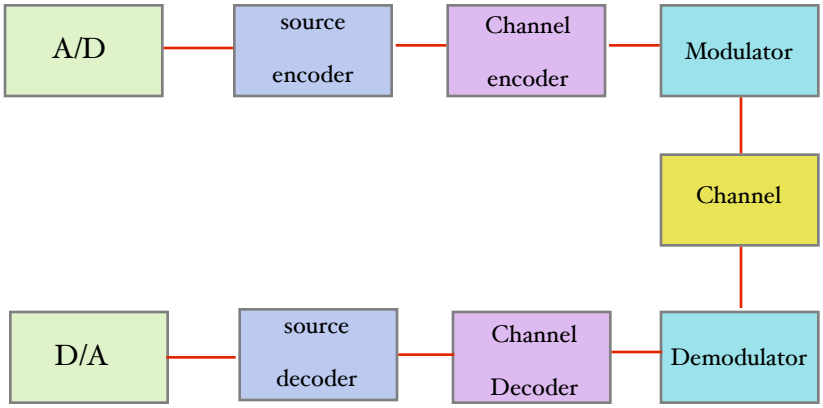
پروژه سیستم ها مخابراتی

علی محرابیان ۹۶۱۰۲۳۳۱ سیده ندا احمدی امیری ۹۶۱۰۱۱۳۲

به نام خداوند مهربان مهربان

مراحل پروژه:



در این پروژه قصد داریم ابتدا پیکسل های یک عکس را با توجه به رنگی که هر پیکسل دارد (عددی بین ۰ تا ۲۵۵) کد گذاری کنیم و به رشته ای از صفر و یک ها تبدیل کنیم. حال با کنار هم قرار دادن کد هر عکس کل عکس را کد گذاری می کنیم. حال بیت های صفر و یک را توسط modulator توسط کانال می فرستیم. در سمت دیگر کانال ابتدا سیگنال کسینوسی را demodulate می کنیم تا رشته صفر و یک فرستاده شده را بازیابی کنیم. سپس با توجه به کد هر رنگ ,رنگ هر پیکسل را مشخص می کنیم و عکس را به این طریق در سمت دیگر بازیابی می کنیم. حال ابتدا در فاز اول نحوه عملکرد هر یک از بلوک های بالا را توضیح می دهیم. سپس در فاز دوم این بلوک ها را کنار هم قرار می دهیم.

فاز اول:

ابتدا قبل از code کردن عکس لازم است به دلیل حجم بالای عکس به دلیل سایز زیاد آن (۵۱۲*۵۱۲) عکس را کوچک کنیم. این کار را توسط تابع imresize انجام می دهیم و سایز عکس را کوچک می کنیم.

```
image=imread('27.gif');
resizedimage=imresize(image,1/8);
```

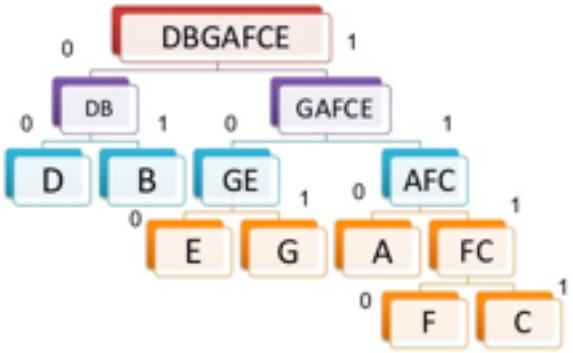
Source Encoder:

این بلوک در واقع هر سمبل را به یک رشته کد باینری تبدیل می کند که در این پروژه هر سمبل یکی از پیکسل های عکس است که رنگ آن در unit-8 مقداری بین255-0 دارد. برای کد گذاری قصد داریم از روش Shannon-fano کد گذاری را انجام می دهیم. توضیحات این روش در ستون مقابل آمده است.

که شکل زیر یک مثال از این روش را بیان می کند.

- Code words by using Shannon-Fano coding

X	D	B	G	A	F	C	E
P	0.32	0.25	0.2	0.1	0.07	0.05	0.01



Shannon -fano

روش شانون یک نوع کد کردن است که برای هر رنگ یک کد در نظر می گیرد. به طوری که هر چه احتمال آن رنگ بیشتر باشد طول رشته باینری تولید شده کوتاه تر است. و هیچ یک از رشته های باینری دنباله یکسانی ندارند.

برای تولید رشته باینری ابتدا احتمال تکرار هر یک از رنگ ها را به دست می آوریم. سپس این احتمالات را به ترتیب نزولی مرتب می کنیم. حال در هر مرحله تمامی سمبل های هر دسته را به دو دسته تقسیم می کنیم و این کار را تا جایی ادامه می دهیم تا به دسته های تکی برسیم.

قواعد دسته بندی Shannon

نحوه دسته بندی به ۲ دسته به این صورت انجام می پذیرد که در نقطه

breaking point مجموع احتمالات سمت چپ از مجموع احتمالات سمت راست بیشتر باشد اما این اختلاف کم ترین مقدار خود را داشته باشد.

نحوه کد گذاری

در هر مرحله به دسته جدا شده سمت چپ یک بیت صفر و به دسته جدا شده سمت راست یک بیت یک اضافه می گردد. این روند کد گذاری سبب می شود سلسله بیت های هیچ دو سمبلی مشابه نباشد. در نتیجه در هنگام decode کردن هیچ دو سمبلی با هم اشتباه گرفته نمی شوند.

حال برای پیاده سازی shannon fano ابتدا یک ماتریس ۳ سطری تعریف کردیم که سطر دوم آن تمامی رنگ ها(اعداد بین ۰ تا ۲۵۵) و سطر اول آن احتمل تکرار هر یک از رنگ ها در عکس مورد نظر بود. و سطر سوم آن بیان گر codeword آن سمبل(رنگ) بود.

```
for i=1:64
   for j= 1:64
      a=resizedimage(i,j);
      probability(1,a+1)=probability(1,a+1)+1;
   end
end
[temp, order] = sort(probability(1,:),'descend');
probabilitysorted =probability(:,order);
```

حال از آن جایی که کد گذاری رنگ هایی که در عکس استفاده نشده اند بی معنی است در نتیجه ابتدا آن رنگ ها را کنار گذاشته سپس ماتریس را به تابع shannon می دهیم.

نحوه کارکرد تابع Shannon-encoder :

از آن جایی که اجرا این تابع باید تا رسیدن به دسته های تک ادامه پیدا کند. لازم است این تابع به صورت بازگشتی تعریف شود تا بتوانیم تابع را در خود تابع صدا کنیم.حال از آن جایی که نقطه شروع و پایان دسته های ورودی تابع و codeword ساخته شده در هر دفعه ای که تابع صدا می گردد تغییر می کنند. در نتیجه لازم است علاوه بر تغییر این متغیر ها در داخل تابع آن ها را به عنوان ورودی نیز به تابع بدهیم تا تغییرات آن ها تا مراحل آخر اعمال شود.

برای ساختن codeword ابتدا codeword تمام سمبل ها را برابر یک قرار می دهیم(این یک زاید است و در مراحل بعدی حذف خواهد شد.) سپس در هر جدا سازی دسته ها codeword دسته سمت چپ را ۱۰ برابر کرده و با صفر جمع می کنیم(رقم صفر را سمت راست عدد قرار می دهیم.) و codeword دسته سمت راست را ۱۰ برابر کرده و با یک جمع می کنیم. به این ترتیب در هر مرحله صدا کردن تابع codeword متناسب با روش شانون تغییر می کند. نحوه اجرای کد به صورت خلاصه در صفحه بعد آمده است.

