length(x);

    % The formula is: a = (n * Σ(xy) - Σ(x) * Σ(y)) / (n * Σ(x^2) - (Σ(x))^2)
    a = (n * sum_xy - sum_x * sum_y) / (n * sum_xx - sum_power_2);


    % The formula is: b = (Σ(y) * Σ(x^2) - Σ(x) * Σ(xy)) / (n * Σ(x^2) - (Σ(x))^2)
    b = (sum_y * sum_xx - sum_x * sum_xy) / (n * sum_xx - sum_power_2);

end
```

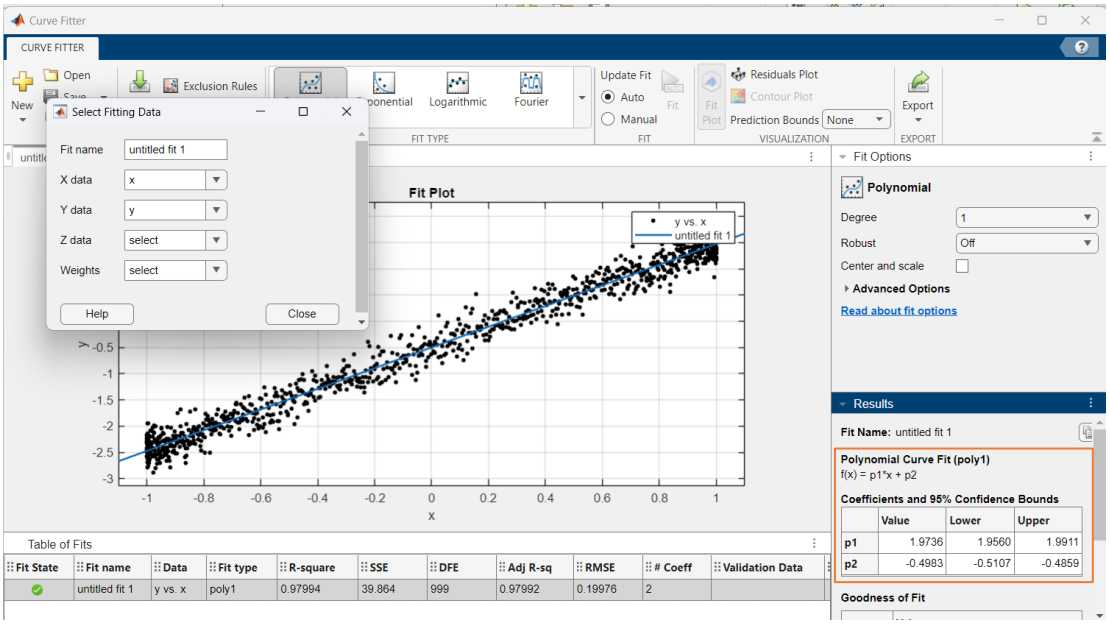
حال داخل اسکریپت p2_4_inference هم حالت با نویز و هم حالت بدون نویز را بررسی میکنیم. کد این اسکریپت به همراه نمودار رسم شده با نویز و همچنین مقادیر a و b (در قسمت کامند ویندو) در تصویر زیر قابل مشاهده میباشند:

```
% no noise
load p2.mat

[a,b] = p2_4(x,y);

disp(['a = ', num2str(a)]); % change the number to str
disp(['b = ', num2str(b)]);

%with noise

noise=0.1*randn(1,length(x)); % normal samples with std = 0.1 and len(x)

% "a" value is multiplied by all elements of vector "x", same as "b"
new_y = a * x + b + noise;

