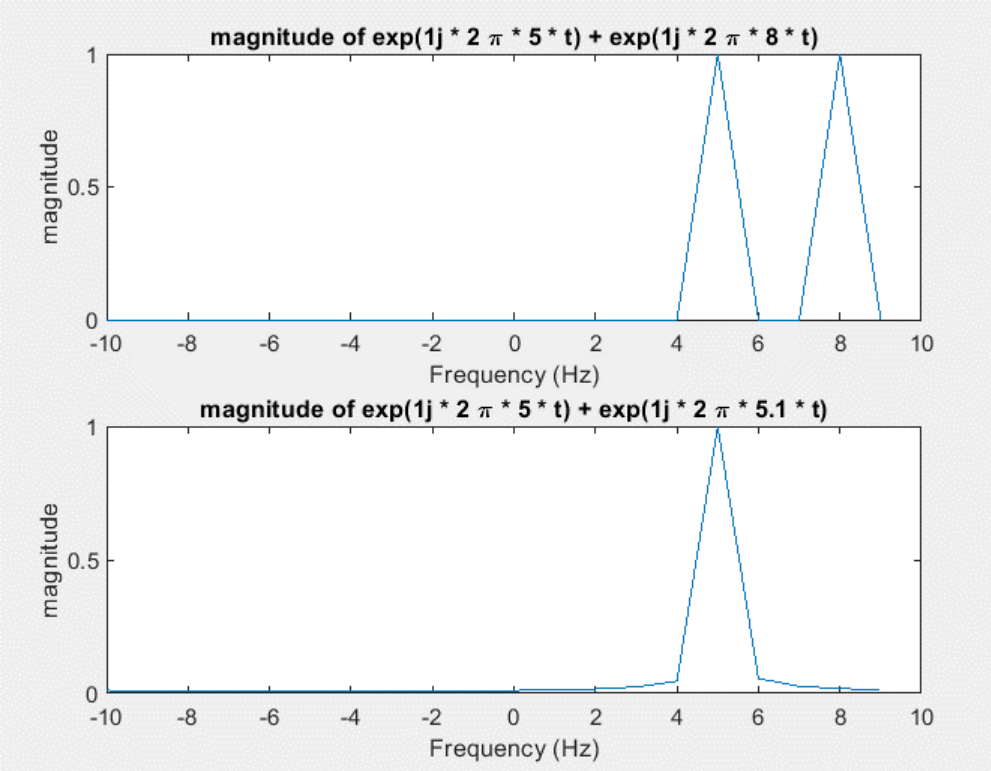
آوا میرمحمدمهدی 810199501

نسا عباسی مقدم 810199457

• بخش اول

قسمت صفر

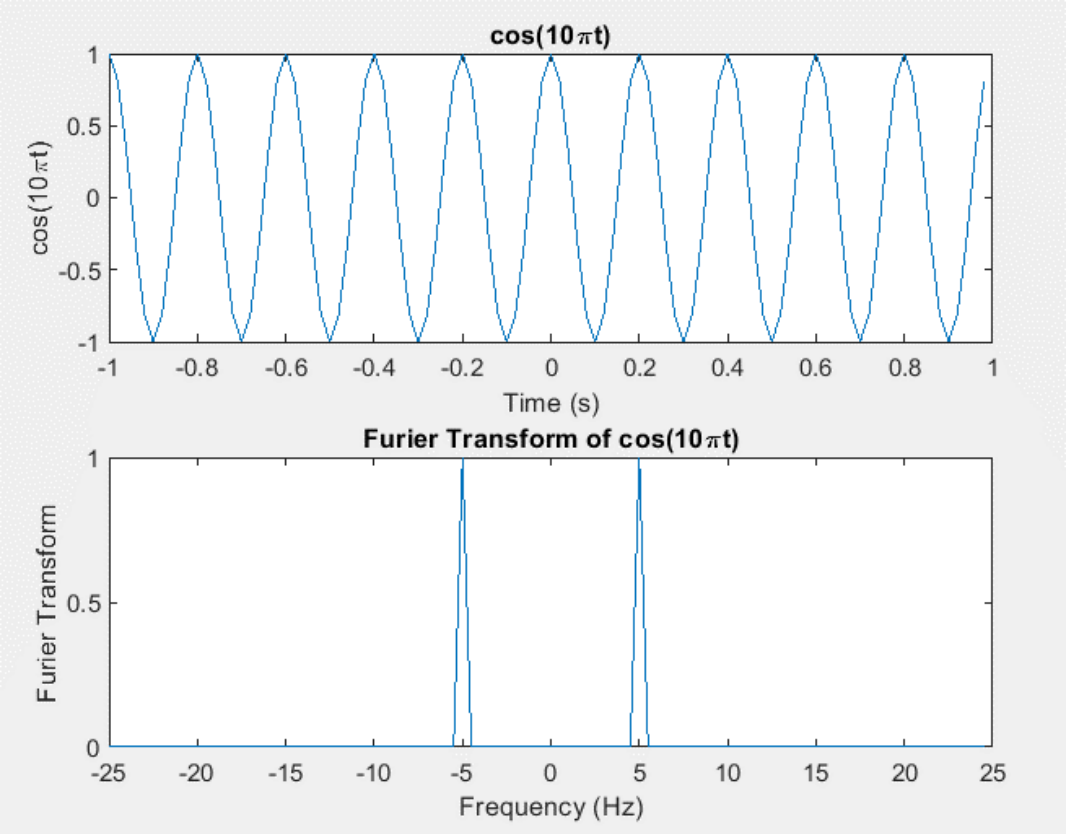
0.الف و ب:



همانطور که می‌بینیم در حالت دوم تنها یک پیک در فرکانس 5 مشاهده می‌شود و در فرکانس 5.1 قله‌ای قابل تشخیص نیست و این به دلیل آن است که اختلاف دو عدد 5 و 5.1 کمتر از 1 هرتز است و به همین دلیل توانایی تفکیک این دو سیگنال در حوزه فوریه وجود ندارد ولی در حالت اول اختلاف دو عدد 5 و 8 بیشتر از 1 هرتز است و می‌توان قله‌هایی متمایز از یکدیگر در این دو فرکانس مشاهده کرد.

قسمت یک

1.الف و ب:



1.ج

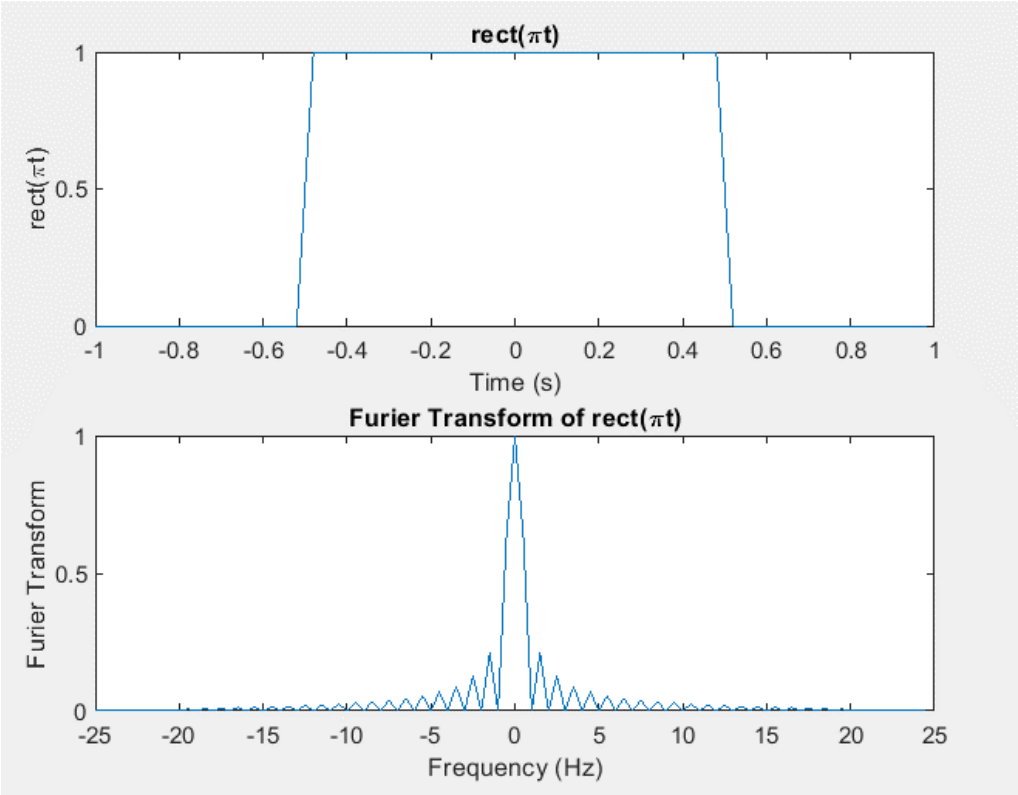
$$\hat{x}(\omega) = \int_{-\infty}^{\infty} \cos(10t) e^{-j\omega t} = \frac{1}{2}\int_{-\infty}^{\infty} \left(e^{-j(\omega-10\pi)t} + e^{-j(\omega+10\pi)t}\right)$$

$$= \pi\delta(\omega-10\pi) + \pi\delta(\omega+10\pi)$$

در متلب normalize شده‌ی تبدیل فوریه را رسم می‌کنیم پس ضرایب π بدست آمده در حالت تئوری بی‌اثر می‌شوند و همچنین نمودار براساس f رسم شده است و می‌دانیم $w = 2\pi f$؛ پس با انجام این تبدیل، تبدیل فوریه برابر با $\delta(f + 5) + \delta(f - 5)$ خواهد شد که مطابق نمودار بدست آمده در متلب است.