حفظ و تغییر codeword هر سمبل

با وجود recursive بودن تابع

```
function codeword=shannon_encoder(begin_point,end_point,p,code)
 for i=begin_point:end_point
    if sum(p(begin_point:i))>sum(p(i+1:end_point))
       break;
    end
 end
 code(begin_point:i)=10*code(begin_point:i)+0;
 code(i+1:end_point)=10*code(i+1:end_point)+1;
 high_point=i;
 low_point=i+1;
  if ((low_point+1)<end_point)
     codeword=code;
    code=shannon_encoder(low_point,end_point,p,code);
   if ((begin_point+1)<high_point)
    codeword=code;
    code=shannon_encoder(begin_point,high_point,p,code);
   end
  end
```

صدا کردن مدام تابع تا جایی که به تک سمبل برسیم.

راه دیگر برای shannon coding :

در این روش به جای تعریف تابع بازگشتی قصد بر آن داریم تا خود درخت را بسازیم. به این ترتیب که در ابتدای کار همه سمبل ها عضو درخت اند. در هر مرحله با استفاده از تابع split سمبل ها را به دو بخش تقسیم می کنیم. اگر تعداد سمبل ها به یک (یا صفر) رسید یعنی به شاخه های درخت رسیده ایم. در غیر این صورت اگر سمبل های هر دسته بیش از یک بود , سمبل ها را در درخت قرار میدهیم و مجددا مرحله split را برای هر دسته تکرار می کنیم. تا جایی که همه سمبل ها در شاخه ها قرار گیرند. نحوه کد گذاری در این روش هم مانند روش قبل است

```
for i=1:size(tree,2)
    x=tree{j,i};
    if size(x,2)>1
    [y,z]=spilit(tree{j,i});
          tree{j+1,nk}=y;
    tree{j+1,nk+1}=z
    nk=nk+2;
    elseif size(x,2)==1
    c=c+1;
    leaf{1,qq}=x;
    qq=qq+1;
```

نحوه encode کردن :

حال که codeword مربوط به هر رنگ را داریم برای code کردن کل عکس لازم است codeword تمامی ۴۰۹۶ پیکسل عکس را کنار هم قرار دهیم. به این عمل encode کردن می گویند. برای انجام این کار صرفا لازم است رنگ پیکسل ها را به ترتیب تشخیص دهیم و codeword مربوطه آن رنگ را به رشته بیت ها بیافزاییم. برای راحت تر شدن این فرایند و مشکلات مربوط به ذخیره سازی رشته بیت باینری توسط متغیر intiger ابتدا codeword ها را به رشته ای از کاراکتر ها تبدیل کردیم و سپس رقم یک زاید اول همه آن ها را از بین بردیم. سپس این رشته ها را در کنار هم قرار دادیم.

```
codeword=' ';
for i=1:size(resizedimage,2)
   for j=1:size(resizedimage,1)
      [r,c]=find(xx==resizedimage(j,i));
      a=num2str(xx(r+1,c));
      a=a(2:end);
      codeword=strcat(codeword,a);
   end
end
code=codeword;
```

یک زاید را حذف می کند.

کد پیکسل منظور را به کد ساخته شده تا الان اضافه می کند.

نحوه decode کردن :

برای دیکود کردن codeword ساخته شده از ۲ نشانه گر استفاده کردیم. به طوری که نشانه گر اول codeword قرار می گیرد و نشانه گر دوم را جلو می بریم سپس در codeword های مربوط به رنگ ها به دنبال رشته بین دو نشانه گر می گردیم. نشانه گر دوم را تا جایی جلو می بریم که اولین codeword با معنا پیدا شود.

حال که رنگ اولین پیکسل را تشخیص داده ایم , نشانه گر اول را به اندازه طول codeword جلو می بریم و مجددا مراحل بالا را تکرار می کنیم تا جایی که رنگ هر ۴۰۹۶ پیکسل عکس تشخیص داده شود.

```
for k=1:size(nmb,2)
    if string(code(j:i))==string(nmb{1,k})
       yy(1,p)=i;
       yy(2,p)=j;
       yy(3,p)=nmb{2,k};
       o=o-length(code(j:i));
       p=p+1;
       j=i+1;
       i=j+1;
       break;
```

چند نمونه از عکس های encode و decode شده توسط تابع shannon :

عکس اولیه

عکس decode شده

بلوک های modulator و demodulator :

برای ارسال بیت های صفر و یک ساخته شده توسط encoder بهتر است ابتدا متناسب با صفر یا یک بودن دیتا سیگنالی با فرکانس مرکزی بسیار بالاتر تولید کنیم و سپس آن را توسط کانال ارسال کنیم تا بتوانیم طول آنتن ارسال کننده را به صورت چشم گیری کاهش دهیم. حال modulator ما در این پروژه M-ary PWM است.

خاصیت این PWM این است که اگر تعداد حالت های کوانتیزاسیون برابر Mحالت باشد پالس تولید شده M حالت مختلف می تواند داشته باشد (m=۱, ...,M) که هر چه m بزرگ تر شود دامنه پالس کوتاه تر و طول زمانی پالس بلند تر می شود.حال ابتدا تابع ساخت پالس ها را به صورت کلی می نویسیم سپس در فاز دوم از یک حالت خاص آن استفاده می کنیم.

این تابع به این صورت کار می کند که اگر بخواهیم N دیتا ارسال کنیم , هر Ts ثانیه یکی از دیتا ها را میفرستیم در نتیجه ارسال همه دیتا ها N*Ts ثانیه طول خواهد کشید. حال با توجه به آن که کدام پالس از بین M پالس موجود را بخواهیم ارسال کنیم شکل پالس از فرمول زیر تعیین می شود.

$$S_m = \begin{cases} \sqrt{\frac{۲M}{mT_s}} cos(۲\pi f_c t) & ۰ \le t \le \frac{mT_s}{M} \\ ۰ & otherwise \end{cases}$$

$$m = ۱, ..., M$$

حال با توجه به این که نمی توانیم سیگنال را به صورت پیوسته ارسال کنیم ابتدا نیاز است آن را نمونه برداری کنیم.حال اگر فرض کنیم فرکانس نمونه برداری ما fs باشد برای تولید سیگنال ارسال کننده کافی است

سیگنالی با طول زمانی N*Ts ثانیه و نرخ نمونه برداری fs تولید کنیم . و مراحل بالا را به ترتیب برای هر دیتا موجود در دنباله دیتا ها اجرا کنیم.

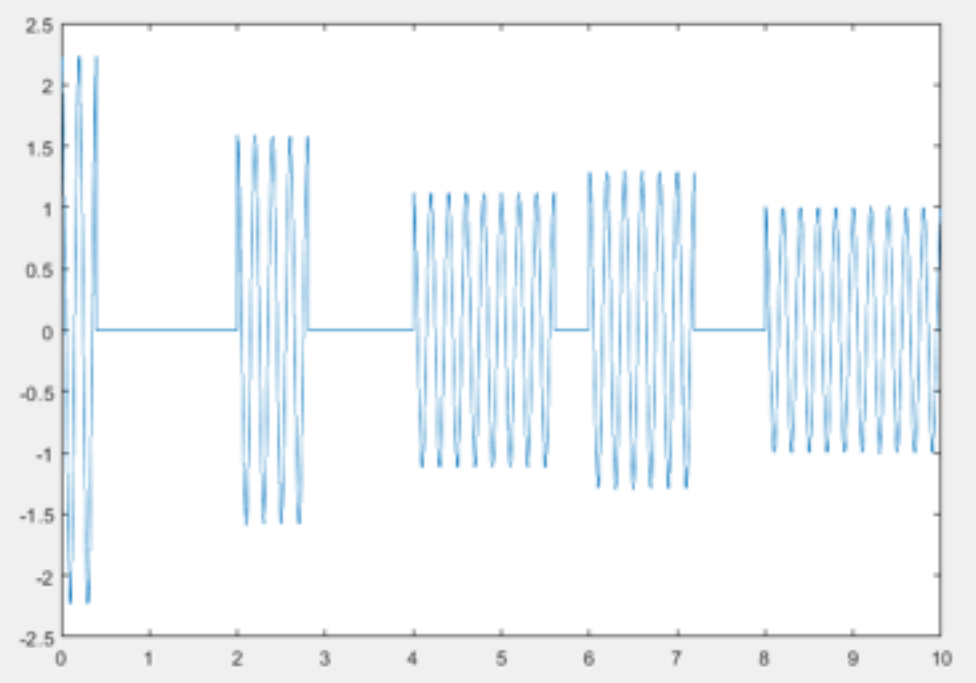
```
tTotal=linspace(0,N*Ts,N*Ts*fn);
modulated=zeros(1,n);
 for m=1:N
   u=sqrt(2*M/(sequence(m)*Ts));
   t=linspace(0,sequence(m)*Ts/M,sequence(m)*Ts*fn/M);
   modulated(1,(1+fix((m-1)*Ts*fn)):(length(t)+fix((m-1)*Ts*fn)))=u*cos(2*pi*fc*t);
 end
end
```

تعیین ضریب

تولید کسینوس با ضریب و طول مذکور

برای مثال برای یک پالس با طول ۵ و sequence زیر شکل موج مادوله شده به صورت زیر در می آید.

