به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی برق

تمرین دوم شبیه سازی

درس سیستم‌های مخابراتی

علی نوریان
۹۸۱۰۲۵۲۷

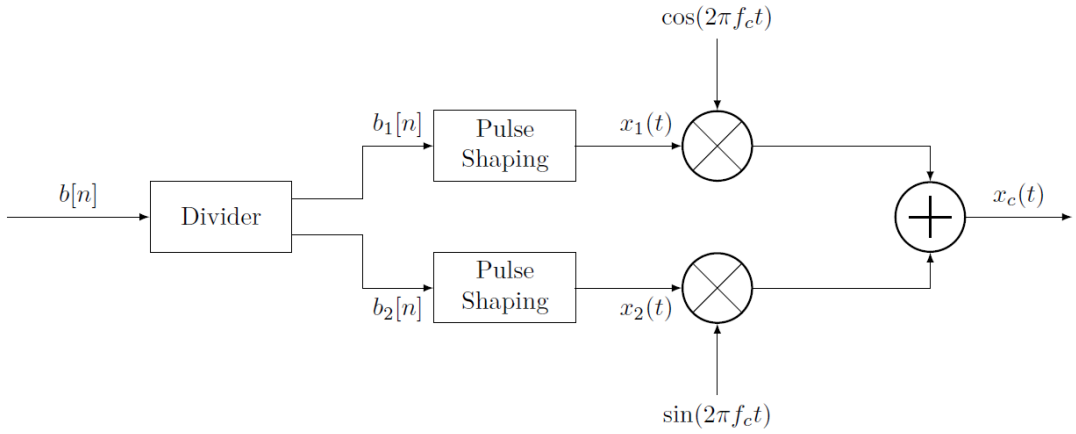
استاد مربوطه :
دکتر پاکروان

دی ۱۴۰۰

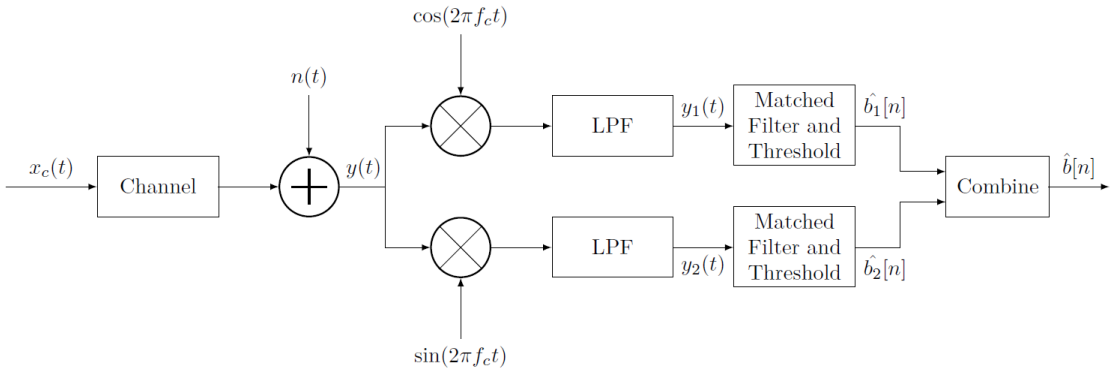
فهرست مطالب

۱ مقدمه

در این پروژه قصد داریم یک سیستم مخابرات دیجیتال را به طور کامل شبیه سازی کنیم و تأثیر پارامترهای مختلف را بر عملکرد این سیستم مشاهده کنیم. دیاگرام بلوکی فرستنده و گیرنده در شکل های ۱ و ۲ نمایش داده شده اند.



شکل ۱: دیاگرام بلوکی فرستنده



شکل ۲: دیاگرام بلوکی گیرنده

۲ پیاده سازی بلوک ها به صورت مجزا

۱.۲

تابع Divide را به صورت زیر تعریف میکنیم:

```
function [b1,b2] = Divide(b)
   b1 = b(1:2:end);
   b2 = b(2:2:end);
end
```

همانطور که در دستورکار خواسته است سیگنال ورودی را به دو سیگنال که یکی شامل درایه‌های زوج سیگنال اصلی و دیگری شامل درایه‌های فرد سیگنال اصلی می‌باشد، تقسیم می‌کنیم.

وارون این تابع را که دو سیگنال تقسیم شده را می‌گیرد و سیگنال اصلی را می‌سازد به صورت زیر تعریف می‌کنیم:

```
function b_hat = Combine(b1_hat, b2_hat)
   b_hat = zeros(1,length(b1_hat)*2);
   b_hat(1:2:end-1) = b1_hat;
   b_hat(2:2:end) = b2_hat;
end
```

۲.۲

تابع PulseShaping، سیگنال صفر یک ورودی و شکل سیگنال متناظر با صفر و یک را دریافت می‌کند. به جای هر درایه از سیگنال اصلی که یک باشد، شکل سیگنال متناظر با یک و به جای هر درایه از سیگنال اصلی که صفر باشد، شکل سیگنال متناظر با صفر را قرار می‌دهد:

```
function y = PulseShaping(x,p0,p1)
   l = length(p0);
   y = zeros(1,length(x)*l);
   for i=1:length(x)
      if x(i) == 1
         y((i-1)*l+1:i*l) = p1;
      else
         y((i-1)*l+1:i*l) = p0;
      end
   end
end
```

۳.۲

تابع AnaloogMod خروجی $x_c(t)$ به صورت زیر تولید می‌کند:

```
function xc = AnalogMod(x1, x2, Fs, fc)
   Ts = 1/Fs;
   L = length(x1)/Fs;
   t = 0:Ts:L-Ts;
   xc = x1.*cos(2*pi*fc*t) + x2.*sin(2*pi*fc*t);
end
```

۴.۲

برای پیاده سازی رفتار کانال، سیگنال مادوله شده قسمت قبل را از یک فیلتر میانگذر عبور می‌دهیم. بهنای باند این فیلتر را برابر BW درنظر می‌گیریم که BW همان پهنای باند سیگنال پیام است:

```
function x_c = Channel(xc, Fs, Fm, BW)
  x_c = bandpass(xc,[Fm-BW/2, Fm+BW/2],Fs);
end
```

۵.۲

سیگنال دریافت شده از کانال را demodulate کنیم. این کار را با کمک تابع زیر انجام می‌دهیم:

```
function [y1,y2] = AnalogDemod(y,Fs,fc,BW)
  Ts = 1/Fs;
  L = length(y)/Fs;
  t = 0:Ts:L-Ts;
  y_1 = y.*cos(2*pi*fc*t);
  y_2 = y.*sin(2*pi*fc*t);

  Y_1 = fftshift(fft(y_1));
  L = length(Y_1);
  f = Fs*(floor(-L/2):floor(L/2)-1)/L;
  Y1 = Y_1 .* ((f < BW) & (f > -BW));
  y1 = 2*ifft(ifftshift(Y1));

  Y_2 = fftshift(fft(y_2));
  L = length(Y_2);
  f = Fs*(floor(-L/2):floor(L/2)-1)/L;
  Y2 = Y_2 .* ((f < BW) & (f > -BW));
  y2 = 2*ifft(ifftshift(Y2));
end
```

۶.۲

تابع نهایی MatchedFilt به صورت زیر (همانطور که اسلایدهای درس آمده است) بررسی می‌کند که سیگنال بدست آمده به ازای هر بیت با سیگنال اصلی کدام بیت مشابه‌تر است. مقدار کورلیشن سیگنال ورودی با سیگنال اصلی بیت صفر و یک در لحظه $t = T$ به همراه بیت تشخیص داده شده به عنوان خروجی این تابع تعریف می‌شوند:

```
function [b,matchedFilt0,matchedFilt1] = MatchedFilt(y,p0,p1)
  l = length(p0); n = length(y)/l;
  y0 = reshape(y,l,n)';
  yc0 = convn(y0,flip(p0))/length(p0);
  yc1 = convn(y0,flip(p1))/length(p1);
  matchedFilt0 = yc0(:,length(p0));
  matchedFilt1 = yc1(:,length(p1));
  b = matchedFilt1 > matchedFilt0;
end
```

می‌دانیم: $corr(x(t), y(t)) = x(t) * y(-t)$، و از آنجایی که دستور convn همزمان n کانوولوشن را حساب می‌کند، از این دستور به جای دستور کورلیشن استفاده کرده‌ایم.