قسمت دو

2.الف و ب:



2.ج

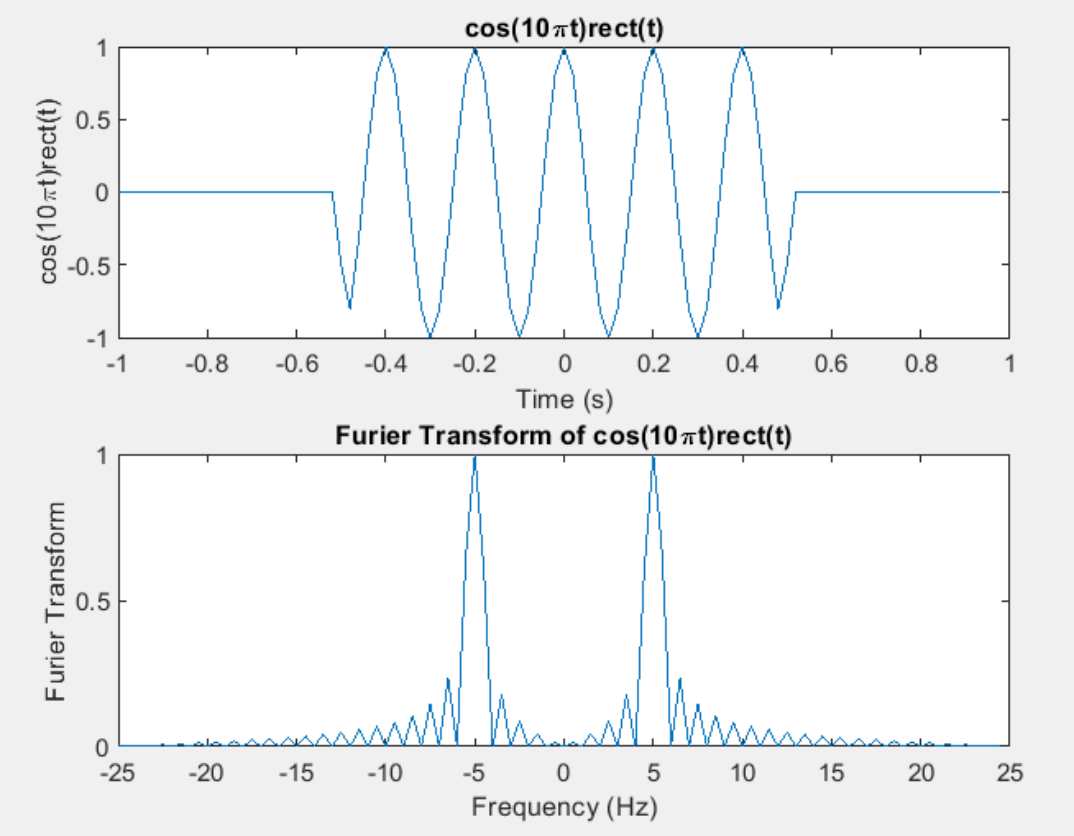
$$\hat{x}(w) = \int_{-1/2}^{1/2} e^{-jwt}\, dt = -jw\left[e^{-jwt}\right]_{-1/2}^{1/2} = \frac{-e^{jw/2}}{jw} + \frac{e^{jw/2}}{jw} = \frac{2\sin(w/2)}{w} = \frac{\sin(w/2)}{w/2}$$

$$= \mathrm{sinc}\left(\frac{w}{2\pi}\right)$$

اگر به جای w، 2πf قرار دهیم به sinc(f) می‌رسیم که اندازه‌ی آن مطابق نمودار رسم شده در متلب است.

قسمت سه

3.الف و ب:



3.ج

$$x_1 = \cos(10\pi t) \qquad x_2 = \Pi(t) \qquad x_3 = x_1(t)\,x_2(t) = \cos(10\pi t)\,\Pi(t)$$

$$\rightarrow \hat{x}_3(\omega) = \frac{1}{2\pi}\left(\hat{x}_1(\omega) * \hat{x}_2(\omega)\right) = \frac{1}{2\pi}\left(\left(\delta(f-5)+\delta(f+5)\right) * \mathrm{sinc}(f)\right)$$

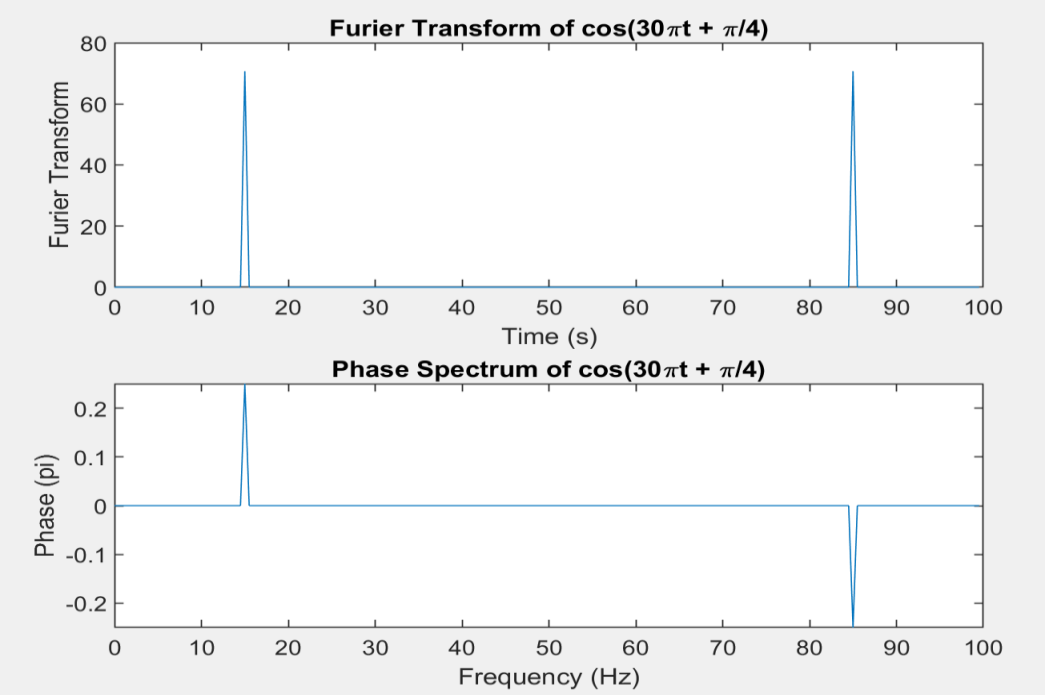
$$= \frac{1}{2\pi}\left(\mathrm{sinc}(f-5)+\mathrm{sinc}(f+5)\right)$$

$$\rightarrow |\hat{x}_3(\omega)| = \left|\mathrm{sinc}(f-5)+\mathrm{sinc}(f+5)\right|$$

همانطور که می‌بینیم در f = 5 و f = -5 داخل sinc برابر با 0 می‌شود که در نمودار نیز در این دو مکان قله دیده می‌شود؛ همچنین در اینجا نیز از ضرایب صرف نظر می‌کنیم چون در متلب نمودار نرمال شده، رسم شده است.

قسمت چهار

4.الف و ب:



4.ج

$$\mathcal{F}\{\cos(30\pi t+\tfrac{\pi}{4})\} = \int_{-\infty}^{\infty} \tfrac{1}{2}\left(e^{j(30\pi t+\pi/4)} + e^{-j(30\pi t+\pi/4)}\right)e^{-j\omega t}\,dt$$

$$= \pi e^{-j\pi/4}\,\delta(\omega+30\pi) + \pi e^{j\pi/4}\,\delta(\omega-30\pi)$$

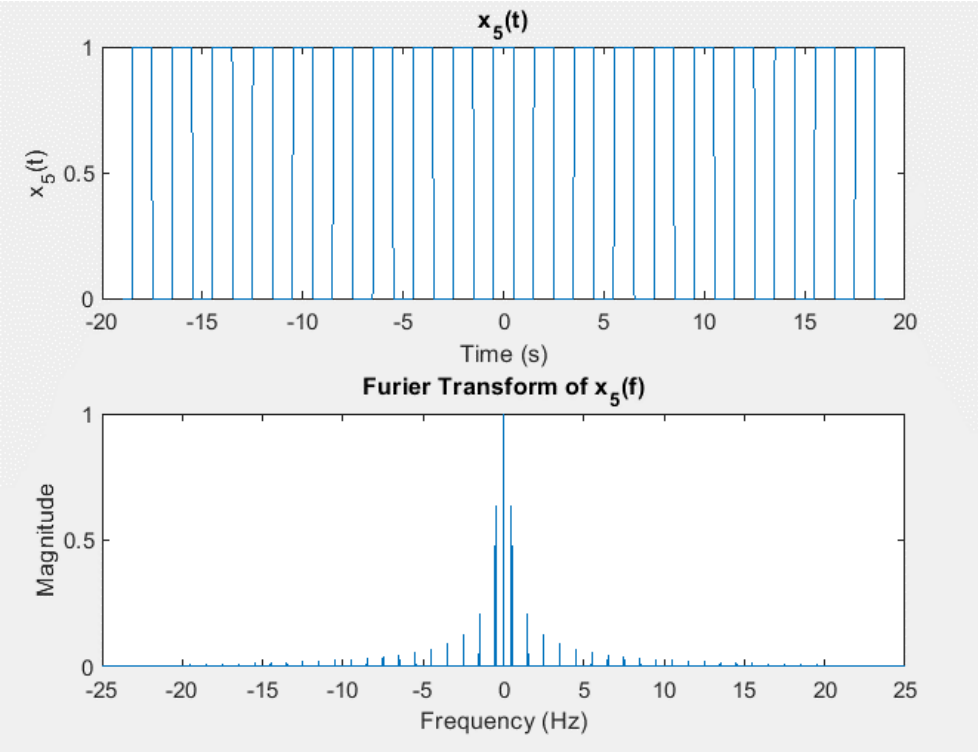
ضرایب به دلیل اینکه در متلب نمودار نرمال شده رسم شده است، تاثیری ندارند؛ همچنین با جایگذاری $w = 2\pi f$ به عبارت $\delta(f+15)e^{-j\frac{\pi}{4}} + \delta(f-15)e^{j\frac{\pi}{4}}$ می‌رسیم و چون نمودار از فرکانس 0 تا 100 رسم شده است، $\delta(f+15)$ را به $\delta(f-15)$ تبدیل می‌کنیم و همانطور که می‌بینیم در نمودار اندازه نیز در دو نقطه 15 و 85 قله به وجود آمده است؛ برای بدست آوردن فاز، ضریب j در توان e را در δ های بدست آمده ضرب می‌کنیم و به عبارت زیر می‌رسیم.

فاز:$\delta(f-15)(\frac{\pi}{4}) + \delta(f-85)(-\frac{\pi}{4})$

و همانطور که در نمودار فاز بر حسب فرکانس نیز می‌بینیم، دو قله مثبت و منفی در فرکانس‌های 15 و 85 دیده می‌شود.

قسمت پنج

5.الف و ب:



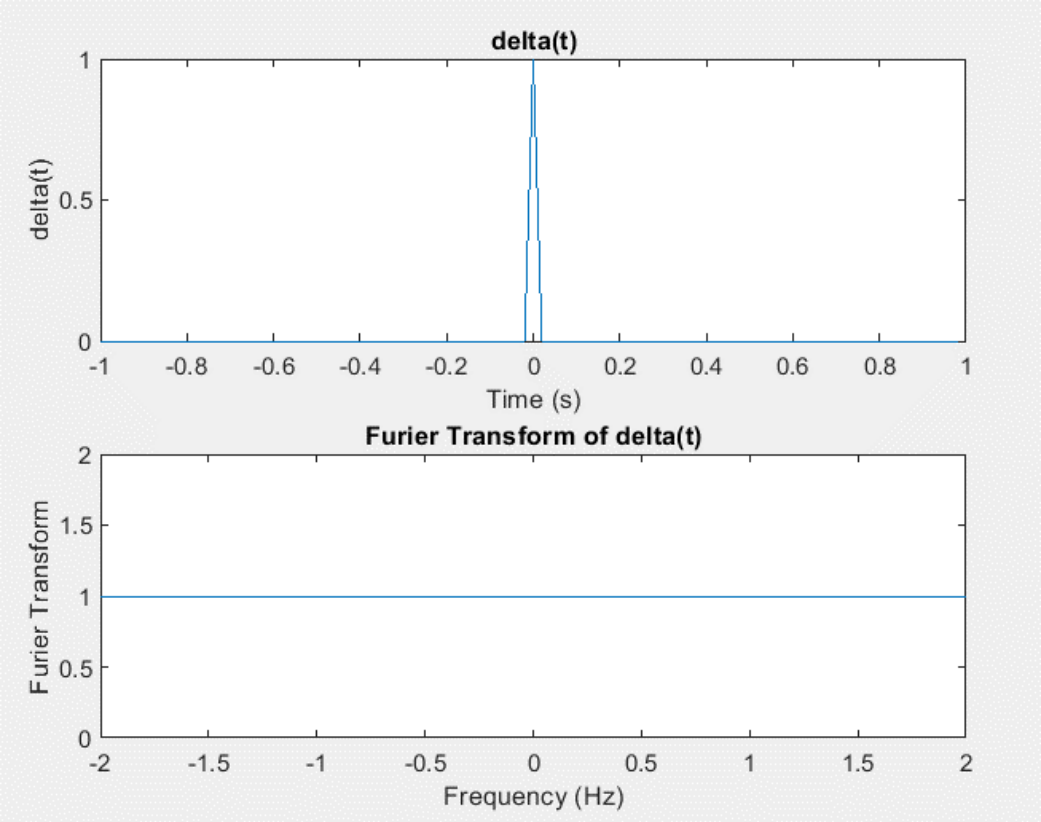
5.ج

از آنجایی که این سیگنال متناوب است و تبدیل فوریه سیگنال های متناوب بصورت$\sum_k 2\pi a_k \delta(\omega - \omega_k)$ است که از مجموع تعدادی تابع ضربه بدست می آید، بنابراین شکل نمودار بدست آمده نیز مجموع ضرایبی (a_k) از توابع ضربه است. طبق نمودار موجود، بین صفر تا پنج، 5 تابع ضربه وجود دارد که با بزرگنمایی خواهیم دید که فاصله هر یک از هم برابر یک واحد است.