Sequence= 1-2-4-3-5



که به خوبی مشاهده می شود هر چه m کوچک تر پالس کوتاه تر و دارای دامنه بزرگتری است.

:demodulator

برای دمولاتور از روش همبستگی سنج یا correlator استفاده کردیم. برای این کار ابتدا در تابع baseband یک ماتریس M سطری تعریف کردیم که در هر سطر آن پالس های پایه ممکن را قرار دادیم. یعنی پالس پایه مربوط به m=1 در سطر اول و پالس پایه مربوط به m=M در سطر آخر قرار دارد. حال از آن جایی که در سیگنال N*Ts ثانیه ای دریافت شده هر Ts ثانیه شباهت بیشتری با یکی از سیگنال های پایه دارد در نتیجه هر Tsثانیه از سیگنال را با تمامی سطر های ماتریس پایه ساخته شده همبستگی می گیریم. و در ماتریس جدیدی ذخیره می کنیم. اگر این کار را برای همه دیتا های ممکن تکرار کنیم نتیجه کار یک ماتریس است که میزان همبستگی پالس i ام را با هر پالس پایه نشان می دهد. برای محاسبه correlation کافی است سیگنال را در conjucate سیگنال mirror شده کانوالو کنیم که از آن جایی که سیگنال حقیقی و گسسته است نتیجه مجموع ضراب درایه ای دو ماتریس را به ما می دهد.

```
for m=1:M
   for n=1:N
      c(m,n)=sum(Sm(m,:).*modulated(1,1+(n-1)*Ts*fs:n*Ts*fs));
   end
end
```

detector:

از آن جایی که هر چه قدر سیگنال ما به یکی از پالس های پایه شبیه تر باشد , میزان کورلیشن بیشتری با آن پالس پایه نسبت به سایر پالس های پایه خواهد داشت. در نتیجه در حالت ایده آل می توان گفت ماکسیمم مقدار corelation در هر ستون نشان دهنده شماره پالس فرستاده شده است. اما در واقعیت نویز وجود دارد و به علت وجود نویز گاهی نمی توان حالت ایده آل بالا را در نظر گرفت. در نتیجه برای detect کردن پالس نقطه های threshhold را برابر وسط بازه های correlation در صورت ایده آن بودن قرار می دهیم. برای مثال در M=2حد های ایده آل برابر 1 , 0.69 است در نتیجه threshhold وجود پالس یک یا صفر را بزرگتر بودن correlation آن با پاس پایه از 0.85 تعریف می کنیم. به این ترتیب اگر سیگنال بالا را به ترتیب به demodulator و decoder بدهیم sequence اولیه تعریف شده به درستی بازیابی می شود.

```
>> detector

detector =

     1     2     4     3     5

>> Sequence

Sequence =

     1     2     4     3     5
```

فاز دوم :

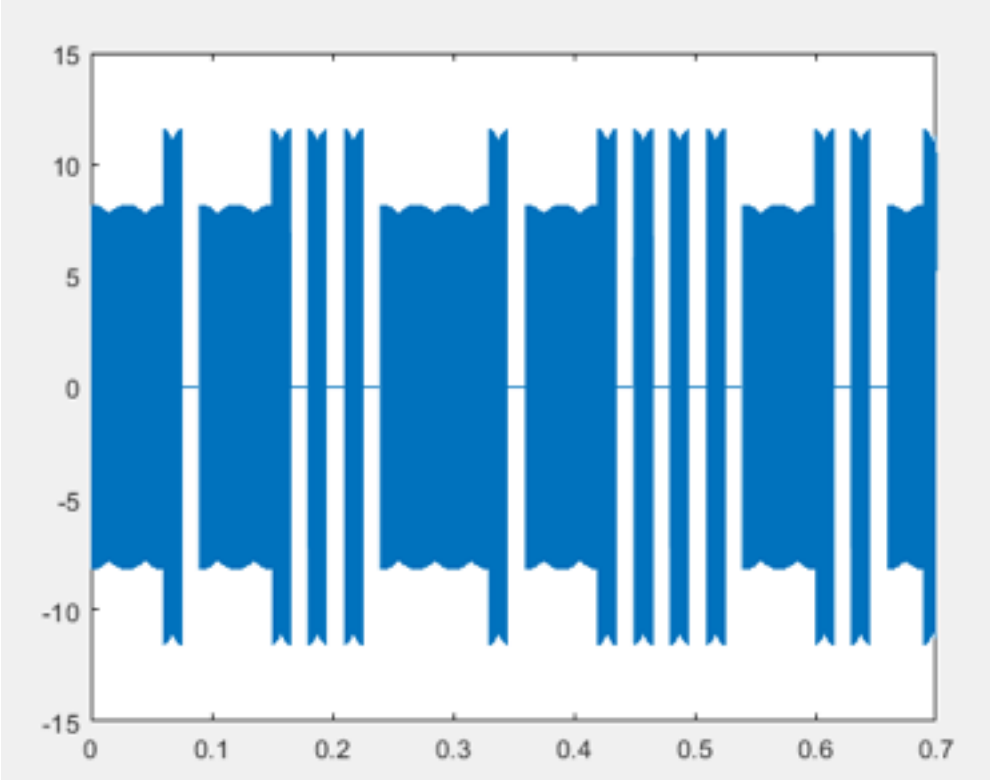
الف) حال می خواهیم بلوک های تعریف شده در مراحل بالا را در کنار یکدیگر قرار دهیم تا به وسیله آن بتوانیم یک عکس را از یک طرف ارسال کنیم و از طرف دیگر دریافت کنیم. ابتدا با استفاده از source-encoder پیکسل های عکس را به صورت codeword در آورده و یک رشته بیت باینری تولید می کنیم.