۳ انتقال دنباله ی تصادفی صفر و یک

متغیرهای مشخص شده در دستورکار را به صورت زیر تعریف می‌کنیم:

```
Fs = 10^6;
fc = 10*10^3;
Fm = 10*10^3;
BW = 10^3;
T = 0.01;
t = 0:1/Fs:T-1/Fs;
```

PAM ۱.۳

دو پالس مربوط به بیت صفر و یک را به صورت زیر تعریف می‌کنیم:

```
p1 = ones(1,length(t));
p0 = -p1;
```

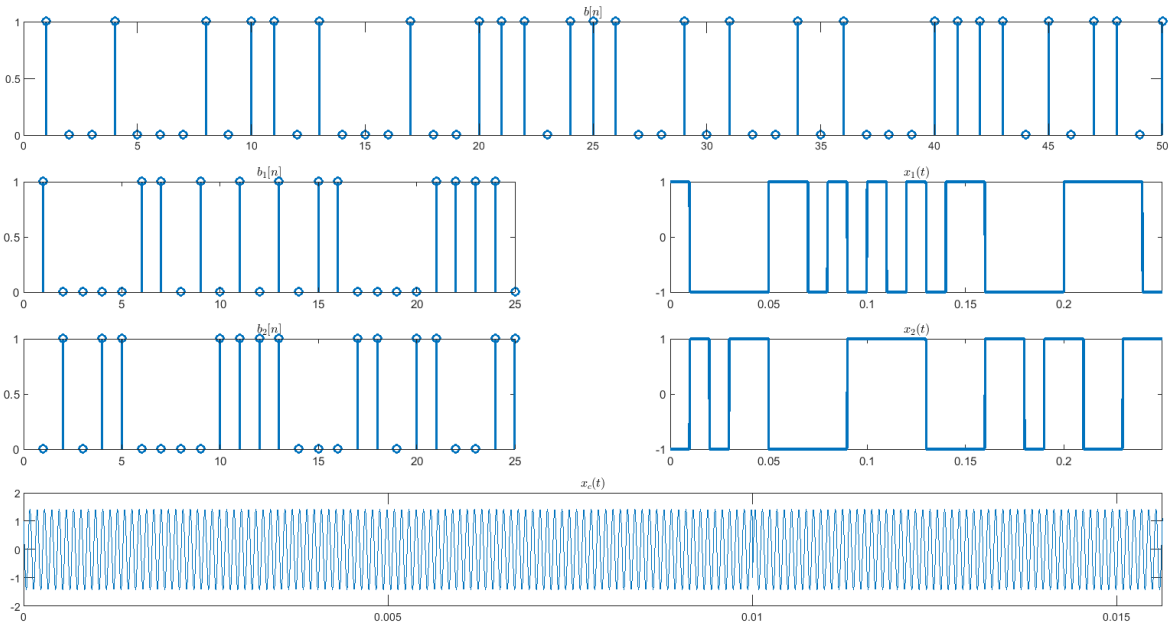
۱.۱.۳

طول سیگنال پیام را برابر ۱۰۰ میگیریم. بازه‌های نتایج شبیه سازی را به شکلی تنظیم می‌کنیم که تصاویر «قشنگ» باشند! لازم به ذکر است تمامی مراحل در تابع زیر صورت می‌گیرد:

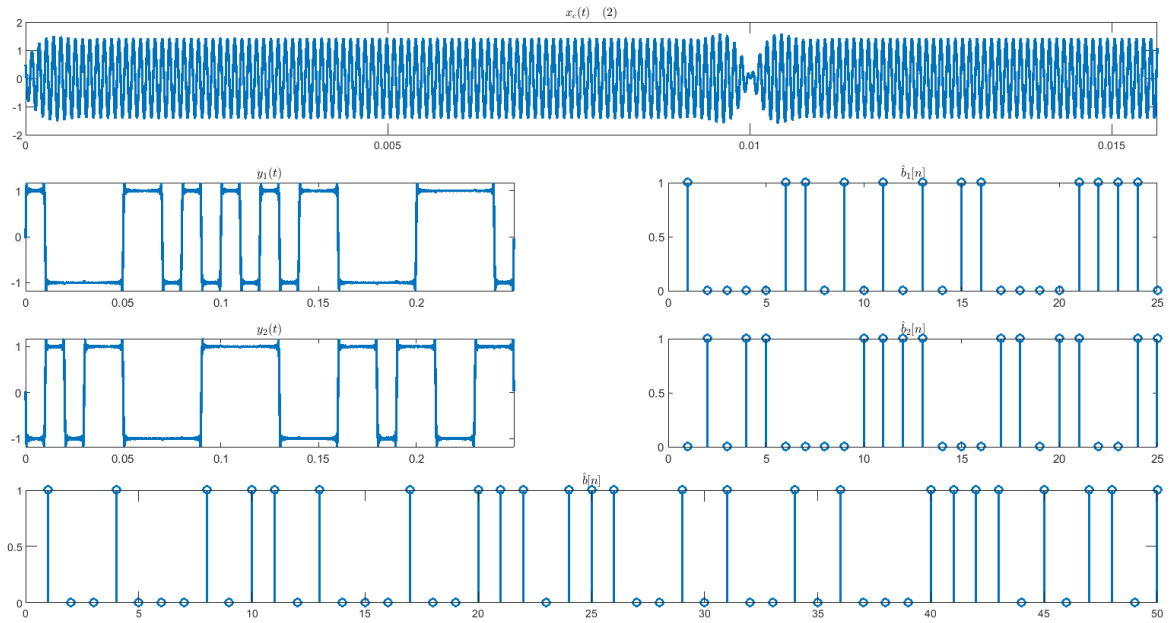
```
function [b_hat,matchedFilt0,matchedFilt1] = ...
             transmit(b,p1,p0,Fs,fc,Fm,BW,snr,draw)
   [b1,b2] = Divide(b);
   x1 = PulseShaping(b1,p0,p1);
   x2 = PulseShaping(b2,p0,p1);
   xc = AnalogMod(x1,x2,Fs,fc);

   x_c = Channel(xc,Fs,Fm,BW);
   x_c = awgn(x_c,snr);
   [y1,y2] = AnalogDemod(x_c,Fs,fc,BW);

   [b1_hat,b1_matchedFilt0,b1_matchedFilt1] = MatchedFilt(y1,p0,p1);
   [b2_hat,b2_matchedFilt0,b2_matchedFilt1] = MatchedFilt(y2,p0,p1);
   b_hat = Combine(b1_hat,b2_hat);
   matchedFilt0 = Combine(b1_matchedFilt0,b2_matchedFilt0);
   matchedFilt1 = Combine(b1_matchedFilt1,b2_matchedFilt1);

   if draw == 10
      showSignals(b,b1,b2,x1,x2,xc,...
             x_c,y1,y2,b1_hat,b2_hat,b_hat,Fs);
   elseif draw == 01
      drawSignalConstellation(matchedFilt0);
   elseif draw == 11
      showSignals(b,b1,b2,x1,x2,xc,...
             x_c,y1,y2,b1_hat,b2_hat,b_hat,Fs);
      drawSignalConstellation(matchedFilt0);
   end
end
```

ورودی اول این تابع سیگنال پیام، ورودی دوم و سوم پالس‌های مربوط به صفر و یک، ورودی چهارم فرکانس نمونه برداری، ورودی پنجم فرکانس حامل، ورودی ششم فرکانس مرکزی کانال، ورودی هفتم پهنای باند پیام، ورودی هشتم مقدار SNR می‌باشد. همچنین ورودی آخر نیز مشخص میکند چه مواردی از این فرایند نمایش داده شوند.

با اجرای این قطعه کد به ازای SNR بی‌نهایت (بدون نویز) خروجی قسمت‌های مختلف بلوک‌های پیاده‌سازی شده به صورت زیر است:



شکل ۳: سیگنال‌های بلوک فرستنده

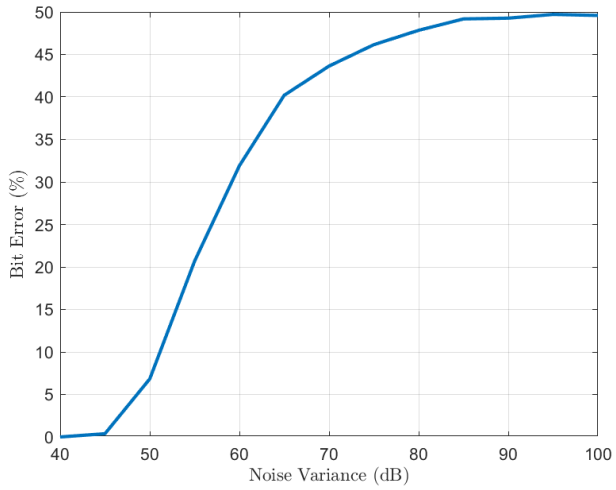


شکل ۴: سیگنال‌های بلوک گیرنده

همانطور که مشخص است سیگنال پیام بدون خطا در گیرنده بازسازی شده است.