- **بخش دوم**

قسمت پک

1.الف و ب:

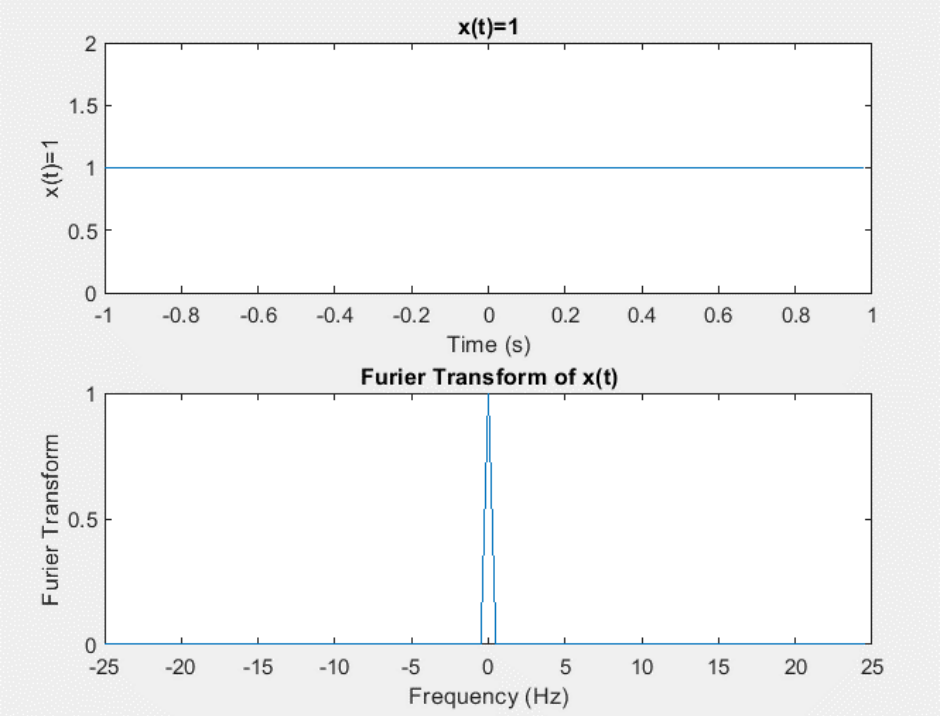


1.ج

تغییرات تابع دلتا در حوزه زمان بسیار شدید است (همانطور که گفته شد توابعی که ناپیوستگی دارند، شدیدترین تغییرات را در حوزه زمان دارند) و همانطور که می‌بینیم تبدیل فوریه‌ی آن که ما را از حوزه زمان به حوزه فرکانس می‌برد برابر تابع ثابت 1 است و فرکانس آن از منفی بینهایت تا مثبت بینهایت گسترده است و تابع ما دارای فرکانس‌های بالاتری است. برای محاسبه تئوری تبدیل فوریه تابع ضربه، همانطور که می‌دانیم این تابع فقط در 0 مقدار دارد و مقدارش برابر با 1 است پس در فرمول تبدیل فوریه به جای t، 0 می‌گذاریم و می‌دانیم e به توان 0 برابر با 1 است پس حاصل نهایی برابر با 1 است.

قسمت دو

2.الف و ب:



2.ج

تابع ثابت 1 هیچ تغییری در حوزه زمان ندارد و همچنین ناپیوستگی ندارد پس این تابع در حوزه فرکانس با فرکانس‌های پایین قابل بیان است و همانطور که می‌بینیم تبدیل فوریه تابع ثابت 1 برابر با یک ضربه در فرکانس 0 است. برای بدست آوردن تبدیل فوریه تابع ثابت 1 می‌توان آن را به صورت e به توان 0 می‌نویسیم و با جایگذاری در فرمول و اینکه می‌دانیم تبدیل فوریه تابع exponential به صورت $\delta(w - w0)$ می‌شود که w0 در آن برابر با 0 است.

- **بخش سوم**

(3-1

برای ایجاد Mapset ابتدا یک cell به اندازه 32*2 تشکیل میدهیم که ردیف اول متشکل از حروف کوچک انگلیسی، فاصله، نقطه، ویرگول، علامت تعجب، سمی کالن و کوتیشن میباشد که در مجموع 32 کاراکتر را تشکیل میدهد و در ردیف دوم این کاراکتر هارا شماره گذاری میکنیم که به هر کاراکتر 5 بیت مرتبط میشود.

```
function setGlobal
    global Mapset
    Mapset= cell(2, 32);
    letters = 'a':'z';
    letters = [letters, ' ', '.', ',', '!', ';', '"'];
    for i=1:32
       Mapset{1, i} = letters(i);
       Mapset{2, i} = dec2bin(i -1, 5);
    end
end
```

(3-2

در این قسمت باید متن پیام را با توجه به سرعت داده شده کد گذاری کنیم. در ابتدا متن دریافتی را با توجه به شماره گذاری کاراکتر ها در Mapset، به دنباله ای از اعداد باینری تبدیل میکنیم. در مرحله بعد باید با توجه به سرعت، آرایه ای از binary number ها تشکیل بدهیم، بصورتیکه اگر سرعت 1 باشد، این مجموعه متشکل از ارقام 0 و 1، اگر سرعت 2 باید، این مجموعه متشکل از ارقام 00، 01، 10 و 11 خواهد بود و به همین صورت به تناسب سرعت این مجموعه را تشکیل میدهیم. سپس

```
binaryNums = strings([1, 2^speed]);
signals = cell(1, 2^speed);
ratio = 1;
for i=1:2^speed
    binaryNums(i) = dec2bin(i - 1, speed);
    t = tstart:ts:tend-ts;
    if i == 1
        signals{1,i} = zeros(1, 100);
    else
        signals{1,i} = (ratio / (2^speed - 1)) * sin(2*pi*t);
        ratio = ratio + 1;
    end
    tstart = tend;
    tend = tend + 1;
end
```

یک آرایه ای از سیگنال ها باید تشکیل شود که تعداد آن مانند قسمت قبل وابسته به سرعت و در اردر 2^{speed} است. این سیگنال ها نمودار های $\sin(2\pi t)$ ای هستند که هرکدام دارای ضریب $\frac{ratio}{2^{speed}-1}$ خواهند بود.

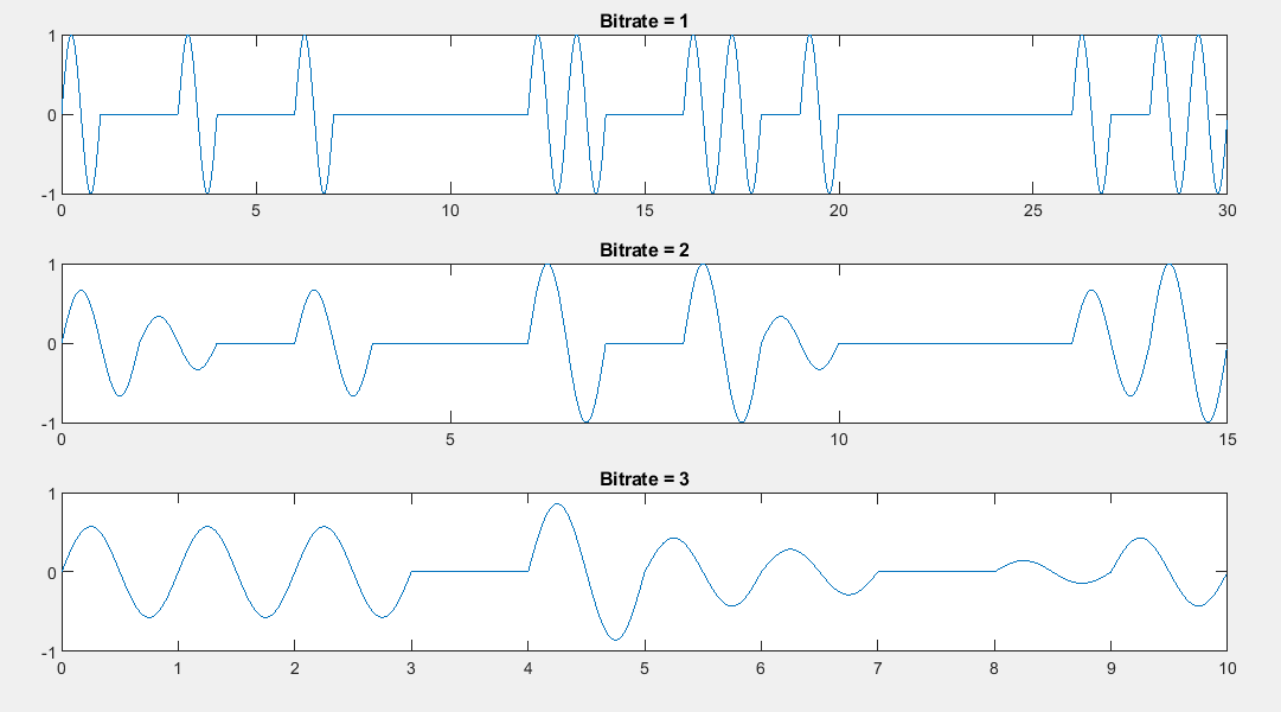
```
tokenCount = floor(length(binaryStr)/speed);
result = [];
startIndex = 1;
for i=1:tokenCount
    endIndex = startIndex + speed - 1;
    part = extractBetween(binaryStr, startIndex, endIndex);
    for j=1:2^speed
        if binaryNums(j) == part
            result = [result, signals{1,j}];
        end
    end
   startIndex = startIndex + speed;
end
```

سپس برای تبدیل پیام کدگذاری شده به سیگنال های سینوسی ابتدا تعداد دسته بندی اعداد باینری را که وابسته به سرعت است بدست آورده و در حلقه ای به آن تعداد، رقم ها را جدا کرده و هربار در آرایه binary number مقدار آنرا جستجو میکنیم، زمانی که این ارقام با هم match شدند، باید سیگنال مرتبط با آن رقم را از آرایه signal به آرایه result اضافه کنیم. در نهایت آرایه result متشکل از سیگنال های سینوسی ای خواهد بود که هر کدام مرتبط به ارقام دسته بندی شده در پیام کدگذاری است.

```
currentLength = length(str) *5;
modSpeed = mod (currentLength, speed);
while speed < 5 && modSpeed ~= 0
    binaryStr = [binaryStr, dec2bin(0,1)];
    currentLength = currentLength + 1;
   modSpeed = mod (currentLength, speed);
end
```

در این قسمت از کد، درصورتی که تعداد بیت‌های یک رشته بر bitrate بخش‌پذیر نبود به تعدادی که بخش پذیر باشد، به انتهای آن بیت 0 اضافه می‌کنیم؛ توجه شود که اینکار در قسمت decoding مشکلی ایجاد نمی‌کند زیرا در آنجا از سمت چپ 5 بیت، 5 بیت جدا می‌کنیم و بیت‌های اضافه‌ی انتهایی در نظر گرفته نمی‌شود.