حال رشته بیت را به قطعه هایی با طول مشخص تقسیم می کنیم به طوری که عددی بین ۱ و M شود. از آن جایی که در سوال خواسته شده که M برابر دو باشد در نتیجه باید رشته را به صورت تک بیت تک بیت ارسال کنیم.برای مادوله کردن هر بیت ارسال شده در صورت صفر بودن m=۱ قرار می دهیم و در صورت یک بودن m=2 قرار می دهیم. با انجام این کار که کد آن در زیر آمده است Sequence مربوطه را تولید کردیم. حال با استفاده از پارامتر های شبیه سازی داده شده در سوال تابع modulator را صدا می کنیم. برای نمونه بازه کوتاهی از سیگنال تولید شده یکی از عکس های مدوله شده در شکل زیر آمده است.

```
for i=1:N
   if codeword(1,i)=='0'
      Sequence(1,i)=1;
   else
      Sequence(1,i)=2;
   end
end
```

جدول ۳: پارامترهای شبیه‌سازی

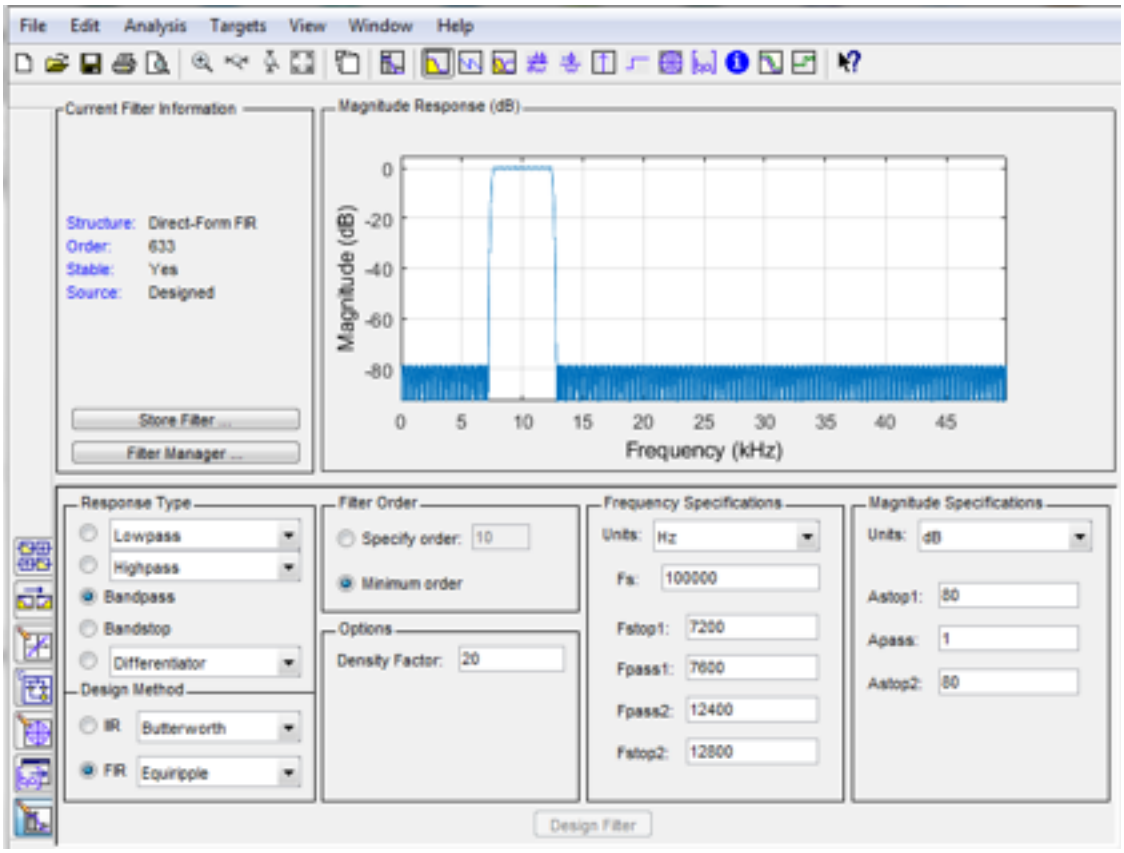
پارامتر	مقدار
فرکانس نمونه‌برداری	۱۰۰KHz
طول هر سمبُل (T_s)	۳۰ms
فرکانس حامل (f_c)	۱۰KHz
فرکانس مرکزی کانال	۱۰KHz
پهنای باند کانال	۵KHz
M	۲



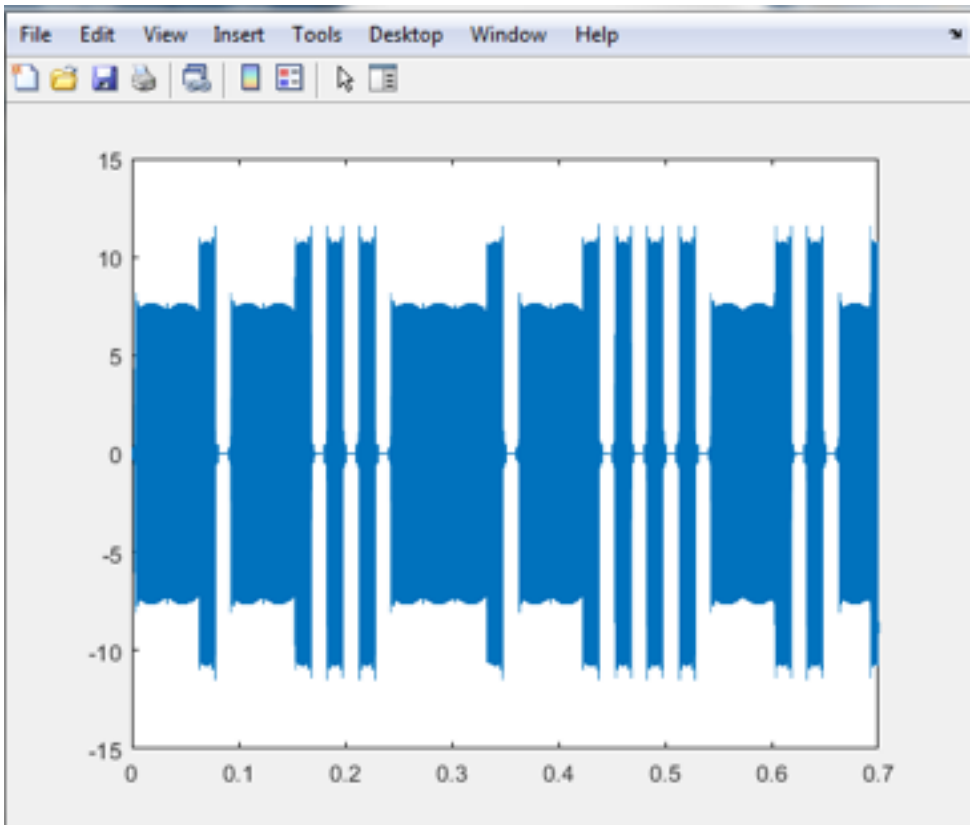
ب)

حال برای تست فرض می کنیم کانال ایده آل است یعنی نویزی در کانال به سیگنال اضافه نمی شود. در این صورت تنها محدودیتی که کانال برای سیگنال مربوطه ایجاد می کند. پهنای باند محدود آن است(پهنای باند ۵ کیلوهرتز با فرکانس مرکزی ۱۰ کیلوهرتز.) در نتیجه در عبور از کانال صرفا سیگنال را از یک فیلتر میان گذر عبور می دهیم.

این فیلتر را توسط filter designer تولید کردیم که شکل آن به صورت زیر است:



حال به دلیل حذف فرکانس های بالا نقاط شکستگی هر بازه نرم تر می شود اما شکل کلی سیگنال همان گونه باقی می ماند. یک نمونه از سیگنال های عبوری در شکل زیر آمده است:



حال که سیگنال از کانال عبور کرد وارد demodulator می شود. در این مرحله ابتدا با توجه به طول سیگنال تشخیص می دهیم تعداد کل بیت های ارسالی چند است. سپس هر پالس را با پالس های پایه یعنی So و S1 کورولیشن می گیریم و در یک ماتریس ذخیره می کنیم . حال هر سیگنال اگر از threshhold مربوطه بیشتر شود تشخیص می دهیم بیت صفر یا یک فرستاده شده است.

```
for n=1:N
    down=1+floor((n-1)*Ts*fs);
    up=length(s0)+floor((n-1)*Ts*fs);
    c(1,n)=sum(s0.*channel(1,down:up));
    c(2,n)=sum(s1.*channel(1,down:up));
end
```

```
th=((c(1,1)+c(2,1))/2);
for i=1:N
    if c(1,i)>th
        detector(1,i)=0;
    else
        detector(1,i)=1;
    end
end
```

حال در ماتریس detector ساخته شده بیت ها به ترتیب قرار گرفته است.حال اگر این بیت ها را کنار هم قرار داده و یک رشته تولید کنیم codeword فرستاده شده را بازیابی کردیم. تنها کافی است رشته کد بازیابی شده را به decoder بدهیم تا عکس اولیه را بازیابی کنیم.