۲.۱.۳

با افزایش واریانس نویز، انتظار داریم خطا افزایش پیدا کند. همچنین برای یک سیگنال باینری ماکسیمم خطا برای 0.5 می‌باشد. بنابراین انتظار داریم با افزایش واریانس نویز مقدار خطا از صفر به سمت ۵۰درصد افزایش پیدا کرده و به آن همگرا شود. خروجی زیر به ازای مقادیر واریانس ۴۰، ۴۵، ۵۰، ...، ۱۰۰ دسی‌بل رسم شده است:



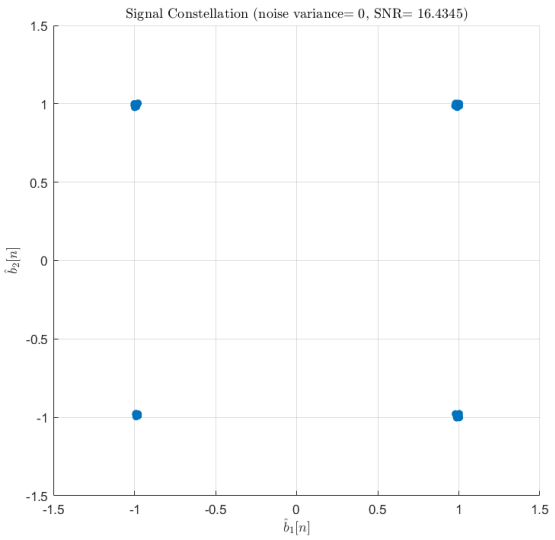
شکل ۵

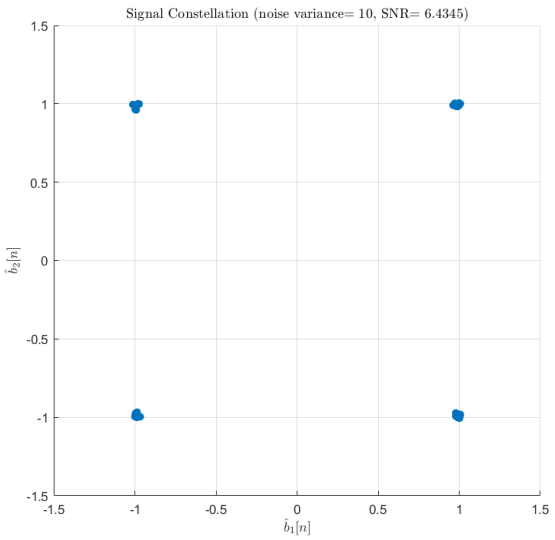
۳.۱.۳

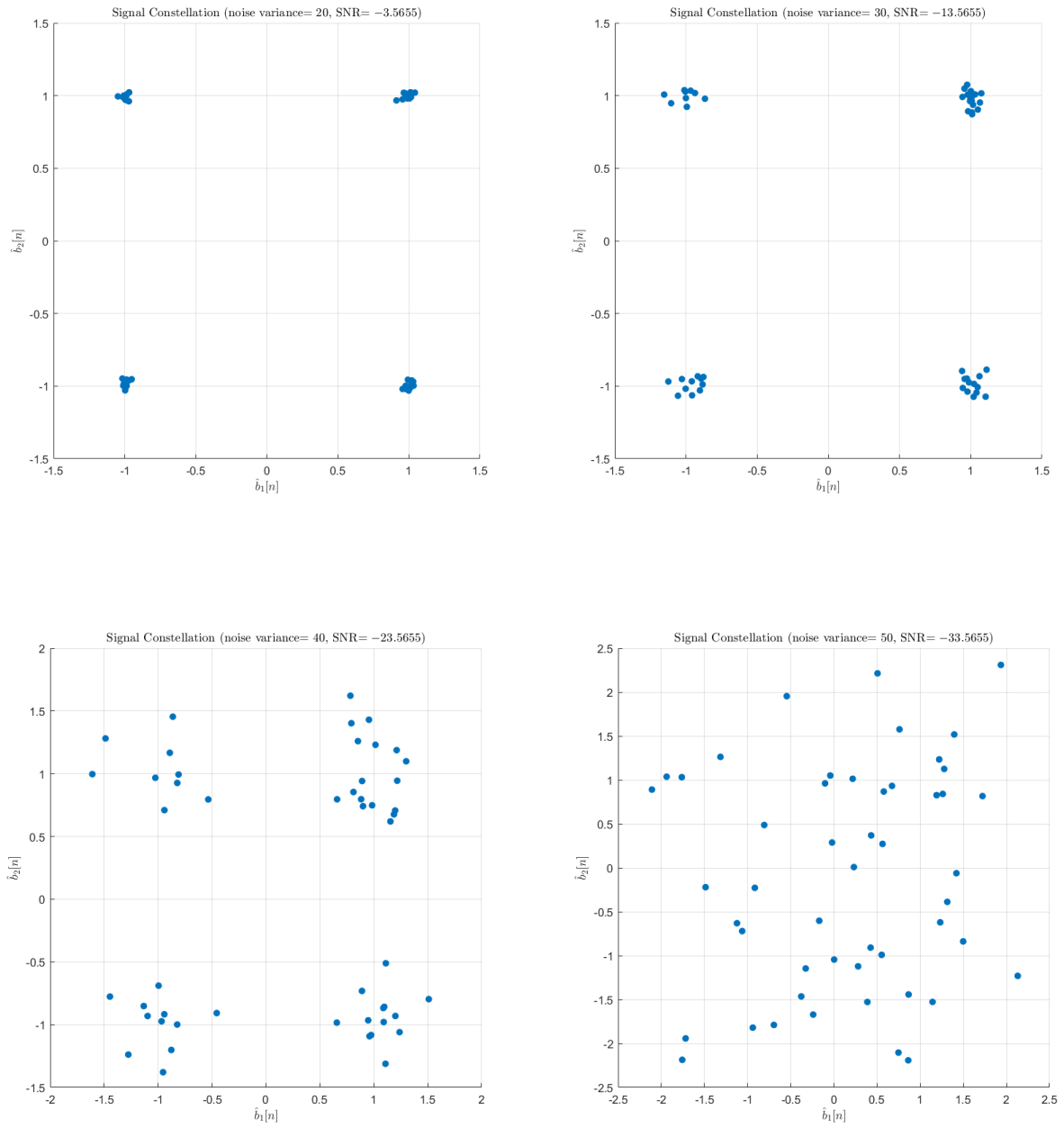
برای این منظور تابع drawSignalConstellation را به صورت زیر تعریف می‌کنیم:

```
function drawSignalConstellation(matchedFilt0)
   figure;
   [b1_matchedFilt0,b2_matchedFilt0] = Divide(matchedFilt0);
   scatter(b1_matchedFilt0,b2_matchedFilt0,'filled'); grid on;
end
```

خروجی تابع فوق به ازای واریانس‌های نویز برابر ۰، ۱۰، ۲۰، ۳۰، ۴۰ و ۵۰ دسی‌بل در ادامه آمده است:







شکل ۶

همانطور که مشاهده می‌شود با افزایش واریانس نویز تجمع نقاط حول ۴ مقدار $(\pm 1, \pm 1)$ کاهش می‌یابد و در نتیجه باعث افزایش خطا در بازیابی می‌شود که مطابق انتظار است.

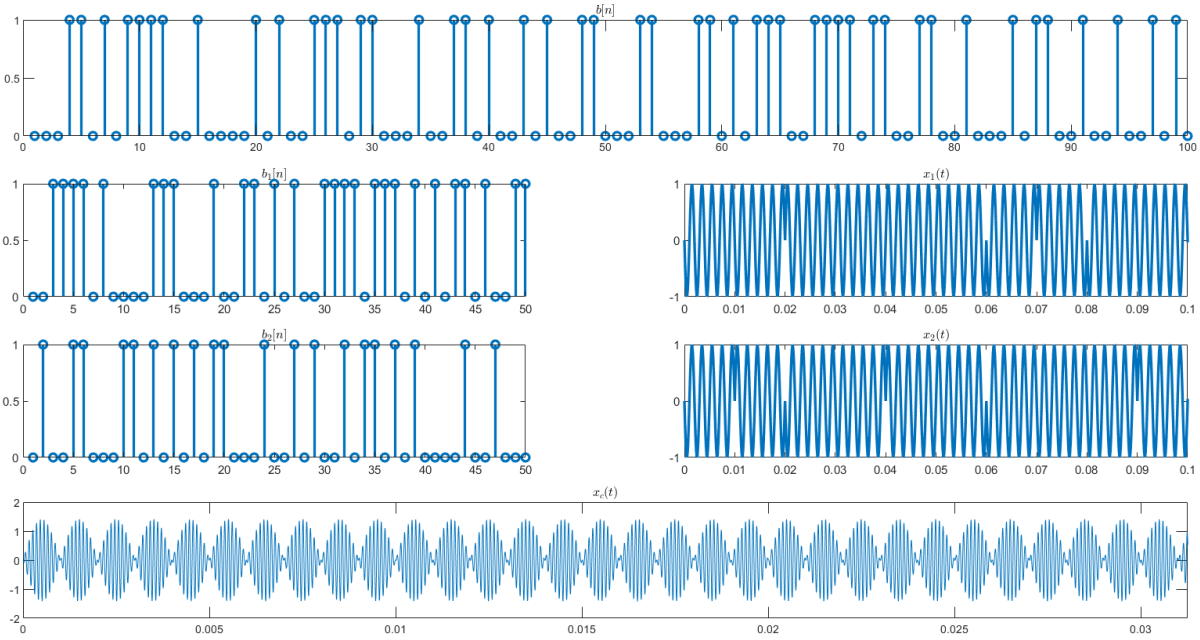
PSK ۲.۳

در این قسمت پالس‌های متناظر با صفر و یک را به صورت سینوسی تعریف می‌کنیم:

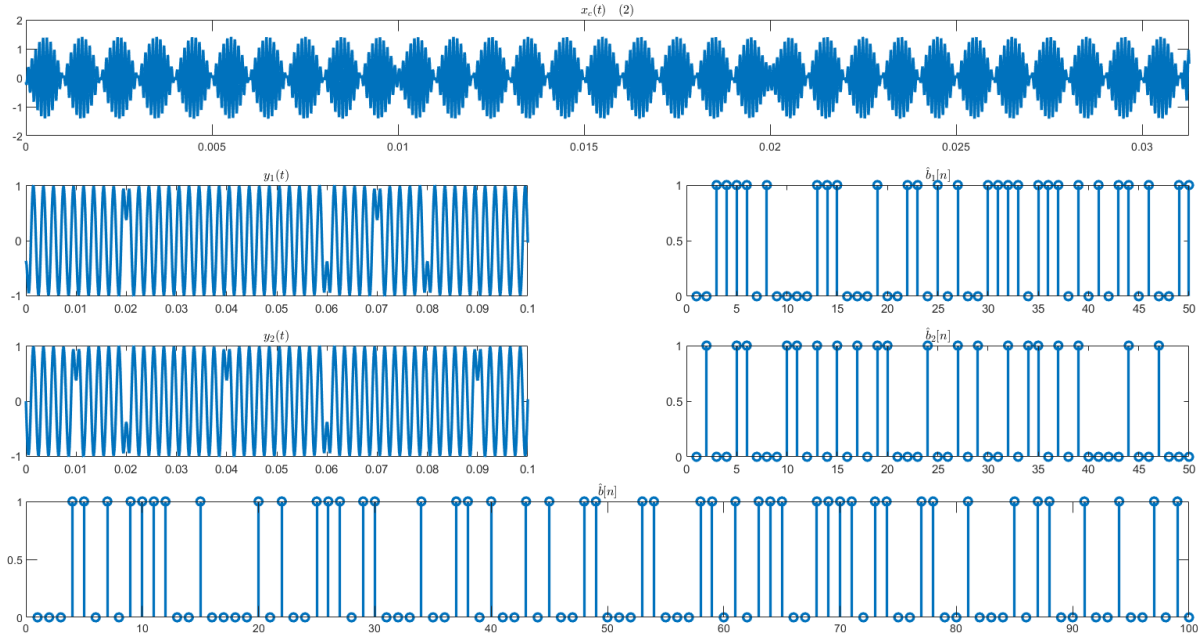
```
p1 = sin(2*pi*500*t);
p0 = -p1;
```

۱.۲.۳

مراحل قسمت قبل را اینبار برای شکل پالس‌های سینوسی PSK تکرار می‌کنیم. خروجی به صورت زیر است:



شکل ۷: سیگنال‌های بلوک فرستنده

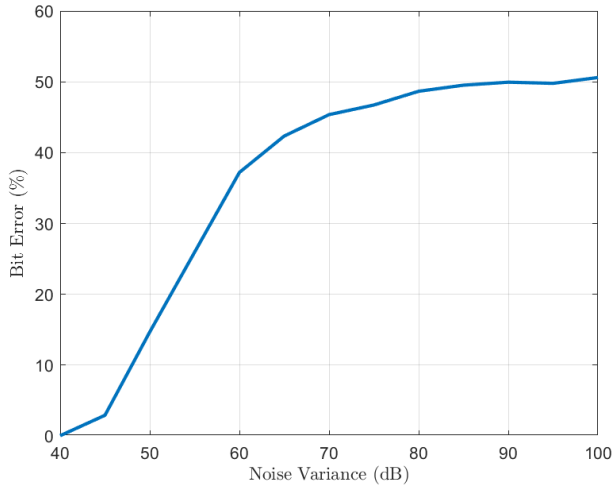


شکل ۸: سیگنال‌های بلوک گیرنده

همانطور که مشخص است سیگنال پیام بدون خطا در گیرنده بازسازی شده است.

۲.۲.۳

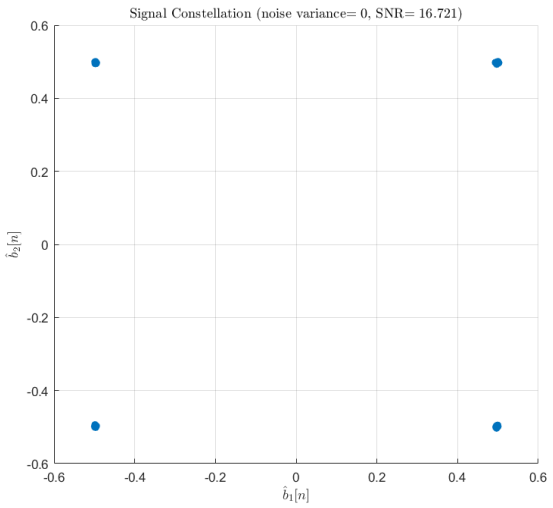
همانند قسمت قبل انتظار داریم با افزایش واریانس نویز مقدار خطا از صفر به سمت ۵۰درصد افزایش پیدا کرده و به آن همگرا شود. خروجی زیر به ازای مقادیر واریانس ۴۰، ۴۵، ۵۰، ...، ۱۰۰ دسی‌بل رسم شده است:

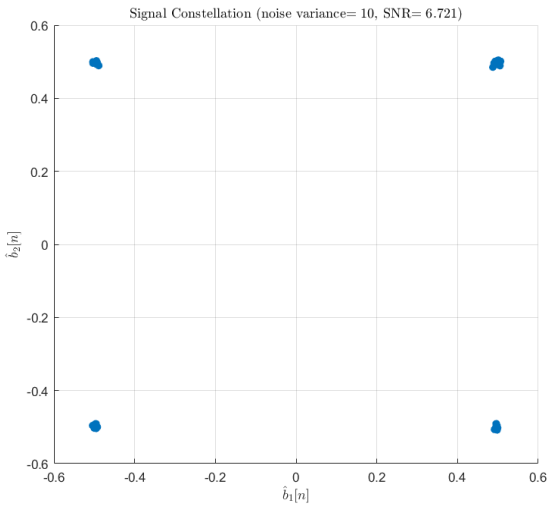


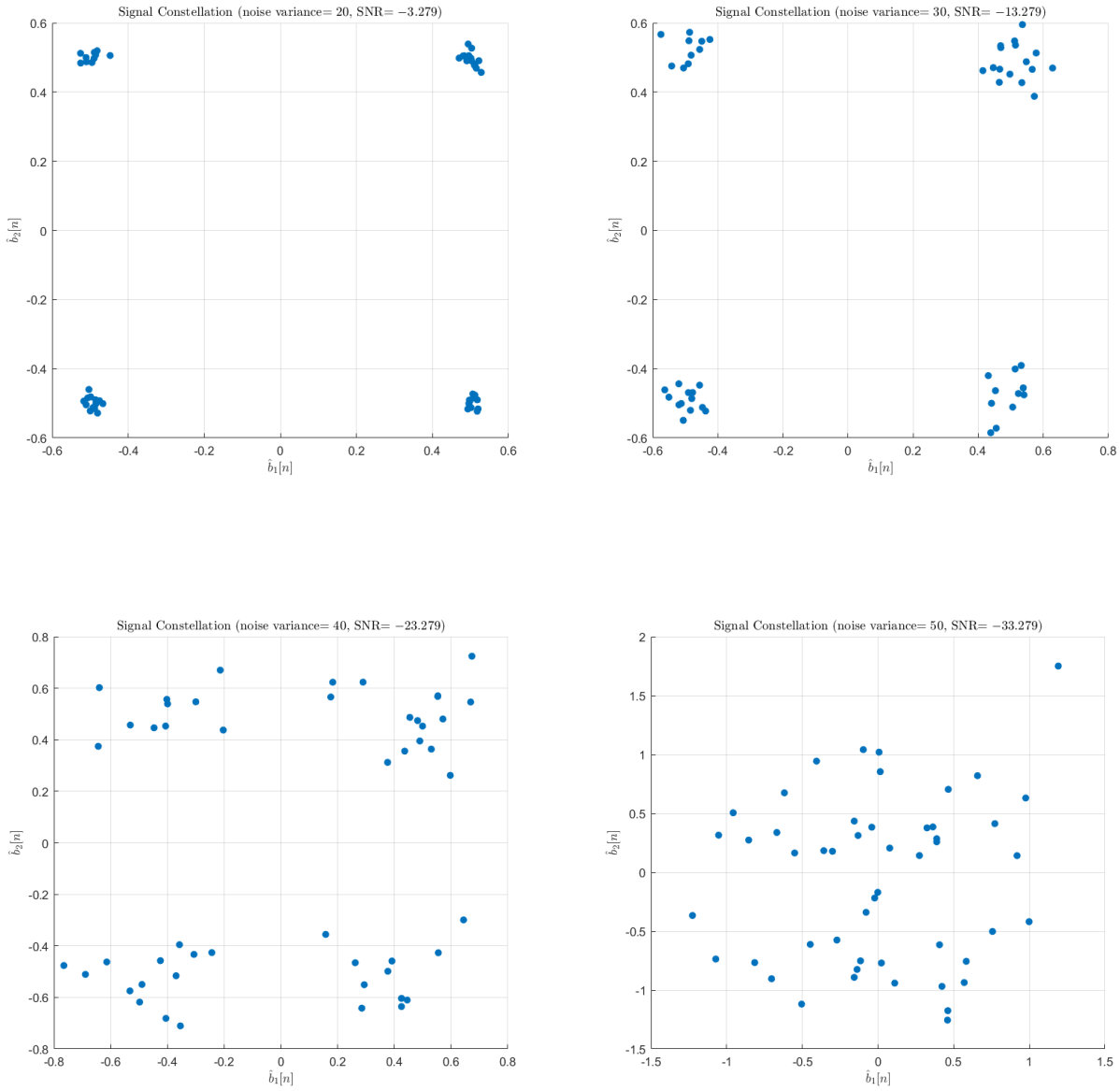
شکل ۹

۳.۲.۳

خروجی مربوطه به ازای واریانس‌های نویز برابر ۰، ۱۰، ۲۰، ۳۰، ۴۰ و ۵۰ دسی‌بل در ادامه آمده است:







شکل ۱۰

مانطور که مشاهده می‌شود با افزایش واریانس نویز تجمع نقاط حول ۴ مقدار کاهش می‌یابد و در نتیجه باعث افزایش خطا در بازیابی می‌شود که مطابق انتظار است.

FSK ۳.۳

در این قسمت پالس‌های متناظر با صفر و یک را به صورت سینوسی با فرکانس‌های متفاوت تعریف می‌کنیم:

```
p1 = sin(2*pi*1000*t);
p0 = sin(2*pi*1500*t);
```

همچنین با توجه به دو سیگنال فوق پهنای باند کانال را برابر $3KHz$ قرار می‌دهیم.

۱.۳.۳

متعامد بودن دو سیگنال را بررسی می‌کنیم:

$$< p_1, p_0 >= \int_0^L \sin(2\pi 1000t)\sin(2\pi 1500t)$$

$$\sin(2\pi 1000t)\sin(2\pi 1500t) = \frac{1}{2}(\cos(2\pi 500t) - \cos(2\pi 2500t))$$

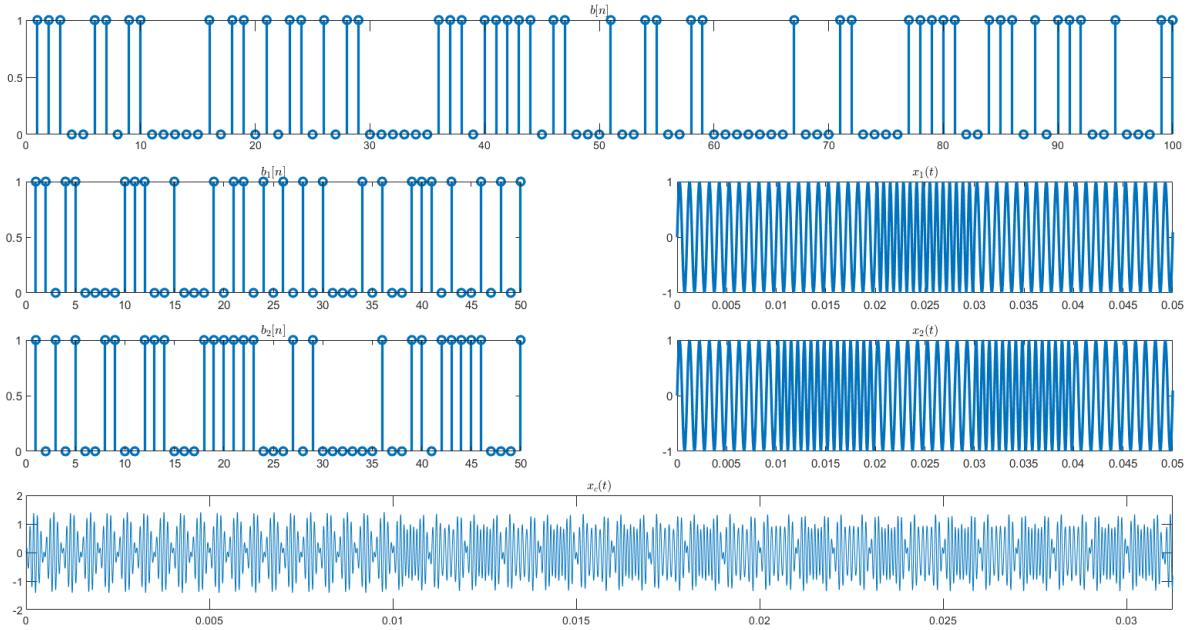
$$\Longrightarrow < p_1, p_0 >= \int_0^L \frac{1}{2}(\cos(2\pi 500t) - \cos(2\pi 2500t))$$

$$= \frac{1}{2}\int_0^L \cos(2\pi 500t) - \frac{1}{2}\int_0^L \cos(2\pi 2500t) = \frac{1}{2}(0) - \frac{1}{2}(0) = 0$$

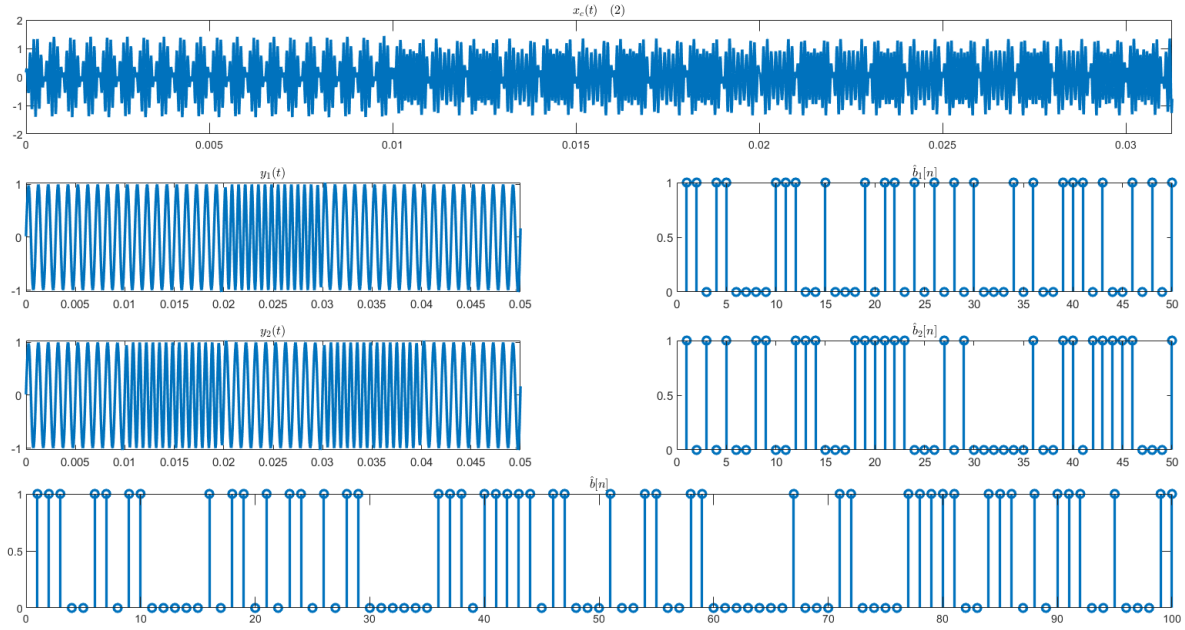
در نتیجه دو سیگنال متعامد هستند.

۲.۳.۳

مراحل قسمت قبل را اینبار برای شکل پالس‌های سینوسی FSK تکرار می‌کنیم. خروجی به صورت زیر است:



شکل ۱۱: سیگنال‌های بلوک فرستنده

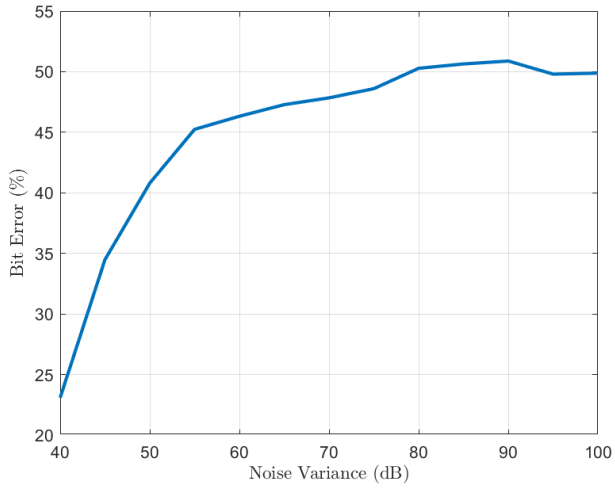


شکل ۱۲: سیگنال‌های بلوک گیرنده

همانطور که مشخص است سیگنال پیام بدون خطا در گیرنده بازسازی شده است.