plot(x,new_y,'.')
grid on
```



که شبیه به نمودار اولیه میباشد، و بنابراین صحت پاسخ ما را تایید میکند.

2-5) مراحل گفته شده را طی میکنیم و به نتیجه زیر میرسیم:

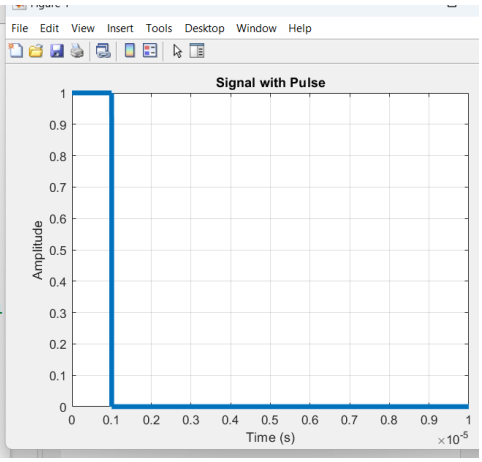


همانطور که در عکس مشاهده میشود، مقادیر به دست آمده با مقادیر کامپیوتری همخوانی دارند.

بخش سوم

3-1) قطعه کد مربوطه و شکل پالس در عکس زیر قابل مشاهده میباشد (کد با کامنت های مناسب توضیح داده شده):

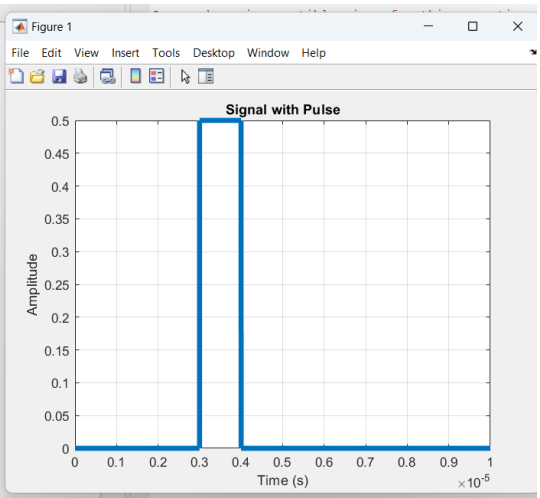
```m
ts = 1e-9;
T = 1e-5;
tau = 1e-6;
start_time = 0;

Total_samples = round(T/ts); % fp operation may fail to return int
pulse_samples = round(tau/ts); % fp operation may fail to return int

signal = zeros(1,Total_samples);

pulse = ones(1,pulse_samples);

signal(1:pulse_samples) = signal(1:pulse_samples) + pulse;

time = start_time:ts:T - ts; % if we don't subtract from ts -> len(time) == 10001

plot(time, signal, 'LineWidth', 4);
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal with Pulse');
grid on;
```



ابتدا یک سیگنال کلی به تعداد تمام سمپل های مورد نظر و به اندازه صفر درست میکنیم. در مرحله بعد، یک سیگنال به تعداد سمپل های پالس و دامنه یک درست کرده، پالس را به سیگنال کلی اضافه کرده و بردار زمان را میسازیم. نهایتا سیگنال را بر اساس بردار رسم میکنیم.

3-2) قطعه کد مربوطه و شکل پالس در عکس زیر قابل مشاهده میباشد (کد با کامنت های مناسب توضیح داده شده):

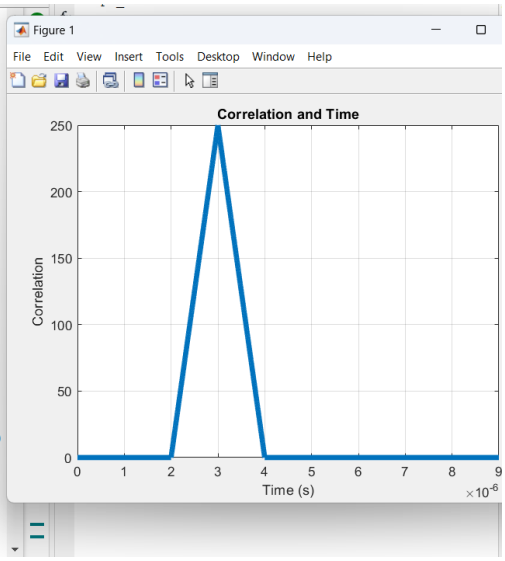
```m
ts = 1e-9;
T = 1e-5;
tau = 1e-6;
start_time = 0;
c = 3e8;
R = 450;
alpha = 0.5;

td = 2 * R / c;
pulse_idx = td / ts;

Total_samples = round(T/ts); % fp operation may fail to return int
pulse_samples = round(tau/ts); % fp operation may fail to return int

signal = zeros(1,Total_samples);
pulse = alpha * ones(1,pulse_samples); % pulse amp is reduced