(3-3



همانطور که در تصویر قابل مشاهده است با افزایش سرعت باید کدگذاری بیشتری روی بیت ها انجام شود بنابراین در هر ثانیه تعداد بیت های بیشتری ارسال شود. بطور مثال در کلمه "signal" که یک کلمه 6 حرفی است، و پس از کدگذاری به 30 بیت تبدیل میشود، اگر بخواهیم با سرعت یک انتقال را انجام دهیم، بنابراین در هر ثانیه 1 بیت جابجا میشود که در مجموع 30 ثانیه زمان خواهد برد. اما اگر بخواهیم با سرعت دو انتقال را انجام دهیم، در هر ثانیه 2 بیت جابجا خواهد شد که در مجموع 15 ثانیه زمان خواهد برد و به همین ترتیب با افزایش سرعت زمان انتقال کاهش میابد.

(3-4

حال در این قسمت باید سیگنال های دریافتی از قسمت قبل را decode کرده و پیام انتقالی از طریق سیگنال را به متن اولیه تبدیل کنیم. برای اینکار ابتدا سیگنال های دریافتی را با توجه به fs تقسیم میکنیم تا هر قسمت از پیام که بصورت سیگنال سینوسی است، بدست بیاید، سپس از تابع interp1 استفاده میکنیم که تابعی است که با دریافت نقاط، یه تابع را رسم میکند. ما نیز با استفاده از نقاط دریافت شده از سیگنال ها اقدام به رسم تابع های سینوسی میکنیم و سپس با استفاده از تابع trapz، correlation تابع سینوسی بدست آمده را با $2\sin(2\pi t)$ میگیریم و برای شبیه سازی بیشتر آنرا در 0.01 ضرب میکنیم. حال اختلاف correlation هارا بدست می آوریم تا بتونیم کد مناسب هر سیگنال را از آرایه binary number بدست آوریم.

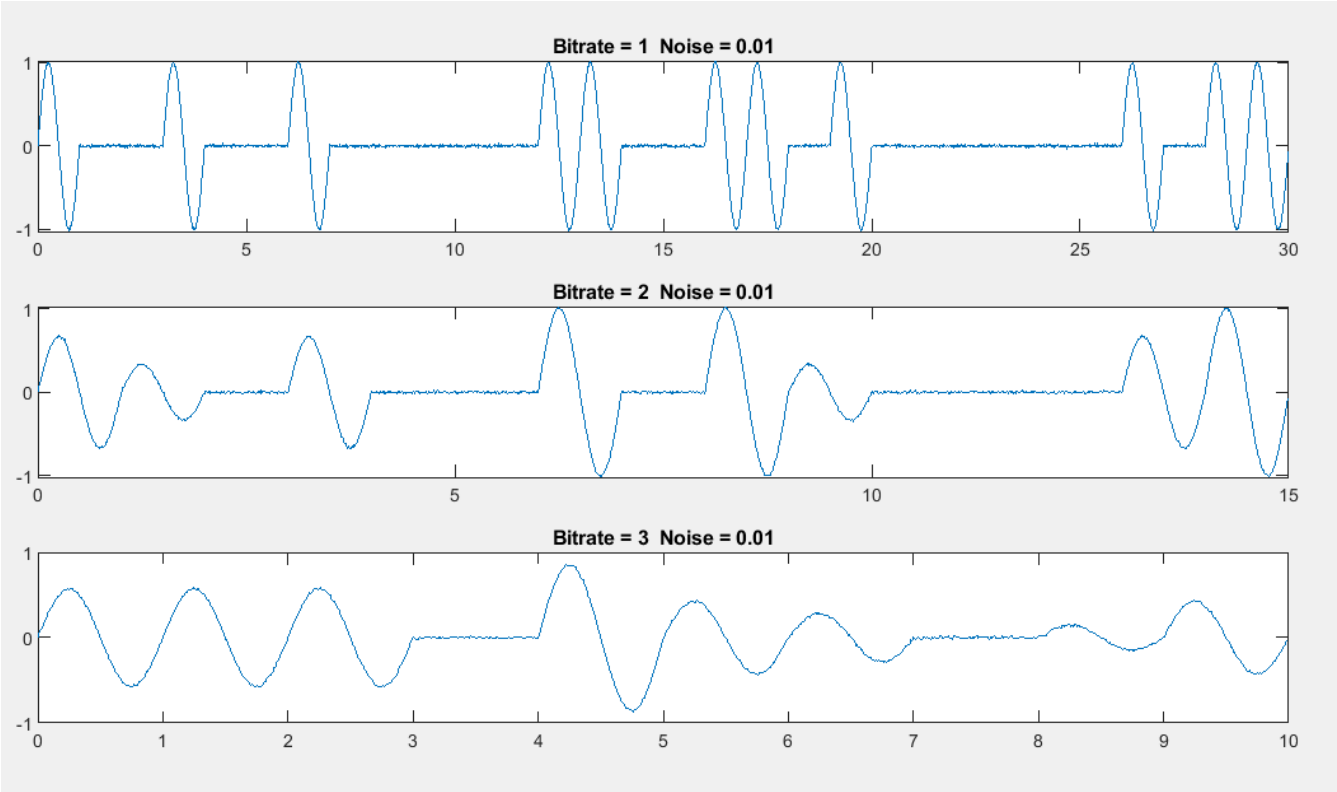
```
sinFunc = interp1(t, decodedSignal{1, i}, t);
Y = 2 * sin(2*pi*t);
plot(sinFunc)
newY = sinFunc .* Y;
corrPart = 0.01* trapz(newY);
ratio = 1;
for k=1:2^speed
    dif = ((ratio-1) / (2^speed - 1));
    corrDif = corrPart - dif;
    if corrDif < 0
        corrDif = dif - corrPart;
    end
    if corrDif <= 1 / (2* (2^speed -1))
        binString(i) = binaryNums(k);
    end
    ratio = ratio + 1;
 end
```

در نهایت مجموعه دنباله بدست آمده که متشکل از اعداد باینری است را به قسمت های 5 تایی تبدیل کرده و کاراکتر مرتبط به هر عدد باینری را از Mapset تشکیل داده بدست می آوریم. درنهایت پیام دریافتی به صورت سیگنال، به متن تبدیل میشود.

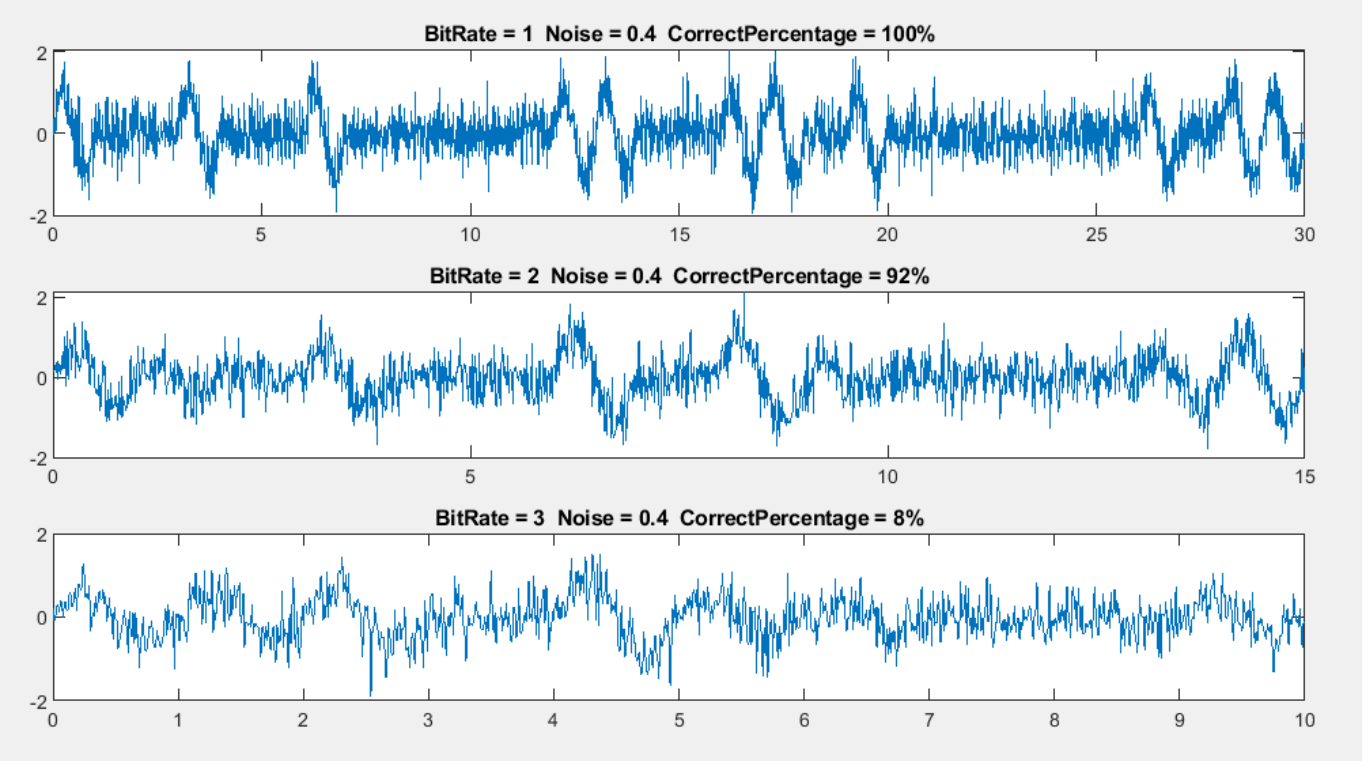
```
while (i + 5) <= (strlength(finalString))
    part = extractBetween(finalString,i+1,i+5);
    for j=1:length(Mapset)
        if part == Mapset{2, j}
            finalOutput(counter) = Mapset{1,j};
            counter = counter + 1;
            break;
        end
    end
    i = i + 5;
end
```