۳.۳.۳

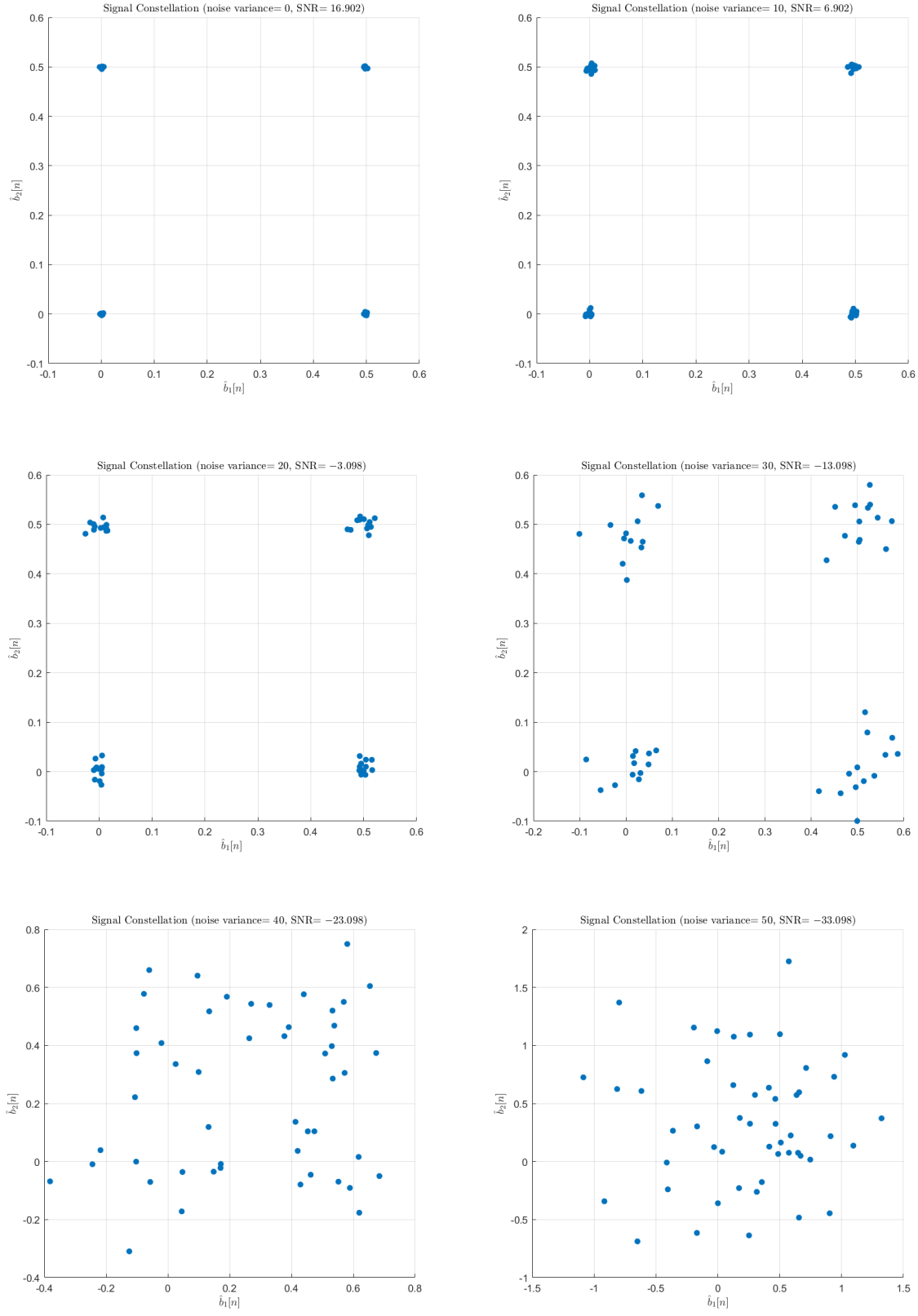
همانند قسمت‌های قبل انتظار داریم با افزایش واریانس نویز مقدار خطا از صفر به سمت ۵۰درصد افزایش پیدا کرده و به آن همگرا شود. خروجی زیر به ازای مقادیر واریانس ۴۰، ۴۵، ۵۰، ...، ۱۰۰ دسی‌بل رسم شده است:



شکل ۱۳

۴.۳.۳

خروجی مربوطه به ازای واریانس‌های نویز برابر ۰، ۱۰ ،۲۰، ۳۰، ۴۰ و ۵۰ دسی‌بل در ادامه آمده است:



شکل ۱۴

۴.۳

همانطور که از نمودارهای Signal Constellation مشخص است در حالت PAM مقاومت بیشتری نسبت به نویز وجود دارد و با این روش معرفی شده برای بازیابی پیام، FSK بیشترین آسیب را از نویز میبیند. همچنین در FSK به پهنای باند بیشتری برای کانال نیاز داریم. در هر سه حالت نیز با افزایش نویز، احتمال خطا به ۵۰درصد میل پیدا می‌کند.

۴ انتقال دنباله ای از اعداد ۸ بیتی

۱.۴

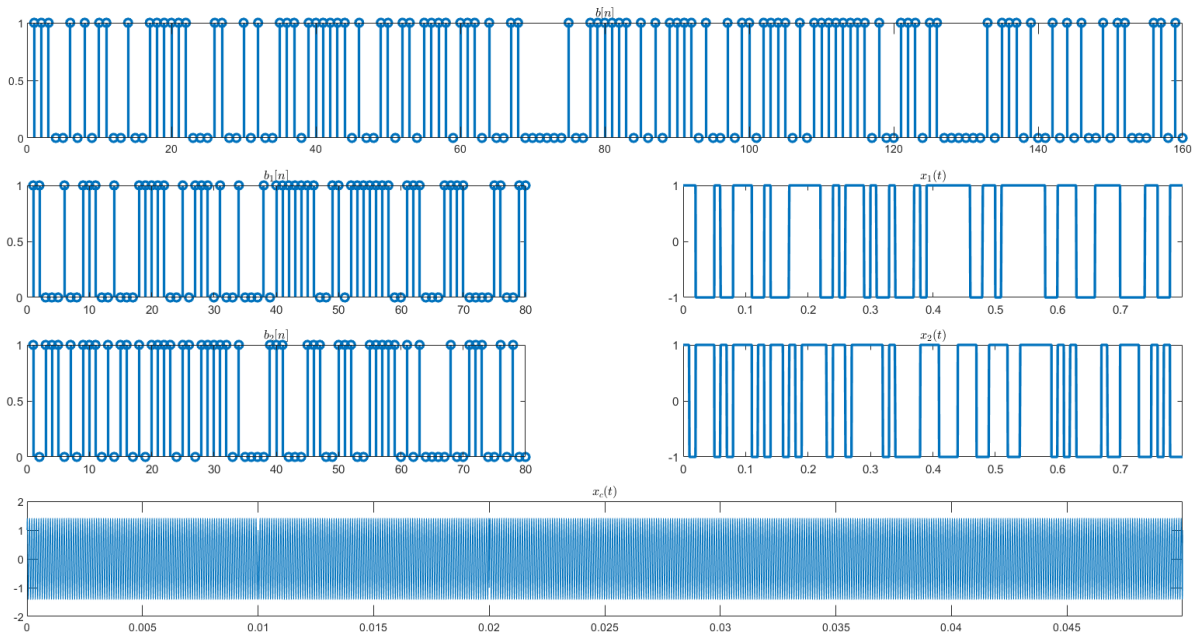
دو تابع مورد نظر را به صورت زیر تعریف می‌کنیم:

```
function y = SourceGenerator(x)
   y = de2bi(x,8,'left-msb')';
   y = y(:)';
end
```