پ)

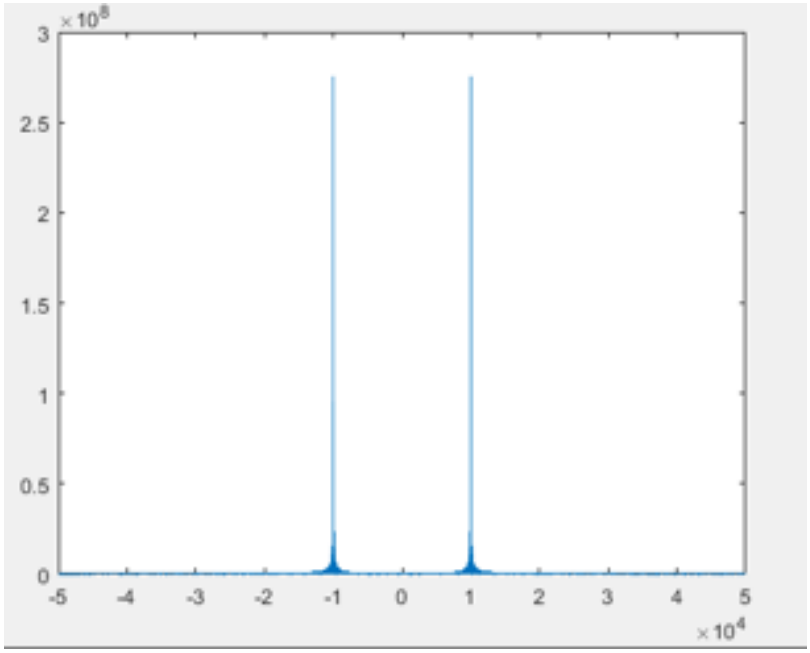
برای به دست آوردن تبدیل فوریه سیگنال و نرمالیزه کرن آن از دو تابع fft و fftshift استفاده می کنیم. حال از آن جایی که می دانیم تبدیل فوریه سیگنال گسسته طیفی بین fs/2- تا fs/2 دارد فرکانس را متناسبا تعریف می کنیم. حال می دانیم طیف سیگنال برابر Xf.*conj(Xf)=|Xf|^2

چند نمونه از طیف ها در صفحه بعد کشیده شده است که مشاهده می شود طیف ها شکل نسبتا یکسانی در حوالی فرکانس ۱۰ کیلو هرتز دارند. که به علت یکسان بودن تقریبی پالس ها (هر عکس مجموعه ای از پالس های صفر و یک است که شکل آن ها ثابت است.) و همچنین وجود cos با فرکانس حامل ۱۰ کیلوهرتز این طیف قابل پیش بینی بود.

حال برای به دست آوردن پهنای باند ابتدا انرژی کل را محاسبه می کنیم که برابر است با مجموع تمامی نقاط طیف. سپس نقطه ای را پیدا می کنیم که که اگر تا آن نقطه از سیگنال را نگه داریم ۹۹ درصد انرژی حفظ می شود. که برای ۳ عکس مختلف فرکانس قطع تقریبا برابر۱۰.۲ کیلو هرتز به دست آمد. که پهنای باند تقریبا برابر ۴۰۰ هرتز به دست می آید که این پهنای باند به شدت از پهنای باند کانال کوچک تر است.

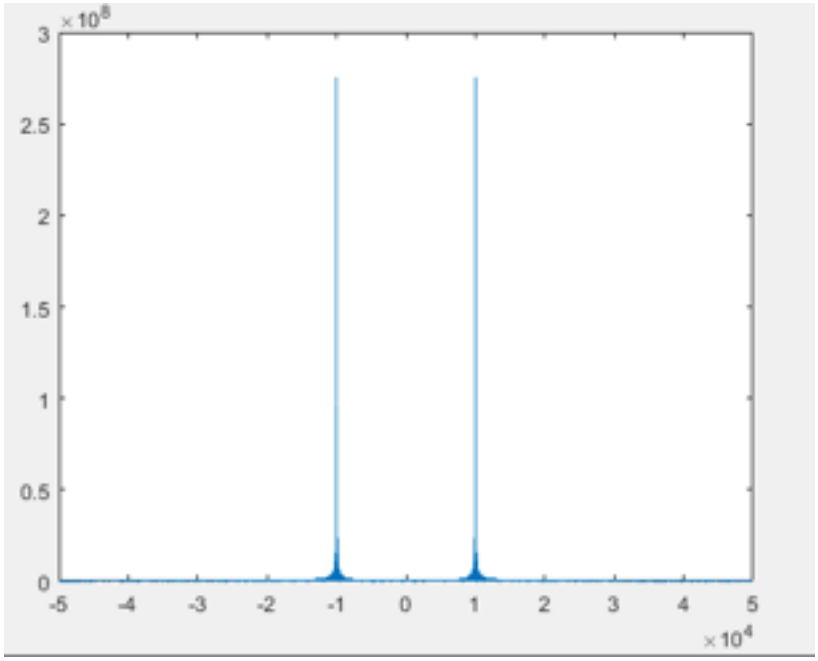
fm =

1.0200e+04



fm =

1.0200e+04



```
S=fourier.*conj(fourier);
Etotal=sum(S);
l=length(fourier);
middle=(l)/2;
Etest=S(middle);
for i=0:(l/2)-1
   Etest=Etest+S(middle+i)+S(middle-i);
   if(Etest>0.99*Etotal)
      breakingpoint=i;
      break
   end
end
```

fm=fs*breakingpoint/l;

ت) در این قسمت ابتدا می خواهیم نویز را به سیگنال مادوله شده اضافه کنیم. سپس سیگنال دارای نویز را وارد کانال کنیم. می دانیم نویز دارای توزیع گوسی دارای میانگین صفر و واریانس σ است . برای تولید این نویز از تابع randn استفاده می کنیم. به این صورت که ابتدا یک ماتریس به طول سیگنال modulated تعریف می کنیم. و در هر خانه آن یک randn می زنیم به این ترتیب سیگنال گوسی با واریانس یک تولید کردیم. برای تغییر واریانس آن باید تمام عناصر بردار را در رادیکال واریانس ضرب کنیم. حال نویز تولید شده را با سیگنال جمع می کنیم و به کانال می دهیم. حال سیگنال خارج شده از کانال را به دمولاتور می دهیم تا رشته بیت فرستاده شده را تشخیص دهد. حال با توجه به وجود نویز ممکن است تعدادی از بیت ها اشتباه تشخیص داده شوند. برای تشخیص تعداد بیت های اشتباه اختلاف رشته بیت تشخیص داده شده را با رشته بیت فرستاده شده حساب مب کنیم.

حال اگر ۰ را یک تشخیص داده باشیم در آن خانه -۱ و اگر یک را صفر تشخیص داده باشیم در آن خانه ۱ قرار می گیرد.در نتیجه قدر ممطلق اندازه در همه خانه هایی که اشتباه تشخیص داده شده اند برابر ۱ است. اگر مجموع قدر مطلق تمام خانه ها را حساب کنیم تعداد بیت خطا را می یابیم.