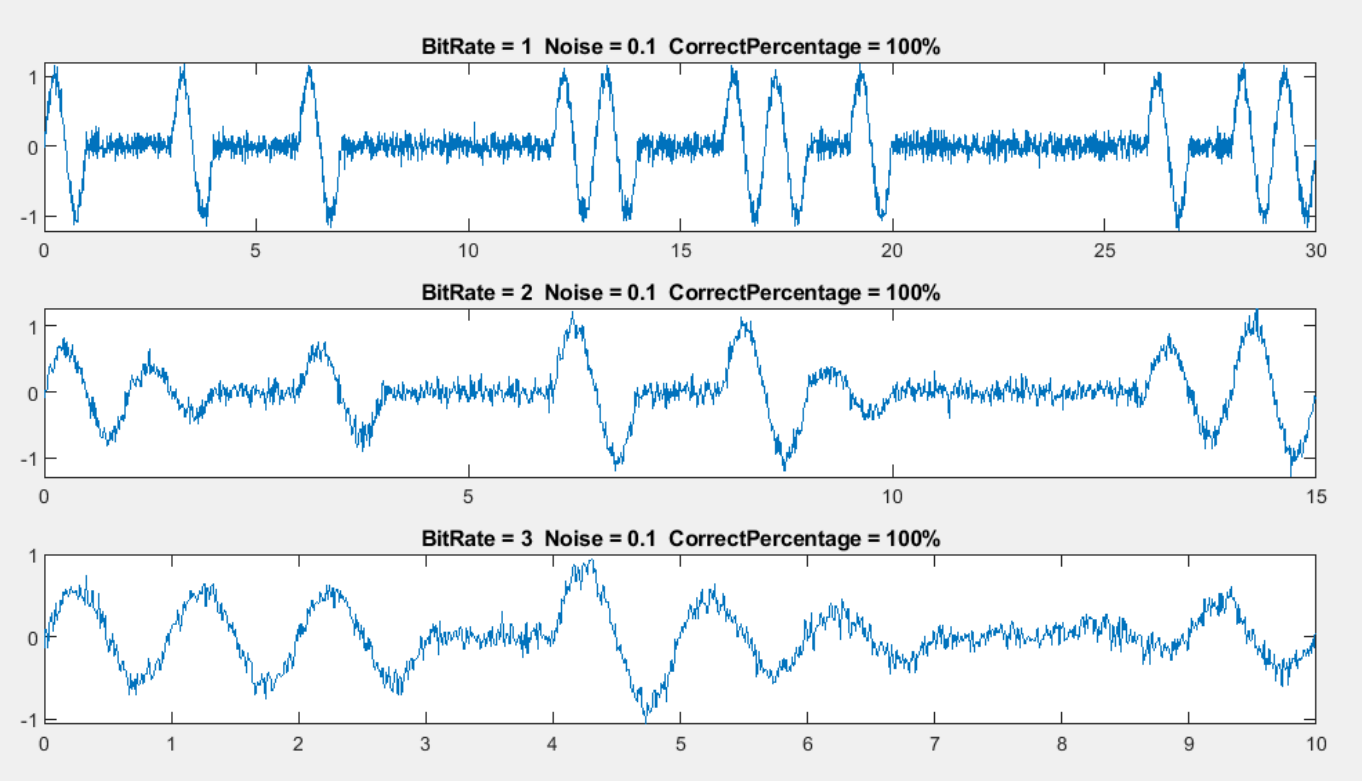
```
BitRate = 1 | Origianl Text = signal | Decoded Text = signal
BitRate = 2 | Origianl Text = signal | Decoded Text = signal
BitRate = 3 | Origianl Text = signal | Decoded Text = signal
```

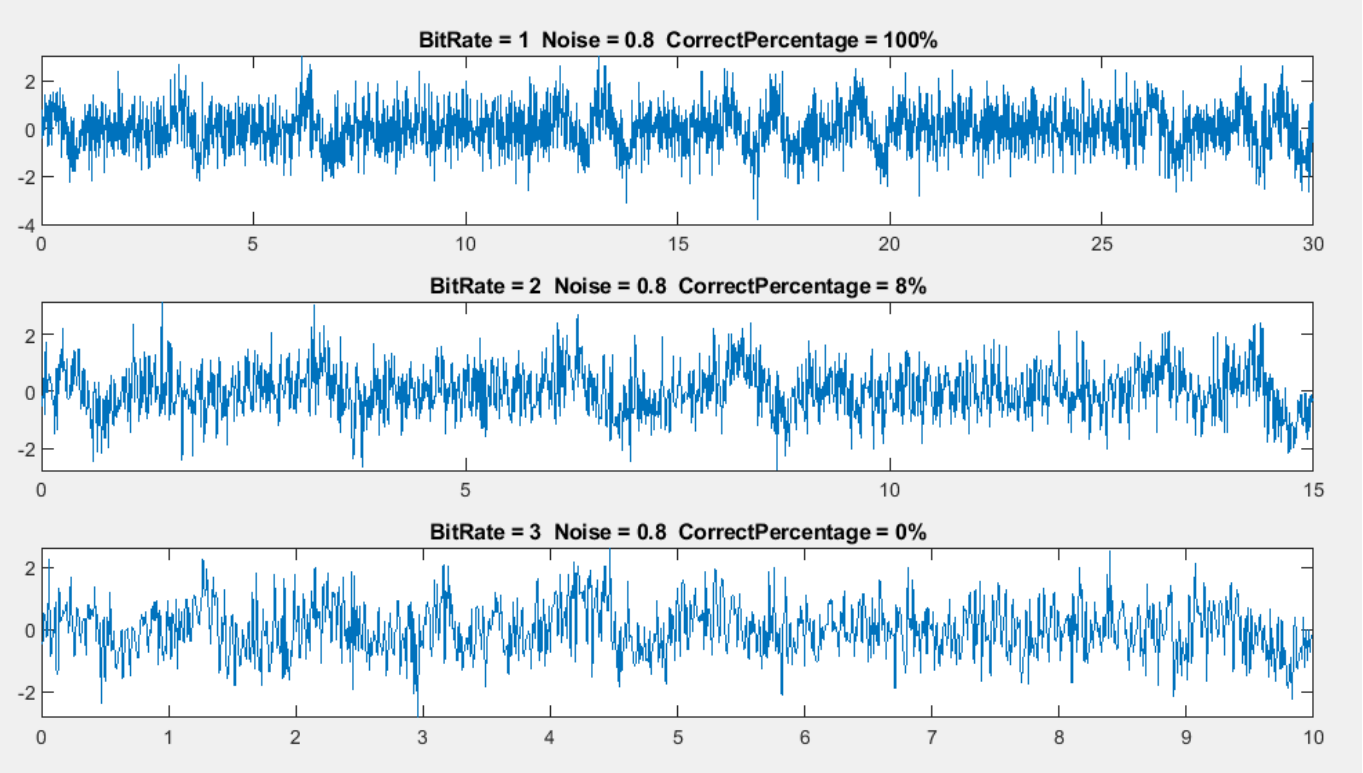
(3-5

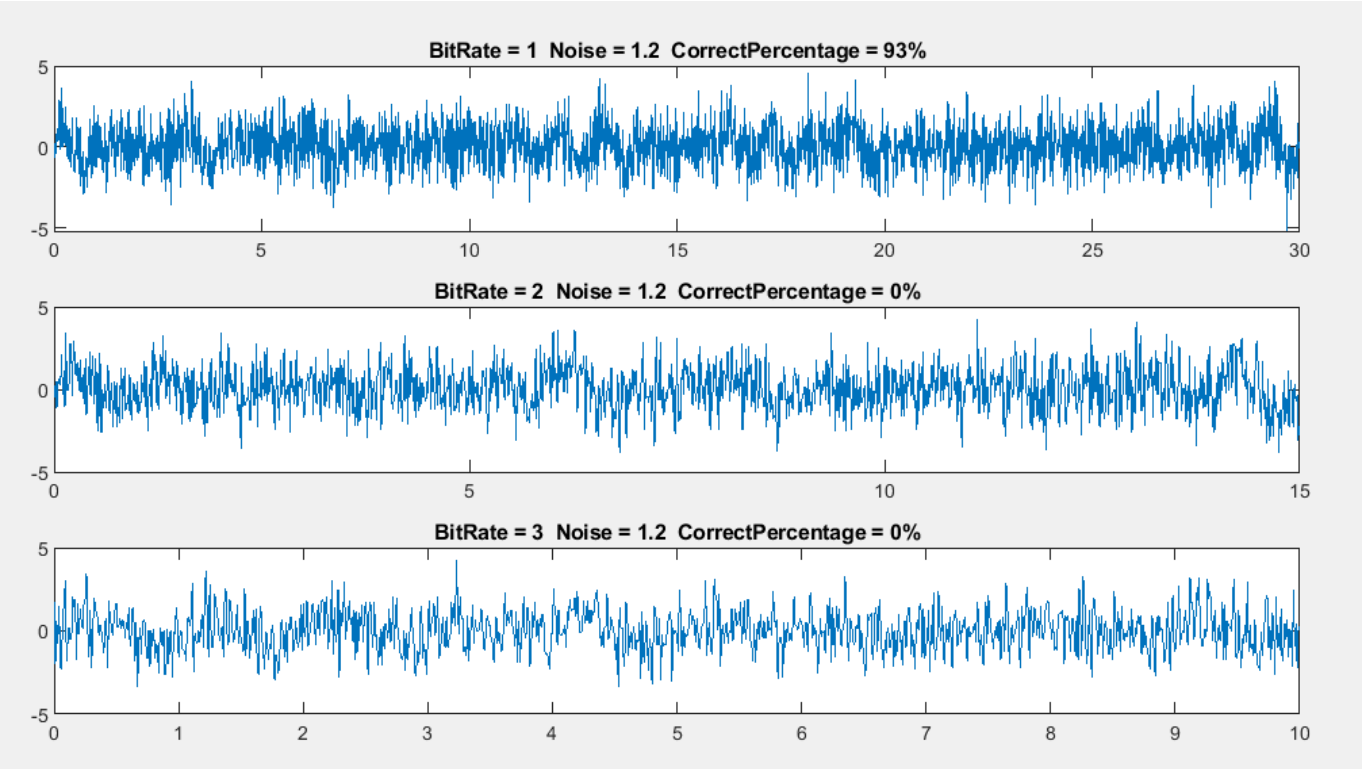


پیام استخراج شده برای هر سه bitrate با نویز 0.01 برابر متن داده شده اولیه (signal) است.


BitRate = 1 Noise = 0.4 CorrectPercentage = 100%
BitRate = 2 Noise = 0.4 CorrectPercentage = 92%
BitRate = 3 Noise = 0.4 CorrectPercentage = 8%


BitRate = 1 Noise = 0.1 CorrectPercentage = 100%
BitRate = 2 Noise = 0.1 CorrectPercentage = 100%
BitRate = 3 Noise = 0.1 CorrectPercentage = 100%





همانطور که در شکل ها نیز قابل مشاهده است، سیگنال با سرعت 1 در برابر سیگنال ها با سرعت 2 و 3 نسبت به نویز مقاوم تر عمل میکند و سیگنال 2 در برابر سیگنال 3 نیز نسبت به نویز مقاوم تر عمل میکند.