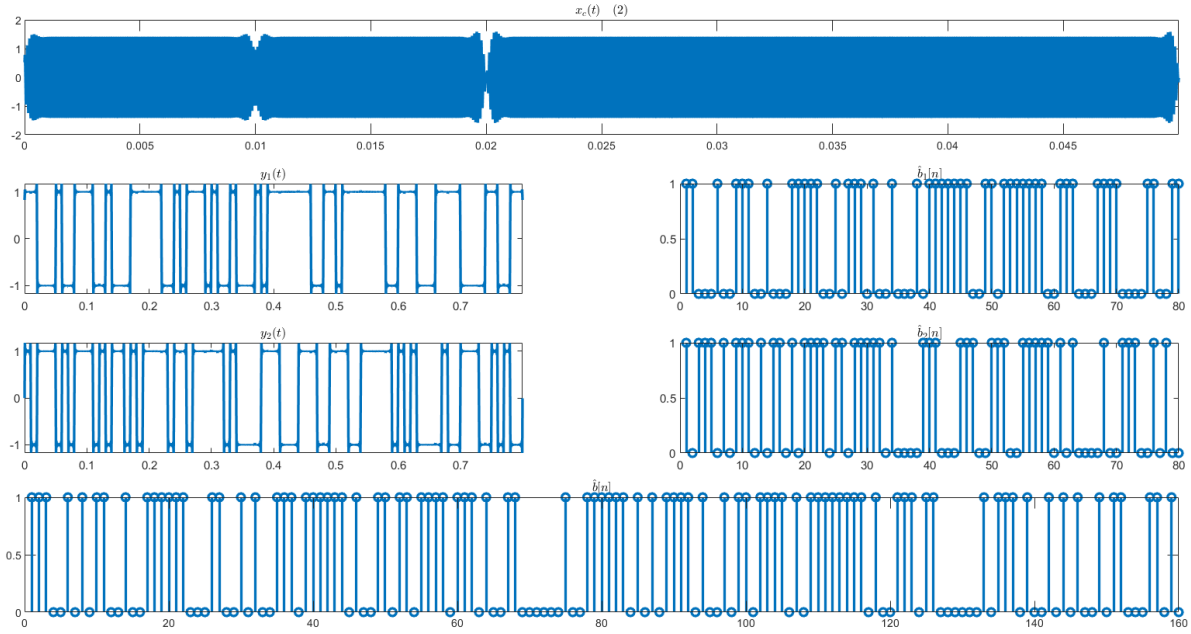
```
function y = OutputDecoder(x)
   n = length(x)/8;
   y = reshape(x',n,8);
   y = bi2de(y,'left-msb')';
end
```

۲.۴

بعد از تولید دنباله‌ای از صفر تا ۲۵۵ آن را به کمک توابعی که نوشتیم به ۰ و ۱ تبدیل کرده و پس از ارسال آن، در گیره آن را بازیابی می‌کنیم. خروجی این فرایند به صورت زیر است:



شکل ۱۵: سیگنال‌های بلوک فرستنده



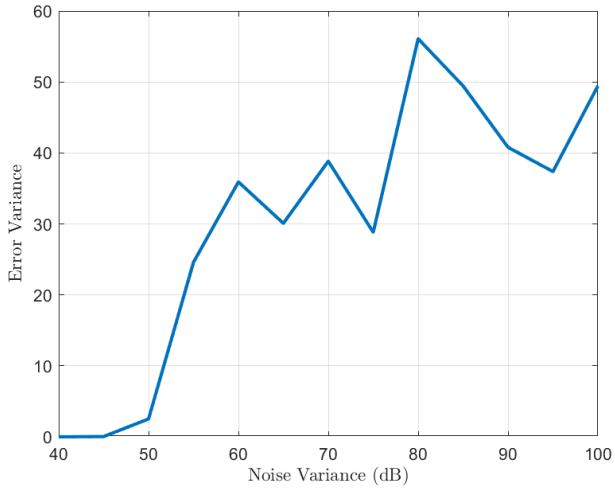
شکل ۱۶: سیگنال‌های بلوک گیرنده

```
>> x

x =

  Columns 1 through 10

   229   100   252   101    59   244   219   221    48    39

  Columns 11 through 20

   234   244   183   175   244   236    11   165    75    26

>> x_hat

x_hat =

  Columns 1 through 10

   229   100   252   101    59   244   219   221    48    39

  Columns 11 through 20

   234   244   183   175   244   236    11   165    75    26
```

شکل ۱۷

همانظور که از شکل فوق پیداست، سیگنال پیام به درستی بازیابی شده است.

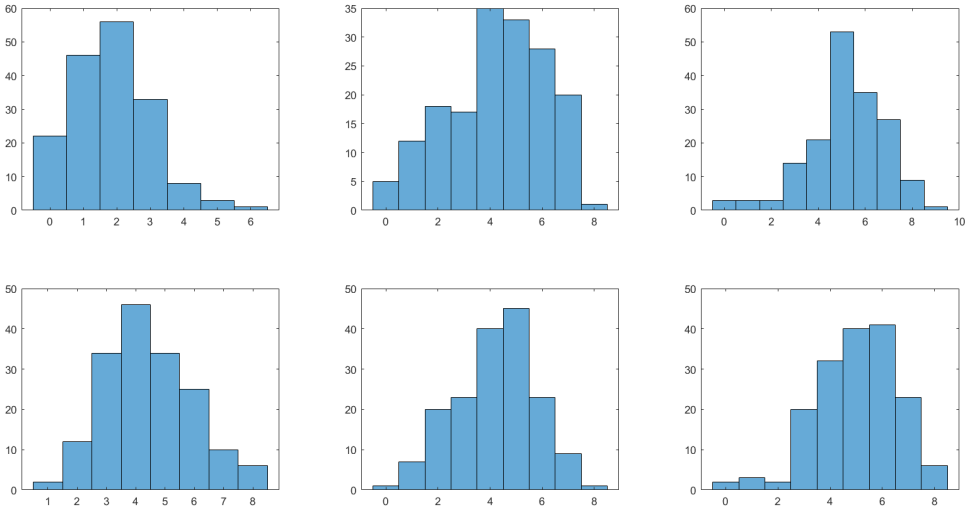
اکنون با توجه به میانگینی که برای خطا در بخش ۲.۱.۳ بدست آورده‌ایم، واریانس خطا را برای یک دنباله بر حسب واریانس نویز رسم می‌کنیم:



شکل ۱۸

۳.۴

توزیع خطا را رسم می‌کنیم:



شکل ۱۹

۴.۴

$$\mathbb{E}[(X-\mu_N)^2|N\to\infty] = 2\times(2^8-1)^2 = 130050$$

همانطور که انتظار داریم واریناس خطا به مقدار بالایی میرسد.

۵ کدینگ منبع

۱.۵

۱.۱.۵

کد دو حرف b و d یکسان است. در نتیجه اگر عبارت 10 دریافت شود قادر به تشخیص آنکه حرف ارسالی b بوده است یا d نخواهیم بود.

۲.۱.۵

اگر در گیرنده دنباله 010 ظاهر شود نمیتوان متوجه شد که دو حرف a و b که پشت سر هم هستند دریافت شده یا به تنهایی حرف d دریافت شده است. به همین دلیل این دنباله نیز مشکل دارد.

۳.۱.۵

با دریافت 0 نمی‌توانیم در لحظه تشخیص دهیم که آیا حرف a آمده است یا خیر. در واقع باید منتظر ماند و بیت بعدی را نگاه کرد. اگر بیت بعدی صفر بود آنگاه می‌توان گفت بیت قبلی حرف a بوده است. در غیر این صورت عبارت 01 را خواهیم داشت که همچنان نمی‌توانیم در این لحظه بگوییم حرف b وارد شده است و الی آخر... .

۲.۵

مسئله بهینه سازی زیر را با استفاده از روش ضرایب لاگرانژ حل می‌کنیم:

$$\min_{l_1,l_2,..,l_M} \sum_{i=1}^{M} p_i l_i \qquad s.t. \sum_{i=1}^{M} 2^{-l_i} \leq 1$$

$$f = \sum_{i=1}^{M} p_i l_i, \qquad g = \sum_{i=1}^{M} 2^{-l_i} \leq 1$$

$$\implies \nabla f(l_1, l_2, ..., l_M) + \lambda \nabla g(l_1, l_2, ..., l_M) = 0$$

$$\implies p_i - \lambda \ln(2) e^{-l_i \ln(2)} = 0 \implies e^{-l_i \ln(2)} = \frac{p_i}{\lambda \ln(2)}$$

$$\implies l_i = -\frac{\ln(\frac{p_i}{\lambda \ln(2)})}{\ln(2)} \tag{۱}$$

در نتیجه طبق رابطه فوق و اینکه نقطه بهینه در حالت تساوی نامعادله است، خواهیم داشت:

$$\sum_{i=1}^{M} 2^{-l_i} = 1 \implies \sum_{i=1}^{M} 2^{\frac{\ln(\frac{p_i}{\lambda \ln(2)})}{\ln(2)}} = 1$$

$$\implies \sum_{i=1}^{M} e^{\ln(\frac{p_i}{\lambda \ln(2)})} = 1 \implies \sum_{i=1}^{M} \frac{p_i}{\lambda \ln(2)} = 1 \implies \frac{1}{\lambda \ln(2)} = 1$$

$$\implies \lambda = \frac{1}{\ln(2)} \tag{۲}$$

در نتیجه با توجه به دو عبارت ۱ و ۲ خواهیم داشت:

$$\Longrightarrow l_i = -\log_2 p_i \tag{۳}$$

۳.۵

با توجه به نتیجه قسمت قبل طول بهینه برای هر حرف و در نتیجه کد آن را بدست می‌آوریم:

$$X = \begin{pmatrix} a & b & c & d & e & f \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{8} & \frac{1}{16} & \frac{1}{32} & \frac{1}{32} \end{pmatrix}$$