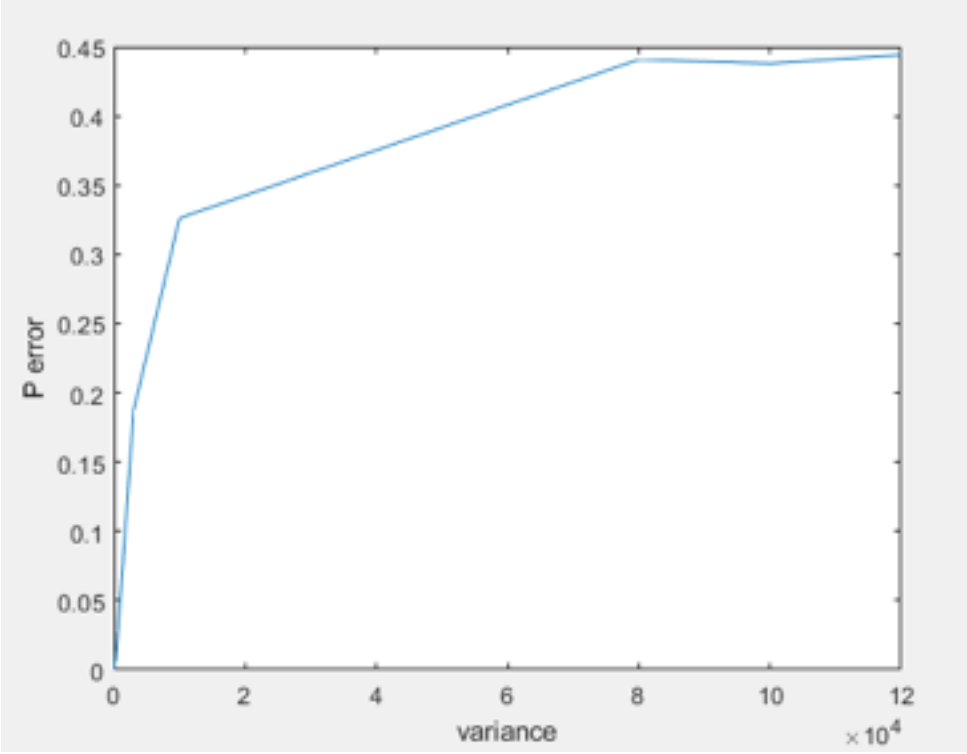
```
var=200;
noise=randn(1,length(modulated))*sqrt(var);
modulatorafternoise=modulated+noise;
a=modulatorafternoise;
```

```
%% Error
error=sum(abs(code-codenew));
```

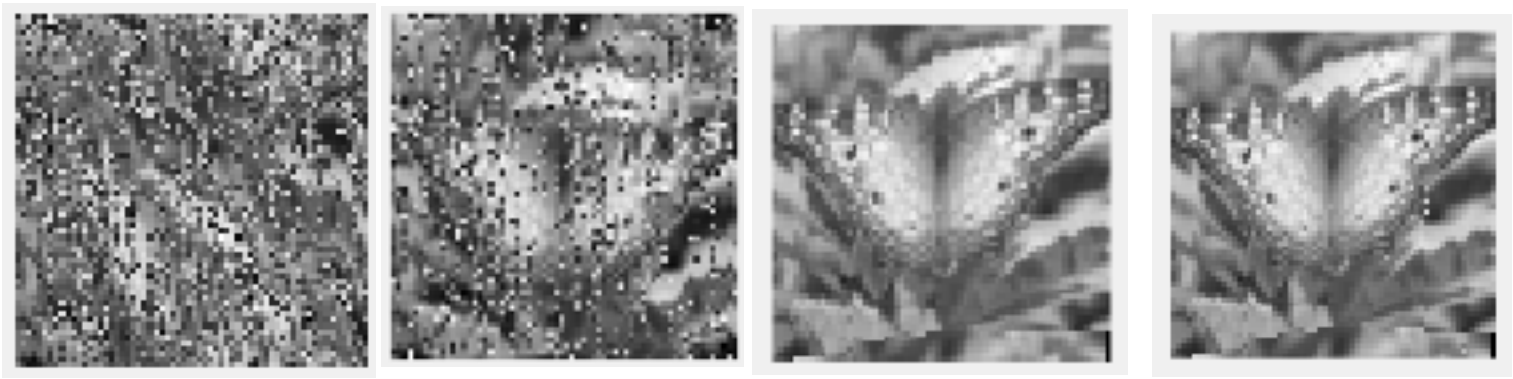
variance	number of error
0	0
4	0
16	0
81	100
225	177
625	820
3000	5402
10000	9384
80000	12680
100000	12597
120000	12733

نمودار احتمال خطا بر حسب واریانس:

تعداد کل دیتا ها در این عکس =۲۸۷۶۵



چند نمونه عکس دریافتی از نویز کم تا نویز بسیار زیاد.



توجیه رفتار حدی احتمال خطا:

اگر واریانس نویز خیلی زیاد شود در واقع می توانیم از خود سیگنال در برابر نویز صرف نظر کنیم. حال در این حالت یک نویز یکنواخت داریم که واریانس آن بسیار بزرگ است. حال از آن جایی که احتمال خطا برابر است با $Q(d/2\ \sigma)$ حال اگر واریانس بسیار بزرگ باشد احتمل خطا برابر $Q(o)$ می شود که همان 0.5 است اگر در نمودار نگاه کنیم نیز احتمال خطا به 0.5 میل می کند.

نکته:

در این مرحله نوع دیکودر را تغییر دادیم به این صورت که از ابتدای کد ساخته شده شروع کرده،در هر مرحله،کد های معتبر ساخته توسط رنگ ها با آن مقایسه می شود تا زمانی کد مشابه آن پیدا شود.پس رنگ موردنظر پیدا شده و کد مربوط به آن رنگ از کداصلی حذف می شود.این فرآیند آن قدر تکرار می شود تا عمل دیکود به طور کامل انجام شود

ث)

برای محاسبه SNR سیگنال در کد نوشته شده SNR را در ۳ مرحله مختلف حساب می کنیم.

SNR transfer :

قبل از عبور از فیلتر کانال توان سیگنال برابر توان سیگنال مادوله شده و توان نویز برابر توان سیگنال گوسی ساخته شده است و SNR از تقسیم توان سیگنال به توان نویز به دست می آید. که به روش زیر محاسبه می شود:

```
Pt=sum(modulated.*modulated);
Pnt=sum(noise.*noise);
SNRt=Pt/Pnt;
```

SNR receive:

بعد از عبور سیگنال به کانال و قبل از دادن آن به دمولاتور قبل به علت عبور نویز از فیلتر توان آن تغییر کرده است. ما توان سیگنال بدون نویز را می توانیم با عبور سیگنال مادوله شده از کانال به دست بیاوریم. حال اگر توان سیگنال با نویز را حساب کنیم از این توان کم کنیم توان نویز بعد فیلتر به دست می آید و در نتیجه SNR قبل از دمولاتور محاسبه می شود.

```
Pr=sum(idealchannel.*idealchannel);
Pnr=sum(channel.*channel)-Pr;
SNRb=Pr/Pnr;
```

SNR after demodulator:

در این مرحله سیگنال عبور کرده از دمودولاتور یک رشته بیت به ما می دهد برای یافتن SNR ابتدا از بیت های تشخیص داده شده مدولاسیون می گیریم. سپستوان سیگنال حاصله را حساب می کنیم و از توان سیگنال ورودی دمولاتور کم می کنیم تا توان نویز جدید به دست بیاید. حال SNR خروجی را به

دست می آوریم. در مرحله ارسال از آن جایی که هنوز تمام فرکانس های نویز حفظ شده اند کمترین SNR را داریم و در خروجی از آن جایی که دمولاتور نقش نویز را کم رنگ می کند بیش ترین SNR را شاهد هستیم. البته مشخص است با افزایش واریانس نویز مقدار SNR در تمامی بخش ها کاهش خواهد یافت.

```
Sequencenew=detector+1;
[demodulated,t]=modulator(M,N,Sequencenew,Ts,fc,fs);
demodulated=filter(Bandpass2,1,demodulated);
Pl=sum(demodulated.*demodulated);
Pnde=Pr-Pl;
SNR=Pl/Pnde;
```

چند نمونه از SNR به ازای نویز با واریانس های مختلف:

VAR=16

```
>> SNRt
SNRt =
    2.0848
>> SNRb
SNRb =
   18.5104
>> SNR
SNR =
   Inf
```

VAR=100

```
SNRt =
    0.3335
SNRb =
    2.9741
SNR =
   2.7059e+05
```

VAR=800

```
SNRt =
    0.0417
SNRb =
    0.3709
SNR =
  -2.4184e+03
```

مشاهده می شود در واریانس کوچک اثر نویز تا حد خوبی توسط دمولاتور از بین می رود برای مثال در واریانس ۱۶ مشاهده می شود SNR خروجی دمولاتور همچنان بی نهایت است در حالی که SNR ورودی ۱۸ است.