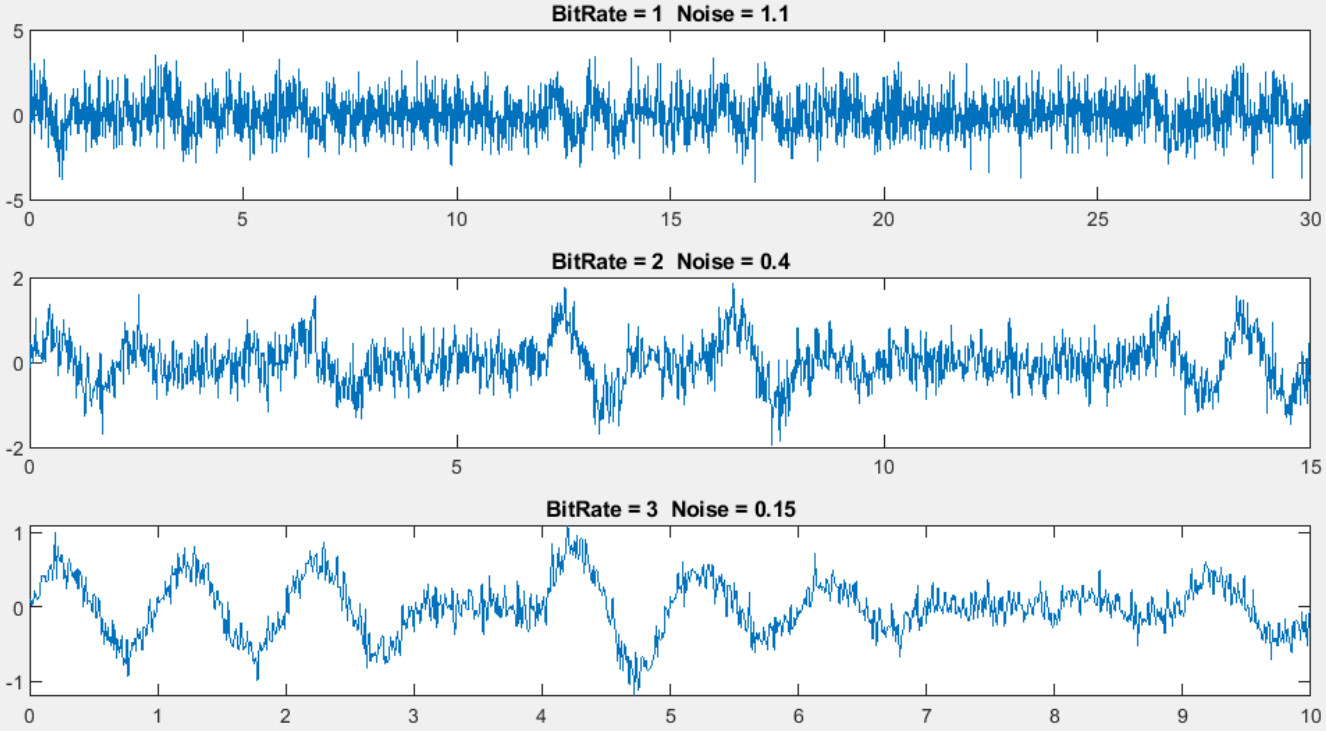
تابع بدست آوردن درصد درستی به این شکل تعریف شده است که به ازای یک نویز مشخص، 100 بار عملیات decode را انجام می‌دهد و در هر بار به تعداد bitrate از بیت‌های باینری تولید شده، جدا می‌کند و درصورتی که مجموعه‌ی جدا شده با بیت‌های اصلی برابر بود، یک واحد به تعداد تشخیص‌های درست اضافه می‌کند و تعداد کل تشخیص‌ها را برابر با تعداد بیت‌ها تقسیم بر bitrate در نظر می‌گیریم و میانگین تشخیص‌های درست در 10000 بار اجرا را پیدا می‌کنیم.

```
correctCount = 0;
startIndex = 1;
tokenCount = floor(length(binaryDecodedStr)/bitRate);
for i=1:tokenCount
    endIndex = startIndex + bitRate - 1;
    decodedPart = extractBetween(binaryDecodedStr, startIndex, endIndex);
    originalPart = extractBetween(binaryOriginalStr, startIndex, endIndex);
    if strcmp(decodedPart , originalPart)
        correctCount = correctCount + 1;
    end
    startIndex = startIndex + bitRate;
end
correctBitPercentage(counter) = (correctCount / tokenCount) * 100;
```

3-7)

برای فهمیدن ماکسیمم نویزی که در آن سیگنال مقاوم است، تابعی مینویسیم که تا زمانیکه کلمه ورودی با خروجی برابر است، مقدار واریانس را در هر مرحله 0.05 افزایش دهد و هربار به تعداد 50 بار جواب را با هر مقدار Std چک کند و زمانیکه شرط برقرار نبود، مقدار نویز را ذخیره کند. در نتیجه آن همانطور که در قسمت قبلی نتیجه گیری شد، مقاومت سیگنال با سرعت 1 از باقی سیگنال ها بیشتر است.

```
while true
    if isWrong
        break;
    end
    finalDecodedSignal{1, bitRate} = encoded;
    finalDecodedSignal{2, bitRate} = std;
    for counter=1:50
        normal_noise = std*randn(1, length(t));
        encoded = encodedWithoutNoise + normal_noise;
        decoded = decoding_amp(encoded, bitRate);
        if ~strcmp(str, decoded)
            isWrong = true;
            break
        end
        finalDecodedSignal{1, bitRate} = encoded;
        finalDecodedSignal{2, bitRate} = std;
        finalDecodedSignal{3, bitRate} = decoded;
    end
    std = std + 0.05;
end
```



BitRate = 1 | Noise = 1.1 | Variance = 1.21

BitRate = 2 | Noise = 0.4 | Variance = 0.16

BitRate = 3 | Noise = 0.15 | Variance = 0.0225

(3-8

در اینجا مقدار دامنه ای که بکار رفته است برابر یک است، اگر قدرت فرستنده ی ما بیشتر می بود می توانستیم دامنه ی سیگنال بیشتری داشته باشیم و بنابراین فاصله threshold هایی که برای تصمیم گیری در نظر گرفته بودیم بیشتر می شد و در نتیجه به نویز حساسیت کمتری ایجاد می شد.

- **بخش چهارم:**

(4-1

توضیح و کد این بخش عینا مثل قسمت 1-3 است.

(4-2

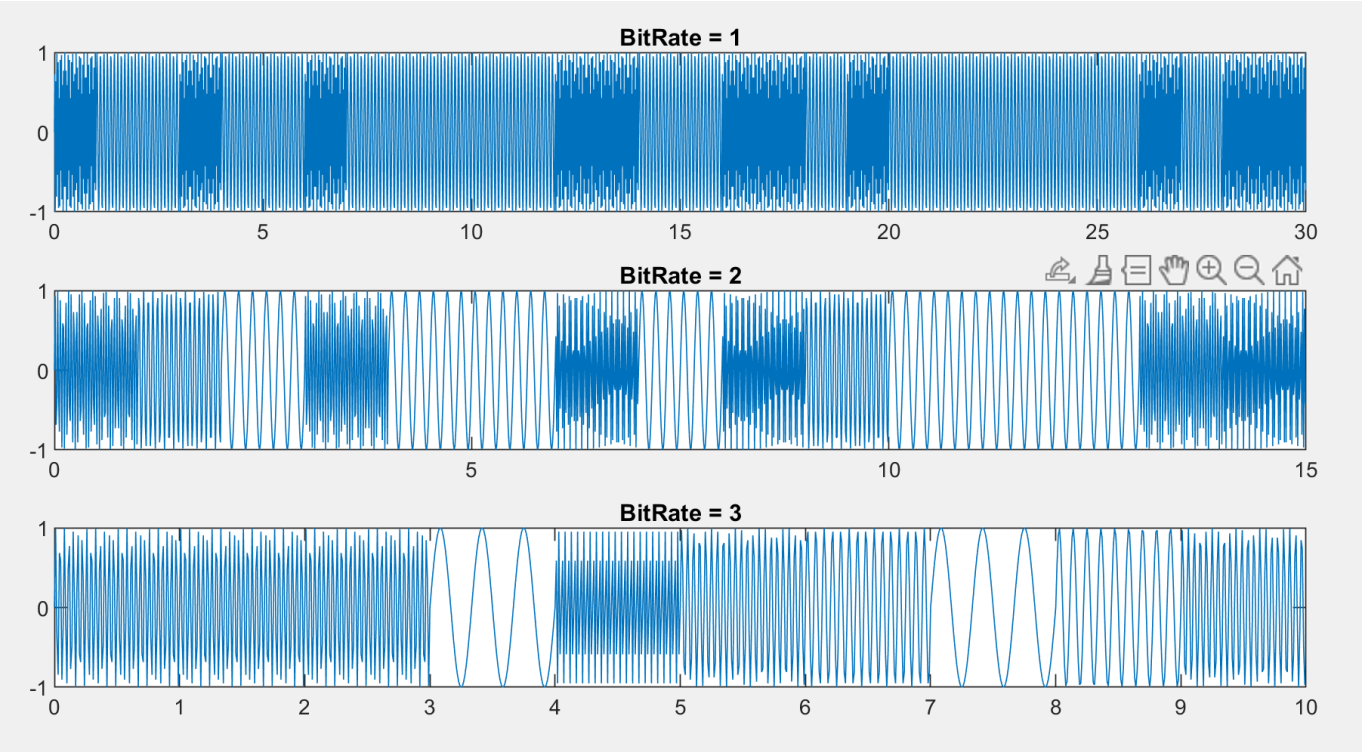
هر گام (اختلاف هر دو فرکانس متوالی) را از تقسیم 50 بر تعداد فرکانس‌های موردنیاز برای تولید بدست می‌آوریم و فاصله‌ی فرکانس آخر از عدد 50 و فاصله‌ی فرکانس اول از 0 نیز را برابر قرار می‌دهیم و برای بدست آوردن فرکانس هر مجموعه از بیت‌ها، فرکانس قبلی را با گام تولید شده جمع می‌کنیم.

```
step = 50 / (2^speed);
freq = (50 - (2^speed - 1) * step ) / 2;
for i=1:2^speed
    binaryNums(i) = dec2bin(i - 1, speed);
    t = tstart:ts:tend-ts;
    signals{1,i} = sin(2*pi*(floor(freq))*t);
    freq = freq + step;
    tstart = tend;
    tend = tend + 1;
end
```

در این قسمت نیز داده‌های باینری را هربار به تعداد bitrate جدا می‌کنیم و هرکدام را با تک تک اعضای اعداد باینری مقایسه می‌کنیم و درصورتی که با یکی از آنها برابر بود، مقدار سیگنال متناظر با آن عدد را در تیجه می‌ریزیم.

```
tokenCount = floor(length(binaryStr)/speed);
result = [];
startIndex = 1;
for i=1:tokenCount
    endIndex = startIndex + speed - 1;
    part = extractBetween(binaryStr, startIndex, endIndex);
    for j=1:2^speed
        if binaryNums(j) == part
            result = [result, signals{1,j}];
            break;
        end
    end
   startIndex = startIndex + speed;
end
```

(4-3



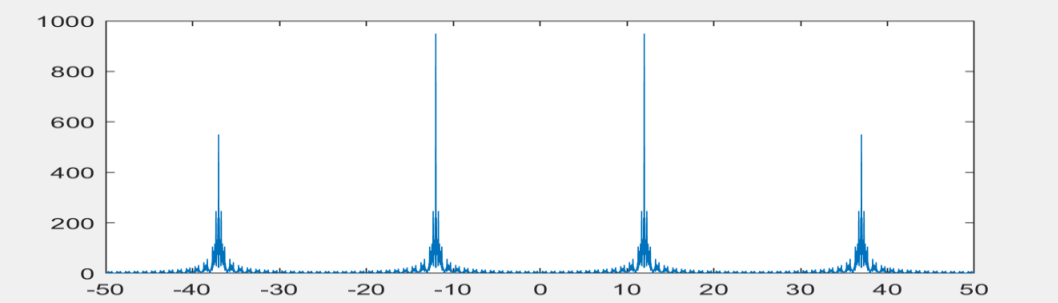
همانطور که مشاهده می‌شود با افزایش bitRate، تنوع فرکانس‌های تولیدی بیشتر می‌شود و همچنین سیگنال در زمان کمتری تولید می‌شود؛ کلمه‌ی 6 حرفی ’signal‘ را در نظر بگیرید که هر حرف در آن 5 بیت است، پس با bitrate = 1، 30 ثانیه، با bitrate = 2، 15 ثانیه و با bitrate = 3، 10 ثانیه به طول می‌انجامد.