% Here the pulse must start from td
signal(pulse_idx:pulse_idx+pulse_samples - 1) = ...
signal(pulse_idx:pulse_idx+pulse_samples - 1) + pulse; % -1 is due to len mismatch

time = start_time:ts:T - ts; % if we don't subtract from ts -> len(time) == 10001

plot(time, signal, 'LineWidth', 4);
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal with Pulse');
grid on;
```



همانطور که مشاهده میکنید در ابتدا مقدار td محسابه شده، سپس ایندکس مربوطه به این زمان حساب شده و نهایتا پالس از ایندکس جدید و با دامنه نصف دوباره رسم شده است.

3-3) قطعه کد مربوطه و شکل کوررلیشن در عکس زیر قابل مشاهده میباشد (کد با کامنت های مناسب توضیح داده شده):

```
ts = 1e-9;
T = 1e-5;
tau = 1e-6;
start_time = 0;
alpha = 0.5;

pulse_idx = 3000; %from the last section
Total_samples = round(T/ts); % fp operation may fail to return int
pulse_samples = round(tau/ts); % fp operation may fail to return int

signal = zeros(1,Total_samples);
pulse = alpha * ones(1,pulse_samples); % pulse amp is reduced

% Here the pulse must start from td
signal(pulse_idx:pulse_idx+pulse_samples - 1) = ...
signal(pulse_idx:pulse_idx+pulse_samples - 1) + pulse; % -1 is due to len mismatch

final_corr = zeros(1,Total_samples - pulse_samples);

for i=1:Total_samples - pulse_samples
    final_corr(i) = dot(signal(i:i+pulse_samples - 1), pulse);
end

time = start_time:ts:T - ts * (pulse_samples + 1); % if we don't subtract from ts -> len(time)

plot(time, final_corr, 'LineWidth', 4);
xlabel('Time (s)');
ylabel('Correlation');
title('Correlation and Time');
grid on;
```



همانطور که مشاهده میشود، به فرض داشتن ایندکس سیگنال و ساختن سیگنال مربوطه، با استفاده از حلقه میتوانیم یک پالس را در طول سیگنال جا به جا کرده (دامنه پالس جا به جا شونده نیز 0.5 میباشد که در ایندکس پیک تاثیری نمیگذارد) و نهایتا مقادیر کوررلیشن را ذخیره کرده و بر اساس زمان رسم کنیم، که همانطور که مشاهده میشود به طور تقریبی در زمان 3 این نمودار پیک میزند. حال با استفاده از کد زیر میتوانیم دقیق زمان را و سپس فاصله را حساب کنیم:

```
[val,index] = max(final_corr);
td = index * ts;
c = 3e8;
R = td * c / 2;
display(R)
```

```
R =

   450

>>
```

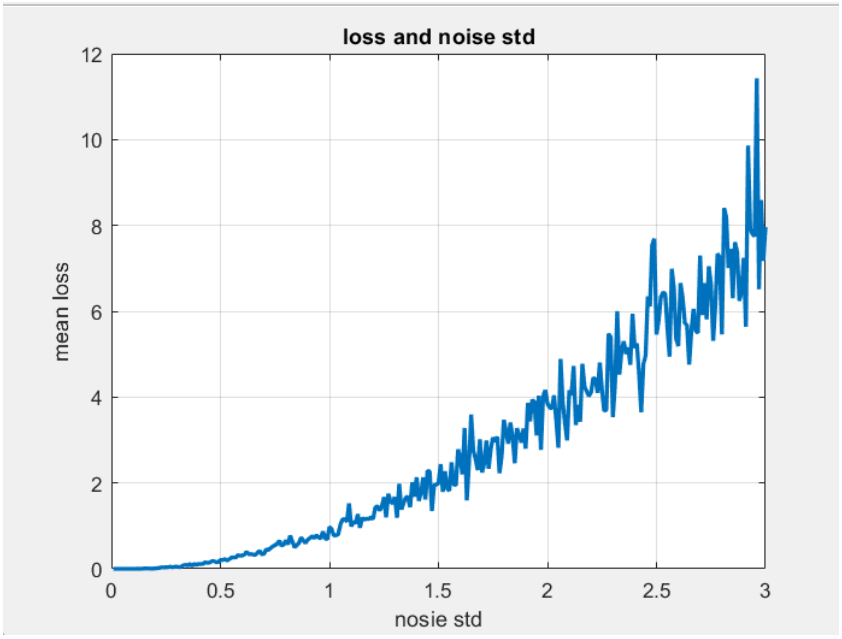
که همان نتیجه ای است که در قسمت قبل به دست آوریم.

3-4) قطعه کد مربوطه در عکس زیر قابل مشاهده میباشد (کد با کامنت های مناسب توضیح داده شده):