اما در واریانس های بزرگ SNR به شدت کاهش می یابد.

نمونه تصویر ر بخش قبل آنده است به ازای واریانس ۲۰۰ و ۸۰۰ برای عکس دیگر داریم:

عکس با نویزی با وارینس ۲۰۰ همچنان تا حد خوبی بازیابی شده است.

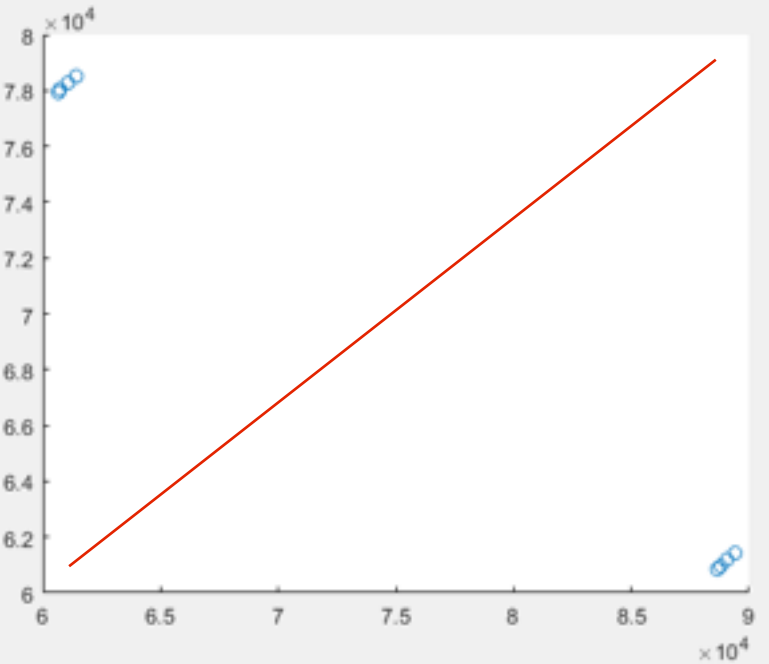
ج)

در قسمت دمولاتور دیدیم که برای detect کردن بیت ها از ماتریس constellation کمک می گرفتیم که در این ماتریس هر سطر correlation سیگنال مادوله شده آن دیتا با یکی از بردار های پایه است. در این قسمت می خوایم این بردار ها را رسم کنیم و تاثیر نویز بر آن ها را مشاهده کنیم . برای این کار با استفاده از تابع scatter یک سطر ماتریس را برحسب سطر دوم رسم می کنیم.

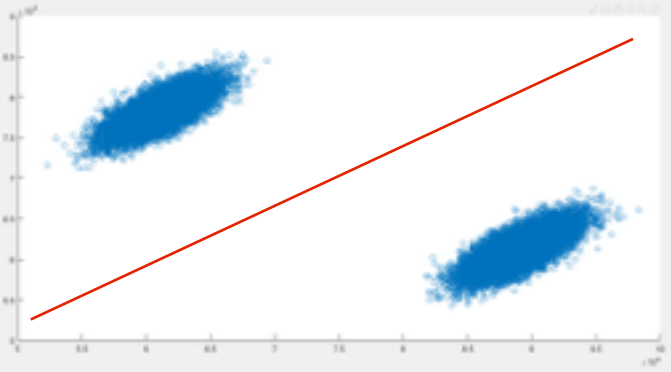
این نمودار را به ازای ۶ واریانس مختلف رسم می کنیم:

```
Part G %%
;figure(11)
;scatter(c(1,:),c(2,:))
```