$$l_i = -\log_2 p_i$$

$$l_a = -\log(\frac{1}{2}) = 1, \quad l_b = -\log(\frac{1}{4}) = 2, \quad l_c = -\log(\frac{1}{8}) = 3$$
$$l_d = -\log(\frac{1}{16}) = 4, \quad l_e = -\log(\frac{1}{32}) = 5, \quad l_f = -\log(\frac{1}{32}) = 5$$

a	b	c	d	e	f
1	2	3	4	5	5
0	10	110	1110	11110	11111

۴.۵

طول متوسط کلمه‌کدها برابر است با:

$$\mathbb{E}[l(X)] = \frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{16} \times 4 + \frac{1}{32} \times 5 + \frac{1}{32} \times 5$$

$$\Longrightarrow \mathbb{E}(l(X)) = \frac{31}{16} = 1.9375 \tag{۴}$$

۵.۵

تابع InformationSource را به صورت زیر می‌نویسیم:

```
function y = InformationSource(n)
  r = rand(1,n);
  a = r <= 0.5;
  b = (r > 0.5) & (r <= 0.75);
  c = (r > 0.75) & (r <= 0.875);
  d = (r > 0.8750) & (r <= 0.9375);
  e = (r > 0.9375) & (r <= 0.9688);
  f = (r > 0.9688) & (r <= 1);
  y = 'a'*a + 'b'*b + 'c'*c + 'd'*d + 'e'*e + 'f'*f;
  y = char(y);
end
```

۶.۵

تابع SourceEncode را با توجه به کدگذاری که انجام داده‌ایم به صورت زیر می‌نویسیم:

```
function y = SourceEncoder(x)
   n = length(x);
   l = sum(x=='a')+sum(x=='b')*2+sum(x=='c')*3+sum(x=='d')*4+...
   sum(x=='e')*5+sum(x=='f')*5;
   y = zeros(1,l);
   k = 1;
   for i=1:n
      if x(i) == 'a'
         y(k) = 0; k = k + 1;
      elseif x(i) == 'b'
         y(k:k+1) = [1 0]; k = k + 2;
      elseif x(i) == 'c'
         y(k:k+2) = [1 1 0]; k = k + 3;
      elseif x(i) == 'd'
         y(k:k+3) = [1 1 1 0]; k = k + 4;
      elseif x(i) == 'e'
         y(k:k+4) = [1 1 1 1 0]; k = k + 5;
      elseif x(i) == 'f'
         y(k:k+4) = [1 1 1 1 1]; k = k + 5;
      end
   end
end
```

۷.۵

تابع SourceDecode را به صورت زیر می‌نویسیم:

```
function y = SourceDecoder(x)
   n = length(x);
   i=1;
   y = [];
   while i <= n
      if x(i) == 0
         y = cat(2,y,'a'); i=i+1;
      elseif x(i+1) == 0
         y = cat(2,y,'b'); i=i+2;
      elseif x(i+2) == 0
         y = cat(2,y,'c'); i=i+3;
      elseif x(i+3) == 0
         y = cat(2,y,'d'); i=i+4;
      elseif x(i+4) == 0
         y = cat(2,y,'e'); i=i+5;
      else
         y = cat(2,y,'f'); i=i+5;
      end
   end
end
```

۸.۵

برای نمونه n را برابر ۵۰ می‌گیریم. نتیجه به صورت زیر است:

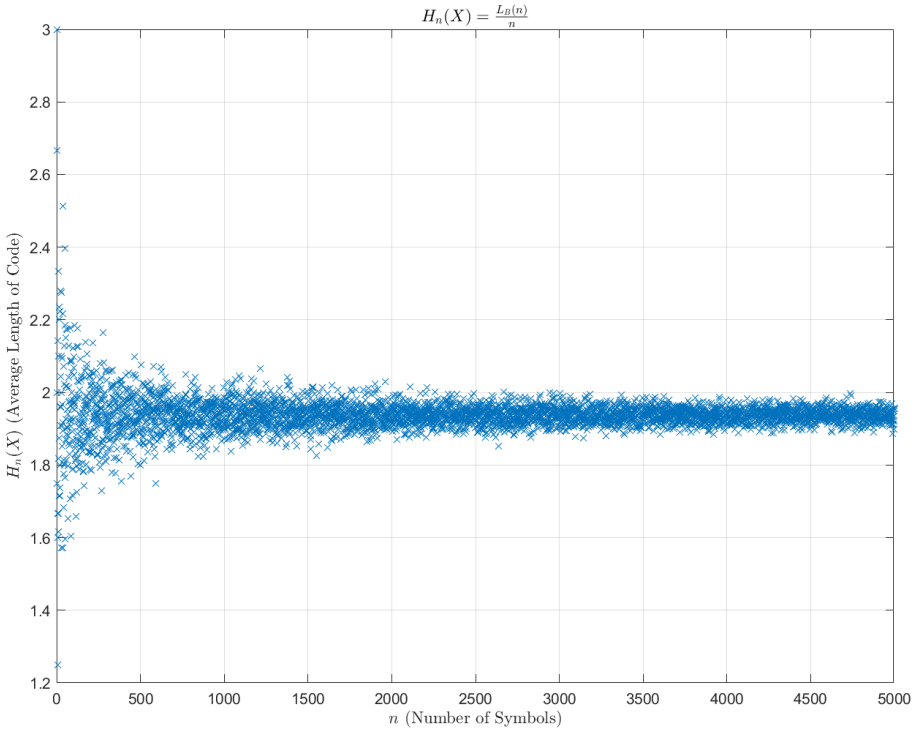
```
x = InformationSource(50);
y = SourceEncoder(x);
xd = SourceDecoder(y);

disp(['x  =   ', x]);
disp(['xd =   ', xd]);
```

```
Command Window
x     =     dadbddaabbabaafbacdcaffaabbbaabcbaaaabdaaaabaabeba
xd    =     dadbddaabbabaafbacdcaffaabbbaabcbaaaabdaaaabaabeba
```

شکل ۲۰

خروجی دیکد شده کاملا مشابه سیگنال اصلی است.

اکنون با تغییر مقدار n به مقادیر بزرگ مقدار متوسط طول کلمه‌کدها ($H_n(X) = \frac{L_B(X)}{n}$) را بدست آورده و بر حسب n رسم می‌کنیم:



شکل ۲۱

همانطور که در شکل فوق پیداست میانگین طول کلمه‌کدها همانطور که انتظار داشتیم به مقداری که در عبارت ۴ بدست آوریم یعنی به عدد 1.9375 میل پیدا می‌کند.

۹.۵

$$\mathbb{E}[(X-\hat{X})^2] = \mathbb{E}[X^2] + \mathbb{E}[\hat{X}^2] - 2\mathbb{E}[X\hat{X}]$$

$$\mathbb{E}[X^2] = \sigma^2$$

$$\mathbb{E}[\hat{X}^2] = \hat{x}_1^2\mathbb{P}(X>0) + \hat{x}_2^2\mathbb{P}(X<0) = \frac{1}{2}\hat{x}_1^2 + \frac{1}{2}\hat{x}_2^2$$

$$\mathbb{E}[X\hat{X}] = \hat{x}_1\int_0^\infty f_X(x)xdx + \hat{x}_2\int_{-\infty}^0 f_X(x)xdx$$

از طرفی می‌دانیم $X \sim \mathcal{N}(0,\sigma^2)$ در نتیجه:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}} \Longrightarrow \int_0^\infty f_X(x)xdx = \frac{\sigma}{\sqrt{2\pi}}, \quad \int_{-\infty}^0 f_X(x)xdx = -\frac{\sigma}{\sqrt{2\pi}}$$

$$\Longrightarrow \mathbb{E}[X\hat{X}] = \frac{\sigma}{\sqrt{2\pi}}(\hat{x}_1 - \hat{x}_2)$$

بنابراین خواهیم داشت:

$$\mathbb{E}[(X-\hat{X})^2] = \sigma^2 + \frac{1}{2}(\hat{x}_1^2+\hat{x}_2^2) - 2\frac{\sigma}{\sqrt{2\pi}}(\hat{x}_1-\hat{x}_2) \tag{۵}$$

حال عبارت فوق را باید بر حسب $\hat{x}_1$ و $\hat{x}_2$ کمینه کرد:

$$\frac{\partial}{\partial\hat{x}_1}\mathbb{E}[(X-\hat{X})^2] = 0 \Longrightarrow \hat{x}_1 - \sqrt{\frac{2}{\pi}}\sigma = 0$$

$$\frac{\partial}{\partial\hat{x}_2}\mathbb{E}[(X-\hat{X})^2] = 0 \Longrightarrow \hat{x}_2 + \sqrt{\frac{2}{\pi}}\sigma = 0$$

$$\hat{x}_1 = \sqrt{\frac{2}{\pi}}\sigma, \qquad \hat{x}_2 = -\sqrt{\frac{2}{\pi}}\sigma \tag{۶}$$

در نهایت خواهیم داشت:

$$\min\mathbb{E}[(X-\hat{X})^2] = (1-\frac{2}{\pi})\sigma^2 \tag{۷}$$