```
ts = 1e-9;
T = 1e-5;
tau = 1e-6;
start_time = 0;
alpha = 0.5;
max_noise_mult = 300;
max_iter = 100;
base_noise_std = 0.01;
true_R = 450;

pulse_idx = 3000; %from the last section
Total_samples = round(T/ts); % fp operation may fail to return int
pulse_samples = round(tau/ts); % fp operation may fail to return int

signal = zeros(1,Total_samples);
pulse = alpha * ones(1,pulse_samples); % pulse amp is reduced

% Here the pulse must start from td
signal(pulse_idx:pulse_idx+pulse_samples - 1) = ...
signal(pulse_idx:pulse_idx+pulse_samples - 1) + pulse; % -1 is due to len mismatch
```

```
noise_loss = zeros(1,max_noise_mult);
for noise_mult = 1:max_noise_mult
    loss_vector = zeros(1,max_iter); % a vector of 100 loss values
    for test_num = 1:max_iter % for 100 iterations
        noise = noise_mult * base_noise_std * randn(1,Total_samples);
        noisy_signal = signal + noise; % updating signal with noise

        %computing correlation and estimated R
        final_corr = zeros(1,Total_samples - pulse_samples);

        for i=1:Total_samples - pulse_samples
            final_corr(i) = dot(noisy_signal(i:i+pulse_samples - 1), pulse);
        end

        [val,index] = max(final_corr);
        td = index * ts;
        c = 3e8;
        R = td * c / 2;

        % loss of R
        loss_vector(test_num) = abs(true_R - R);
    end

    noise_loss(noise_mult) = mean(loss_vector);
end

noise_mult_vec = base_noise_std:base_noise_std:max_noise_mult * base_noise_std;

plot(noise_mult_vec, noise_loss, 'LineWidth', 2);
xlabel('nosie std');
ylabel('mean loss');
title('loss and noise std');
grid on;
```

همانطور که مشاهده میشود ابتدا سیگنال طبق مراحل قبل تولید شده، سپس با 300 انحراف معیار مختلف، هر بار 100 نویز تولید شده و میانگین خطای این نویز ها به عنوان خطای مربوط به انحراف معیار متناظر گزارش شده است. نهایتا مقادیر این میانگین خطا ها رسم شده که به شکل زیر میباشد:



همانطور که مشاهده میشود، طبق انتظاری که داشتیم خطا با بالا رفتن قدرت نویز بیشتر میشود و رابطه ای مستقیم بین این دو وجود دارد.

بخش چهارم

4-1) فایل ضبط شده را با کمک کد زیر لود کرده و فرکانس نمونه برداری آن را مشاهده میکنیم:

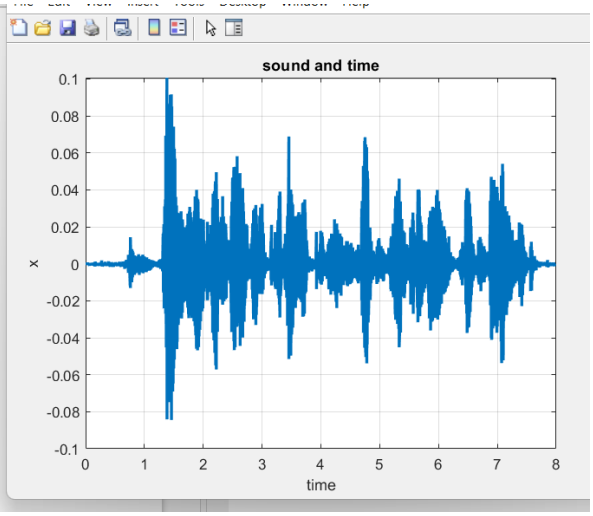
```
[x,fs] = audioread('poem.wav');
display(fs)
sound(x,fs);
```

```
fs =

        48000

fx >>
```

همانطور که مشاهده میکنید فرکانس نمونه برداری این فایل صوتی 48000 میباشد.

4-2) قطعه کد مربوطه و شکل موج صدا در عکس زیر قابل مشاهده میباشد:

```
[x,fs] = audioread('poem.wav');
time_step = 1 / fs;

time_vec_len = length(x);
time = time_step:time_step:time_vec_len * time_step;

plot(time, x, 'LineWidth', 2);
xlabel('time');
ylabel('x');
title('sound and time');
grid on;

sound(x,fs)
audiowrite('x.wav',x,fs);
```

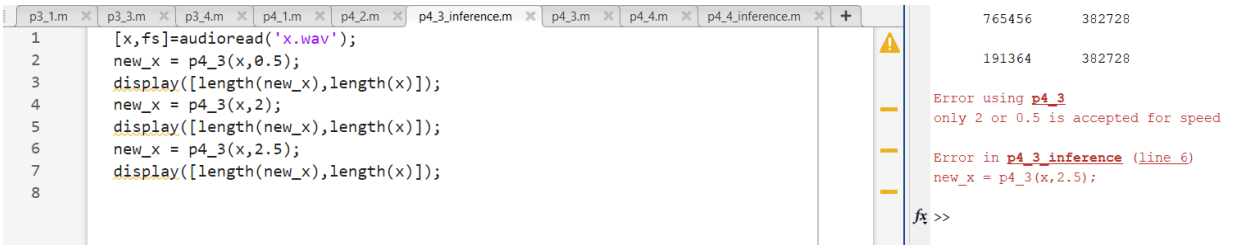


در اینجا فاصله زمانی نمونه ها را ابتدا با تقسیم یک بر فرکانس حساب کرده و سپس با کمک طول x و مقدار این فاصله های زمانی، محور زمان خود را تشکیل داده و نهایتا مقادیر را رسم میکنیم.

4-3) تابع نوشته شده به صورت زیر میباشد:

```
function new_x=p4_3(x,s)
    if s == 0.5
        new_x = zeros(1,2 * length(x)); % twice the length of x
        new_x(1:2:length(new_x)) = x;
        for i = 2:2:length(new_x)
            if i ~= length(new_x) % last sample may cause error
                new_x(i) = (new_x(i - 1) + new_x (i + 1))/2; % interpolation
            end
        end
    elseif s == 2
        new_x = zeros(1,round(0.5 * length(x))); % half the length of x
        new_x = x(1:2:length(x)); % removing half the samples
    else
        error('only 2 or 0.5 is accepted for speed')
    end
    fs = 48000; % assuming this won't change
    sound(new_x,fs)

end
```

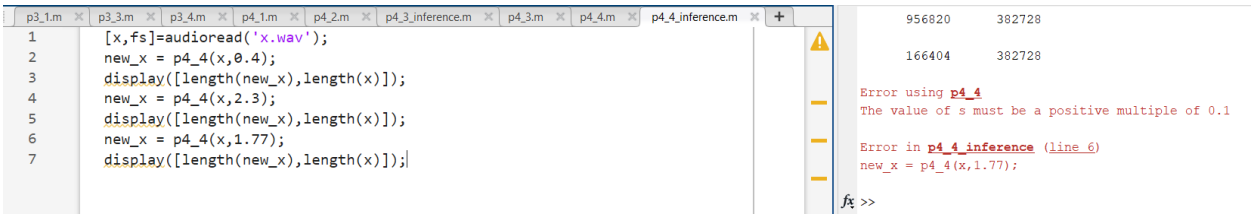
تابع را به صورت زیر، سه بار با مقادیر 0.5، 2 و غیر این دو تست میکنیم. همانطور که مشاهده میشود، طول وکتور برگشتی تغییر میکند و صدا نیز دچار تغییر میشود:



4-4) تابع نوشته شده به صورت زیر میباشد:

```
function new_x = p4_4(x, s)
    % Error handling
    if mod(s, 0.1) ~= 0 || s < 0
        error('The value of s must be a positive multiple of 0.1');
    end

    fs = 48000;  % Assuming this won't change
    p = 10;
    q = 10 * s; % int needed
    new_x = resample(x,p,q); % resampling by the needed factor of 1/s
    sound(new_x, fs);
end
```

که در آن از تابع resample استفاده شده که دو مقدار صحیح p و q را گرفته و از سیگنال به نسبت p/q سمپل کم میکند (حالت تند شدن) یا به سیگنال به همین نسبت با کمک اینترپولیشن سمپل اضافه میکند. از طرفی مقادیر غیر بخش پذیر بر 0.1 یا منفی نیز در این روش مورد قبول نمیباشد. تابع در خطوط زیر تست شده است:



که همانطور که میبینید طول وکتور و صدای پخش شده با توجه به سرعت تغییر کرده و تابع نیز به درستی خطا ها را هندل میکند.