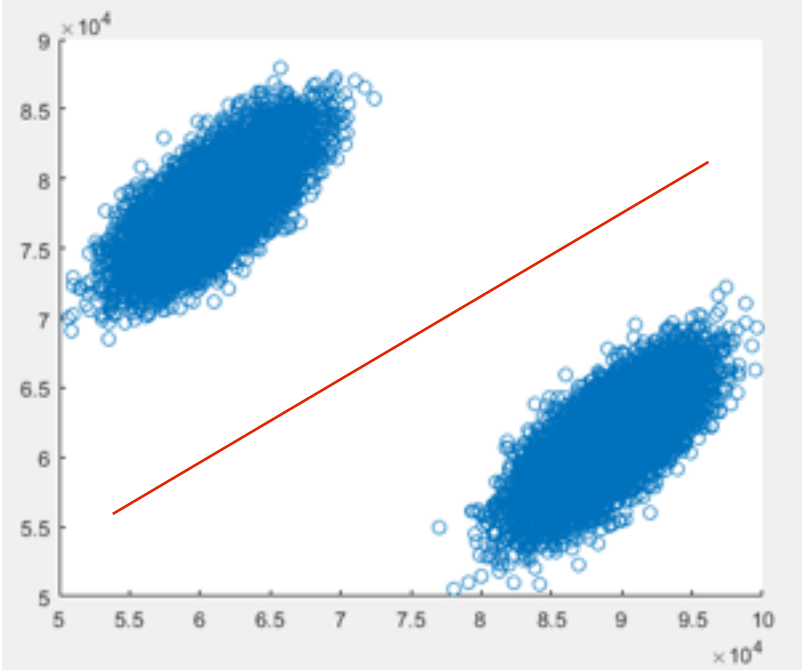
var 0



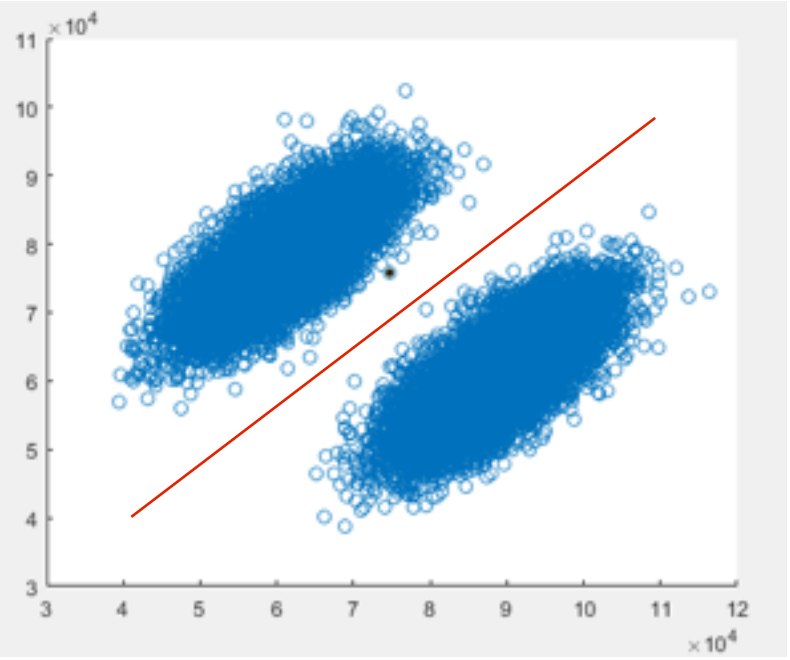
var50



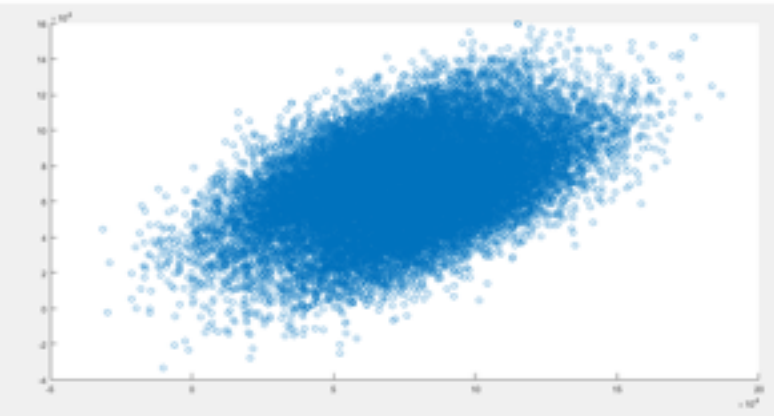
var 500



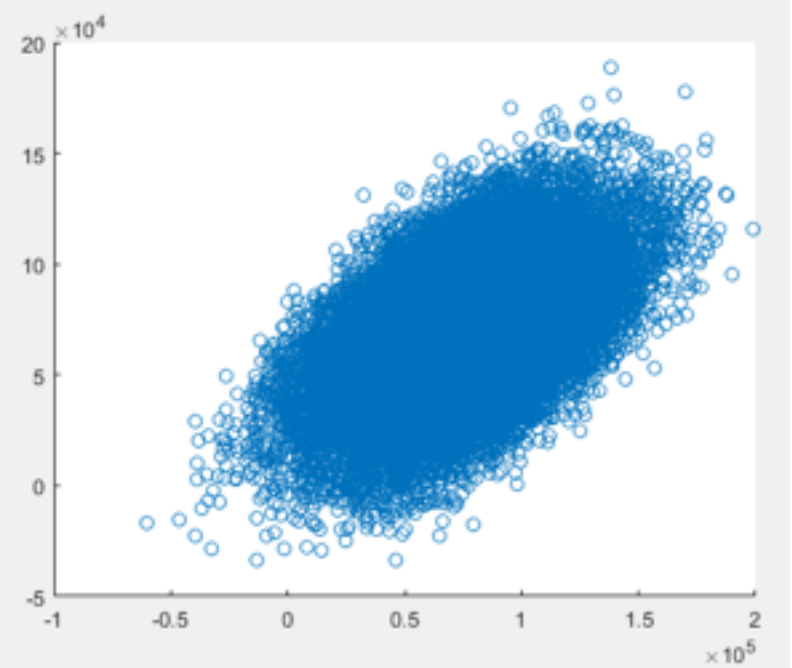
var=1000



var 8000

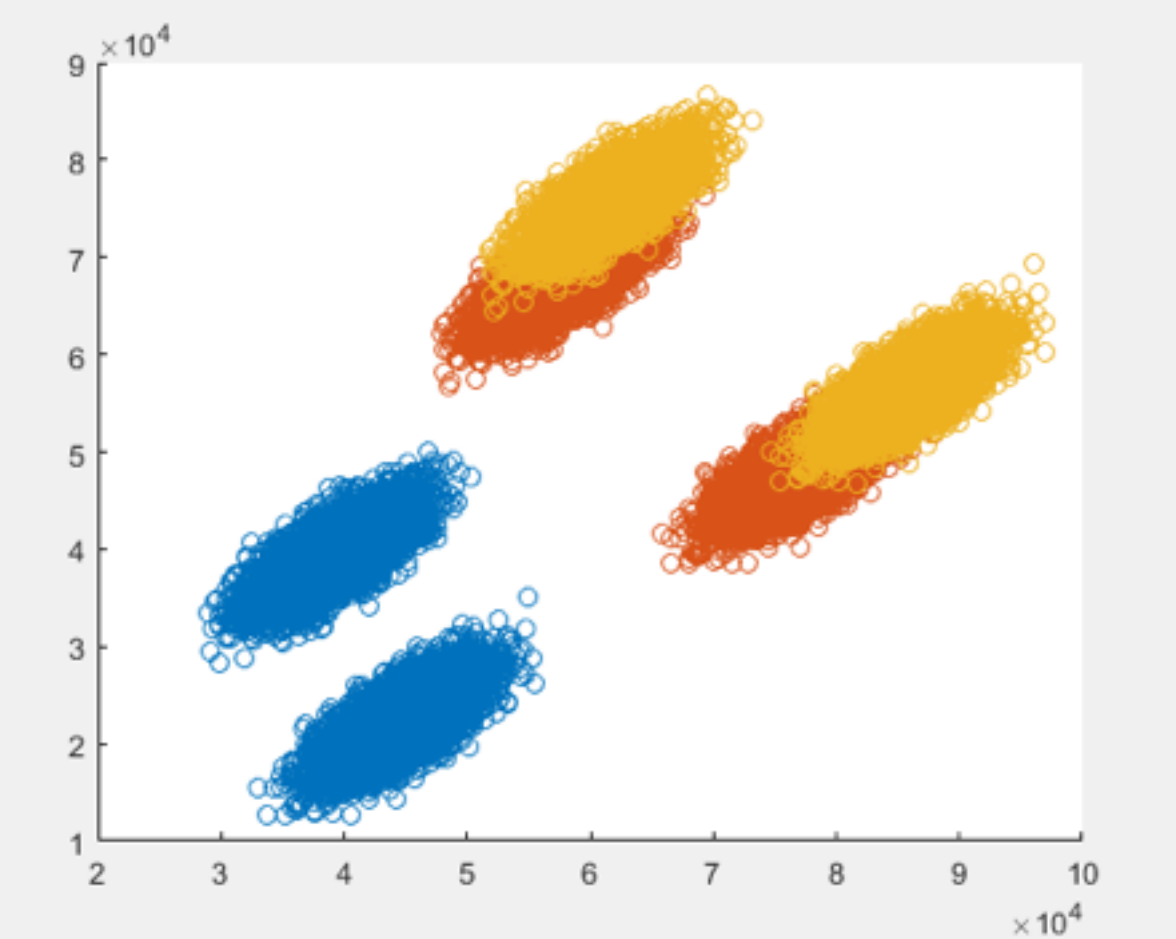


var=10000



مشاهده می شود که در فرکانس های پایین به علت کم بودن اثر نویز کورلیشن سیگنال با بردار های پایه تقریبا مقادیر یکسانی می دهد(بنا به این که سیگنال ارسالی صفر یا یک باشد دو مقدار ممکن داریم.) که این مسیله در واریانس برابر صفر به خوبی مشاهده می شود.همچنین تمایز بین دو کورلیشن ممکن در این واریانس بسیار زیاد است. در نتیجه توانایی تفکیک آن ها بسیار ساده تر است اما با اضافه شدن واریانس به دلیل تغییر شکل رندوم ایجاد شده در سیگنال هم همبسگی های مربوط به هر بردار پایه پخش تر می شود هم تمایز بین ۰ و ۱ کم تر می شود. در نتیجه احتمال تشخیص صحیح بیت ها کاهش می یابد. و در واریانس های بالا پراکندگی زیاد و تماز آن چنان کم می شود که بیت ها غیر قابل تشخیص می شوند.

در این روش تاثیر coherent نبودن علاوه بر وجود نویز را مشاهده می کنیم. به این ترتیب که cos مدوله کننده یک شیفت فاز نسبت به حالت قبل پیدا می کند. واضح است هر میزان شیفت فاز بیشتر باشد تغییر شکل سیگنال بیشتر است در نتیجه تشخیص بیت ها سخت تر می شود. برای راحت تر حس کردن این موضوع بردار های constellation مربوط به ۳ شیفت فاز مختلف $\pi/3$ و $\pi/6$ و $\pi/12$ را روی یک نمودار رسم کردیم. که آبی مربوط به فاز $\pi/3$ و رنگ قرمز مربوط به $\pi/6$ رنگ زرد مربوط به فاز $\pi/12$ است .



تمایز در فاز های کمتر بیشتر است این مسیله تشخیص را راحت تر می کند.