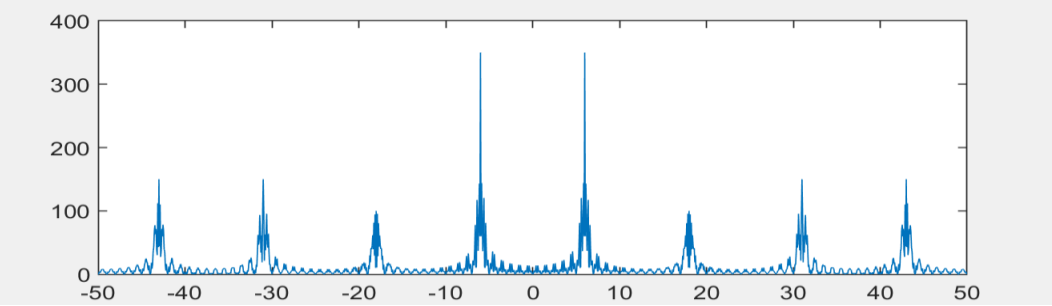
(4-4

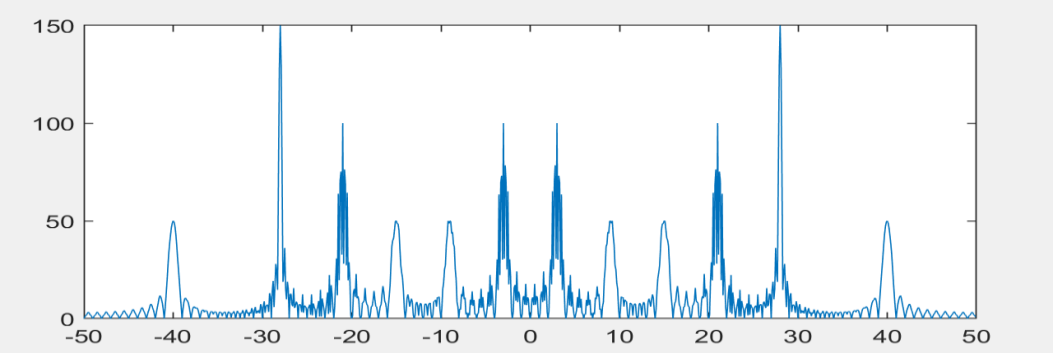
```
for i=1:tokenCount
    endIndex = i * fs;
    decodedSignal{1, i} = codedStr(startIndex:endIndex);
    startIndex = endIndex + 1;
    t = tstart:ts:tend-ts;
    N = length(t);
    f = -fs/2:fs/N:fs/2-fs/N;
    x = interp1(t, decodedSignal{1, i}, t);
    xF = fft(x);
    xFshifted = abs(fftshift(xF));
    plot(f, xFshifted);
    [~, indexAtMaxY] = max(xFshifted);
    peakFreq = f(indexAtMaxY(1));

    if peakFreq < 0
        peakFreq = -peakFreq;
    end
    for k=1:2^speed
        threshold = step / 2;
        dif = peakFreq - allFreqs(k);
        if dif < 0
            dif = -dif;
        end
        if (dif <= threshold)
            binString(i) = binaryNums(k);
            break;
        end
     end
    tstart = tend;
    tend = tend + 1;
end
```

در این قسمت داده‌های نمودار را 100تا، 100تا جدا می‌کنیم (چون در هر ثانیه 100 داده وجود دارد و fs = 100 است) و از هر قسمت تبدیل فوریه می‌گیریم و شیفت‌داده‌ی آن را نیز محاسبه می‌کنیم؛ حال ماکسیمم داده‌ها در هر قسمت را پیدا می‌کنیم و فرکانس متناظر با آن را نیز پیدا می‌کنیم (چون فرکانس‌ها قرینه‌ی هم هستند، اولین فرکانس قله پیدا شده، منفی است و باید مثبت شده‌ی آن را در نظر بگیریم؛ حال فرکانس بدست آمده را با تمامی فرکانس‌های موجود برای bitrate مقایسه می‌کنیم و درصورتی که اختلاف آنها کمتر از نصف فاصله‌ی هر دو فرکانس متوالی بود، آن را به عنوان جواب انتخاب می‌کنیم.

در زیر نمودار اندازه تبدیل فوریه بر حسب فرکانس برای3و 2 و1 = bitrate نمایش داده شده است و همانطور که می‌بینیم نقاط ماکسیمم در فرکانس‌های تعیین شده اتفاق افتاده است و هرچه bitrate افزایش یابد، فاصله‌ی دو فرکانس متوالی کمتر می‌شود.







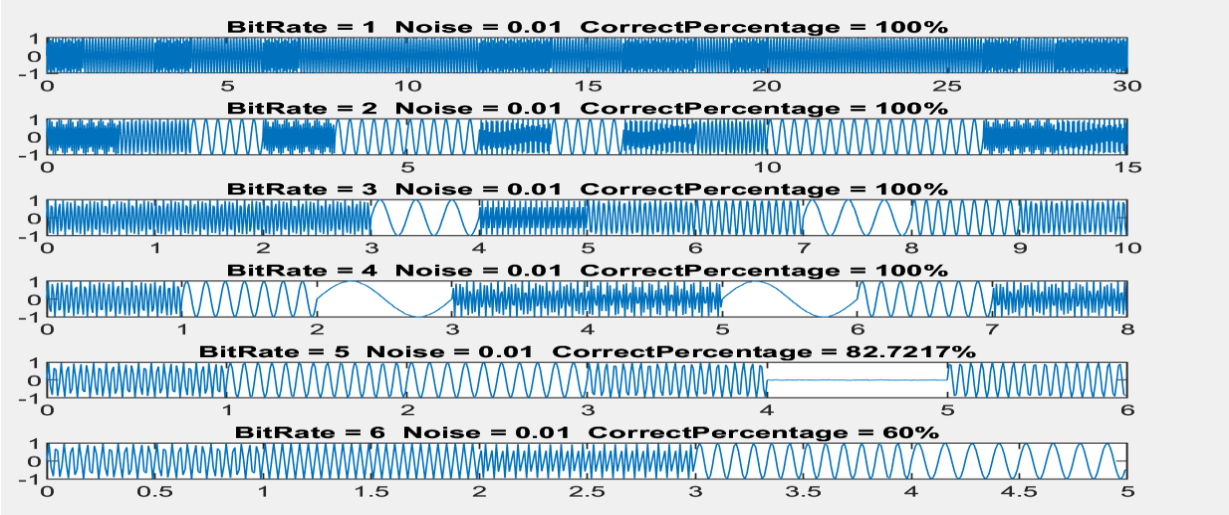

```
figure("Name", "signalPlots")
for bitRate=1:3
    encoded = coding_amp(str, bitRate);
    fs = 100;
    ts = 1/fs;
    tstart = 0;
    tend = length(encoded) /fs;
    t = tstart:ts:tend-ts;
    subplot(3,1,bitRate);
    plot(t, encoded);
    title(['BitRate = ', num2str(bitRate)]);
    decoded = decoding_amp(encoded, bitRate);
    disp(['BitRate = ', num2str(bitRate), ' | Origianl Text = ', str ,' | Decoded Text = ', num2str(decoded)]);
end
```

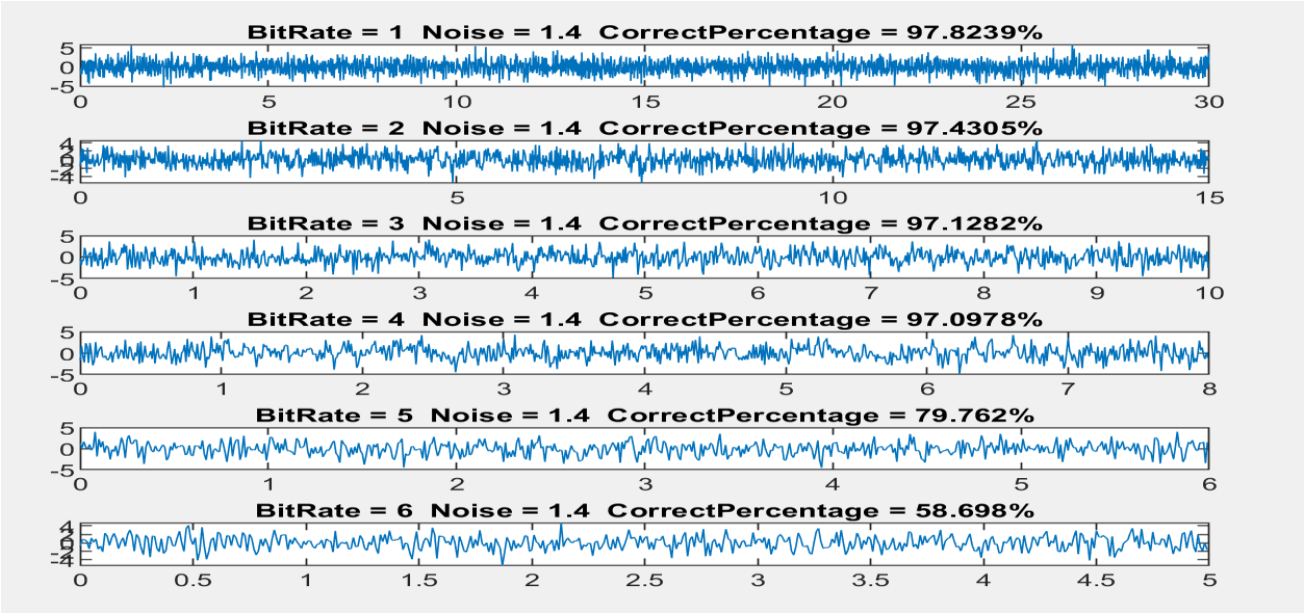
در این قطعه کد، برنامه را به ازای bitrate های 1 تا 3 امتحان می‌کنیم و نتیجه‌ی خروجی درستی آن، در زیر آمده است:

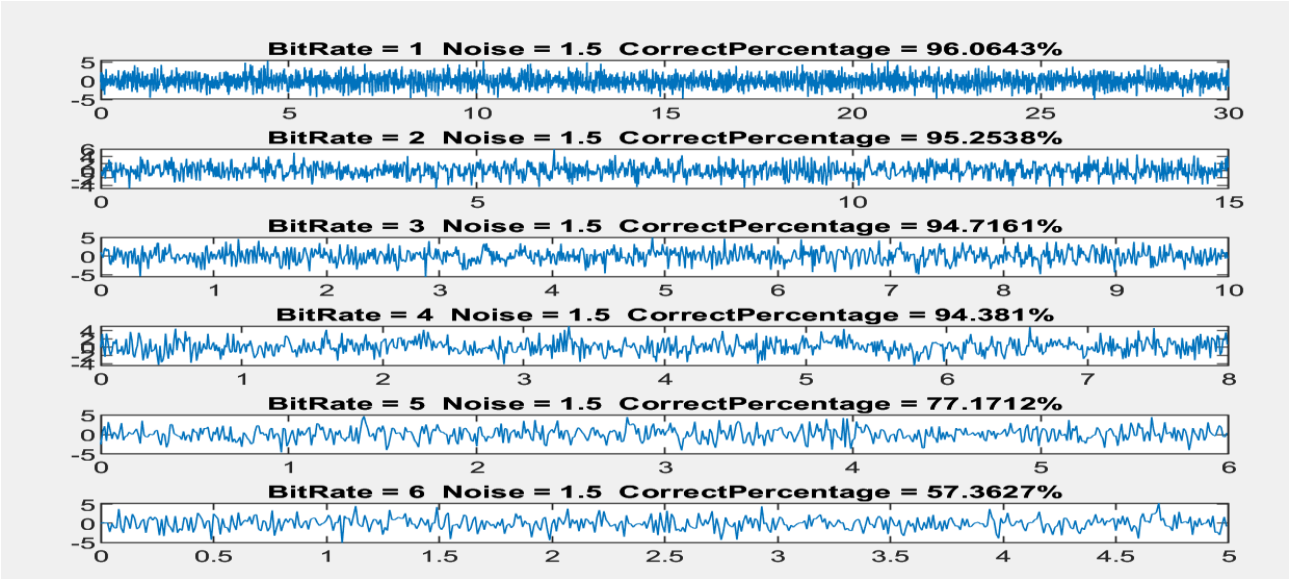
```
BitRate = 1 | Origianl Text = signal | Decoded Text = signal
BitRate = 2 | Origianl Text = signal | Decoded Text = signal
BitRate = 3 | Origianl Text = signal | Decoded Text = signal
```

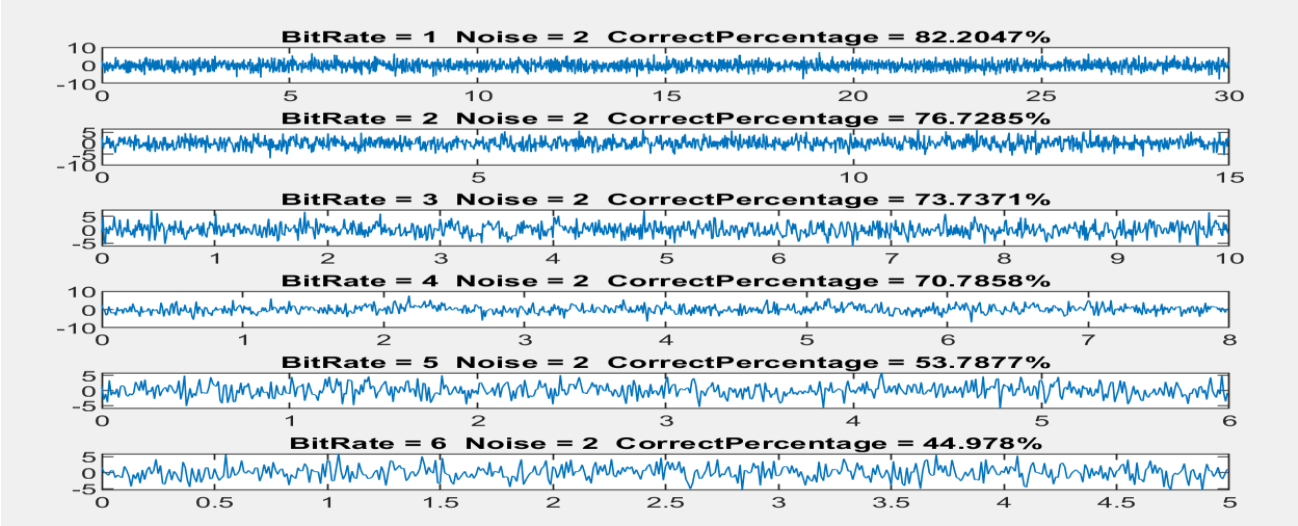
(4-5

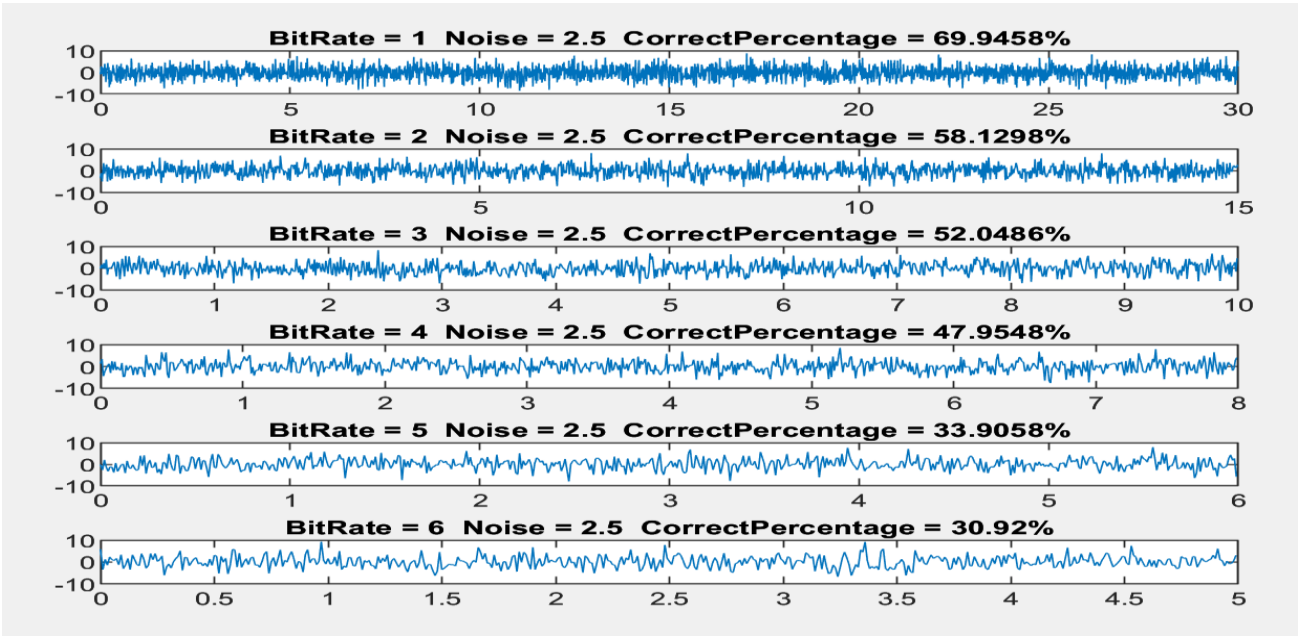
همانطور که در نمودار زیر قابل مشاهده است، با قراردادن نویز 0.01، کلمه‌ی "signal" با هر سه bitrate 1، 2 و 3 پاسخ درست می‌دهد و با bitrate های 5 و 6 درصد درستی کاهش می‌یابد.




BitRate = 1 Noise = 1.4 CorrectPercentage = 97.8239%
BitRate = 2 Noise = 1.4 CorrectPercentage = 97.4305%
BitRate = 3 Noise = 1.4 CorrectPercentage = 97.1282%
BitRate = 4 Noise = 1.4 CorrectPercentage = 97.0978%
BitRate = 5 Noise = 1.4 CorrectPercentage = 79.762%
BitRate = 6 Noise = 1.4 CorrectPercentage = 58.698%


BitRate = 1 Noise = 1.5 CorrectPercentage = 96.0643%
BitRate = 2 Noise = 1.5 CorrectPercentage = 95.2538%
BitRate = 3 Noise = 1.5 CorrectPercentage = 94.7161%
BitRate = 4 Noise = 1.5 CorrectPercentage = 94.381%
BitRate = 5 Noise = 1.5 CorrectPercentage = 77.1712%
BitRate = 6 Noise = 1.5 CorrectPercentage = 57.3627%

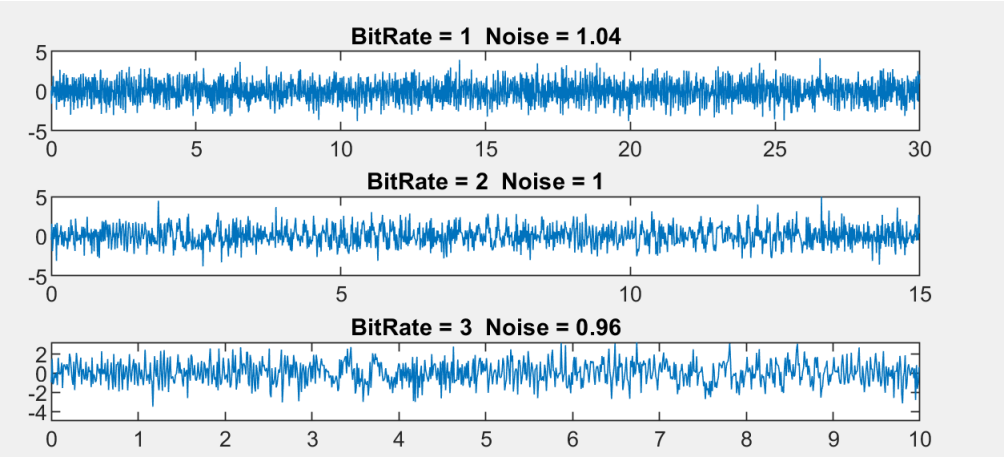




همانطور که می‌بینیم درصد درستی بیت‌های باینری تولید شده با افزایش bitrate کاهش می‌یابد و دلیل آن این است که فرکانس‌های بیشتری تولید می‌شود و فاصله‌ی هر دو فرکانس متوالی کمتر می‌شود پس با اضافه شدن نویز امکان اشتباه بیشتر می‌شود.

تابع بدست آوردن درصد درستی مانند تابع توضیح داده شده در قسمت 3-6 است با این تفاوت که bitrate از 1 تا 6 در نظر گرفته شده است.

(4-7



همانطور که می‌بینیم اختلاف آستانه نویزی که bitrate 1 و 2 و 3 تحمل می‌کنند بسیار به هم نزدیک است. این نزدیک بودن به دلیل ساختار کلمه "signal" است.

```
BitRate = 1 | Noise = 1.04 | Variance = 1.0816
BitRate = 2 | Noise = 1 | Variance = 1
BitRate = 3 | Noise = 0.96 | Variance = 0.9216
```

نحوه محاسبه این آستانه نیز همانند تابع توضیح داده شده در قسمت 7-3 است.

(4-8

همانطور که توضیح داده شد هرچه تفاوت دو فرکانس متوالی کمتر باشد، کمتر نسبت به نویز مقاوم است پس با افزایش پهنای باند می‌توانیم فاصله‌ی دو فرکانس متوالی را بیشتر کنیم و درنتیجه کدگذاری نسبت به مویز مقاوم‌تر می‌شود؛ همچنین با بیشتر شدن پهنای باند، سرعت انتقال اطلاعات نیز افزایش می‌یابد.