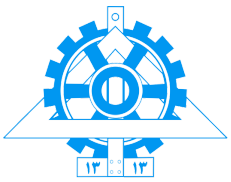
تمرین کامپیوتری چهارم

📡 سیگنال ها و سیستم ها 📡



مجید صادقی نژاد - 810101459

آرمان خورشیدی - 810101417

پاییز ۱۴۰۳

استاد: دکتر اخوان

سوال اول

بخش اول

از آن جا که دیتا ست هم در تابع encode و هم در تابع decode استفاده می شود تابعی می نویسیم تا دیتاست خواسته شده ی صورت پروژه را در یک متغییر ذخیره کرده تا پس از این نیاز توانایی دسترسی به دیتاست را داشته باشیم.

کد تابع تولید کننده ی دیتا ست به صورت زیر است که فقط با یکبار اجرا شدن دیتا ست را در یک متغیر ذخیره می کند.

```
function GenerateMapset()

Mapset = cell(2,32);
a_code = double('a');
for i = 0:25
    Mapset{1,i+1} = char(i+a_code);
    Mapset{2,i+1} = dec2bin(i,5);
end

Mapset{1,27} = ' ';
Mapset{2,27} = dec2bin(26,5);
Mapset{1,28} = '.';
Mapset{2,28} = dec2bin(27,5);
Mapset{1,29} = ',';
Mapset{2,29} = dec2bin(28,5);
Mapset{1,30} = '!';
Mapset{2,30} = dec2bin(29,5);
Mapset{1,31} = '"';
Mapset{2,31} = dec2bin(30,5);
Mapset{1,32} = ';';
Mapset{2,32} = dec2bin(31,5);

save('MAPSET.mat',"Mapset");
end
```

بخش دوم

برای این تابع ابتدا با یک تابع مجزا پیام متنی را تبدیل به پیام باینری می کنیم به این صورت که هر حرف را در دیتاست پیدا کرده و معادل باینری آن را قرار می دهیم. کد این تابع به شکل زیر است:

```
function binMsg = msgToBin(msg)
    GenerateMapset();
    load MAPSET.mat Mapset;
    msg = char(msg);
    if msg(1,end) ~= ';'
        msg = [msg ';'];
    end
    binMsg = [];
    for c = msg
        tmp = [Mapset{1,:}]==c;
        binMsg = [binMsg Mapset{2,tmp}];
    end
end
```

سپس تابع encode را می نویسیم این تابع پیام باینری را دریافت کرده و آن را به قطعه های باینری به طول نرخ داده تقسیم می کند سپس بر اساس نرخ داده مخرج کسر دامنه را محاسبه می کند (برای مثال برای چهار بیت مخرج کسر ۱۵ است) سپس هر قطعه را به عدد دسیمال متناظر تبدیل کرده و آن را بر روی مخرج کسر محاسبه شده قرار می دهد تا ضریب دامنه ی هر قطعه باینری محاسبه شود و در نهایت این ضریب را در سیگنال سینوسی ضرب کرده و به سیگنال نهایی اضافه می کند کد این تابع به شکل زیر است:

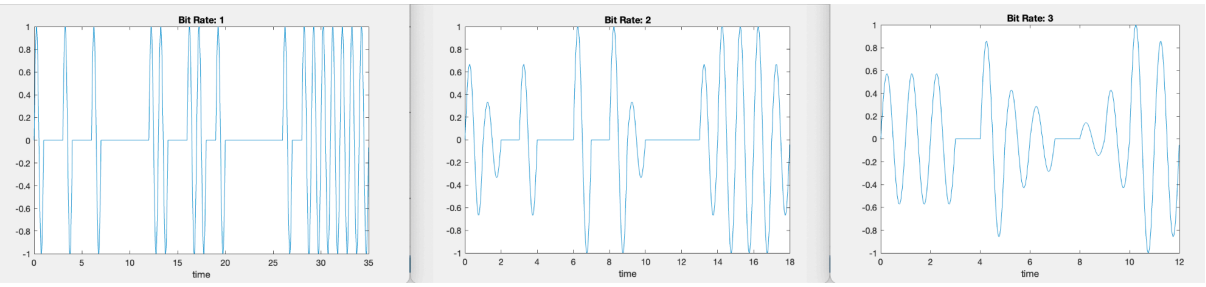
```
function signal = coding_amp(msg, bit_rate, freq)

freq = double(freq);
T = 1/freq;
t = 0:T:1;
t = t(1:end-1);
template = sin(2*pi*t);
signal = [];

msg = msgToBin(msg);
msg = [msg repmat('0',1,mod(bit_rate-mod(length(msg),bit_rate),bit_rate))];
msg = reshape(msg,bit_rate,length(msg)/bit_rate)';
coefficient = 1/(2^bit_rate-1);

for c = 1:size(msg,1)
    signal = [signal double(bin2dec(msg(c,:)))*coefficient*template];
end

end
```

بخش سوم

حال تابع را با کد زیر تست می کنیم:

```
clear;close;clc;

msg = 'signal';
frequency = double(100);

out_1bit = coding_amp(msg,1,100);
out_2bit = coding_amp(msg,2,100);
out_3bit = coding_amp(msg,3,100);

figure
time1 = 0:length(out_1bit)-1;
time1 = (1/frequency) * time1;
plot(time1,out_1bit)
title("Bit Rate: 1")
xlabel("time")


figure
time2 = 0:length(out_2bit)-1;
time2 = (1/frequency) * time2;
plot(time2,out_2bit)
title("Bit Rate: 2")
xlabel("time")


figure
time3 = 0:length(out_3bit)-1;
time3 = (1/frequency) * time3;
plot(time3,out_3bit)
title("Bit Rate: 3")
xlabel("time")
```

نتیجه به صورت زیر است:



بخش چهارم

برای نوشتن تابع خواسته شده سیگنال دریافتی را به سیکل های جدا گانه تقسیم می کنیم حال هر سیکل را با سیگنال سینوسی corrolate می کنیم، همچنین تابع corrolate به همان نحوی پیاده سازی شد که در صورت پروژه خواسته شده بود بنابراین خروجی آن یک عدد صحیح است که معادل دسیمال، عدد باینری کد می باشد. و در نهایت اعداد دسیمال تبدیل به اعداد باینری می شوند و توسط تابعی عکس تابع تبدیل کننده ی پیام به باینری ، کد باینری را به پیام تبدیل می کنیم:

```
function message = decoding_amp(signal,bit_rate,frequency)

    frequency = double(frequency);
    T = 1/frequency;
    t = 0:T:1;
    t = t(1:end-1);
    template = sin(2*pi*t);

    signal = reshape(signal,frequency,length(signal)/frequency)';
    bincode = [];
    for i = 1:size(signal,1)
        bincode = [bincode dec2bin(round(correlate(template,signal(i,:), frequency,bit_rate)),bit_rate)];
    end
    message = BinToMessage(bincode);

end
```

سپس این تابع و تابع قبلی را با کد زیر آزمایش می کنیم:

```
clear;close;clc;

msg = 'signal';

out_1bit = coding_amp(msg,1,100);
out_2bit = coding_amp(msg,2,100);
out_3bit = coding_amp(msg,3,100);


decoding_amp(out_1bit,1,100)
decoding_amp(out_2bit,2,100)
decoding_amp(out_3bit,3,100)
```

نتیجه به صورت زیر است:

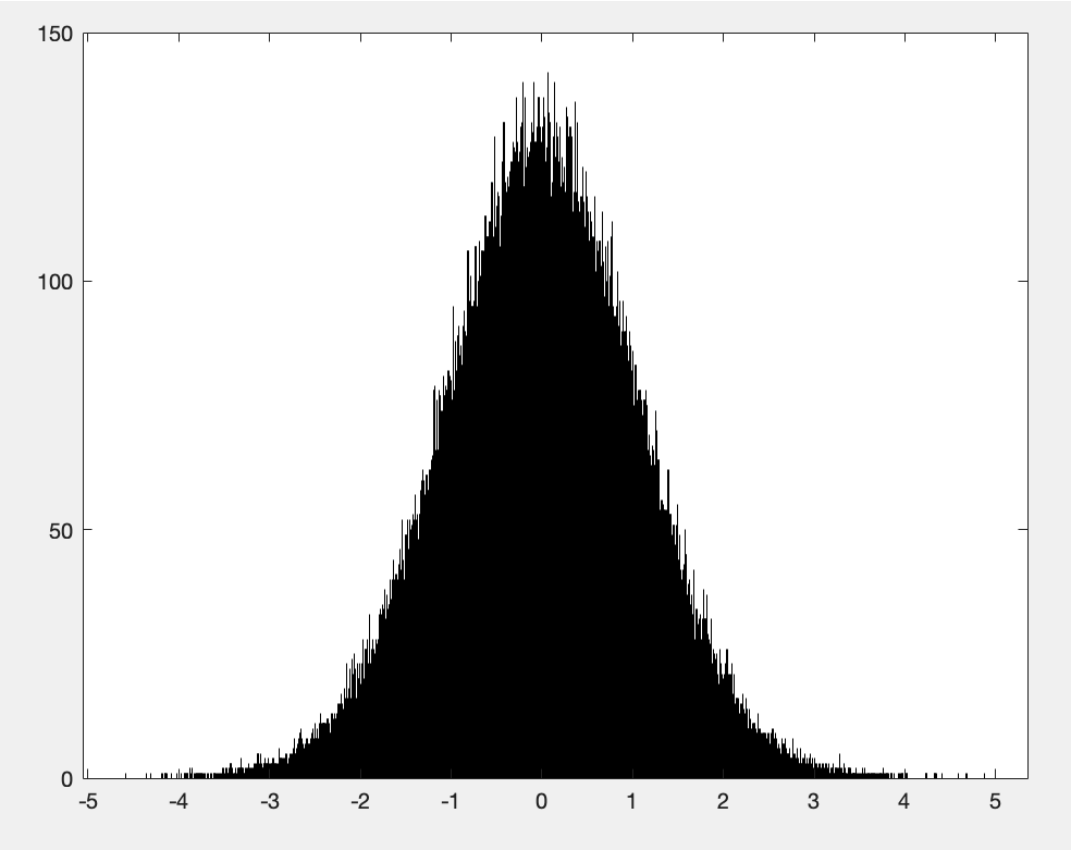
```
ans =

    'signal'


ans =

    'signal'


ans =

    'signal'
```

بخش پنجم

برای نشان دادن موارد خواسته شده باید هیستوگرام تابع randn را رسم کنیم و همچنین می توانیم از توابع آماده متلب برای میانگین گیری و واریانس گیری استفاده کنیم:

```
close;clear;clc;

x_n = randn([1,300000]);
histogram(x_n,10000);
mn = mean(x_n)
vr = var(x_n)
```

هیستوگرام و نتایج خواسته شده به صورت زیر است:



```
mn =

    3.3671e-04


vr =

    1.0031
```

همانطور که مشاهده می شود خروجی این تابع یک نویز گاوسی با میانگین صفر و واریانس یک است.

بخش ششم

برای آزمایش مورد خواسته شده یک سیگنال نویزی با قدرت خواسته شده (۰.۰۱) تولید می کنیم و به تابع decode می دهیم:

```
clear;close;clc;

message = cell(2,3);
noise_str = 0.01;
for bit_rate = 1:3
    signal = coding_amp('signal',bit_rate,100);
    signal = signal + noise_str*randn(size(signal));
    message{1,bit_rate} = ['Bit Rate: ' char(double('0')+bit_rate)];
    message{2,bit_rate} = decoding_amp(signal,bit_rate,100);
end

message
```

نتیجه به صورت زیر است

```
message =

  2×3 cell array

    {'Bit Rate: 1'}    {'Bit Rate: 2'}    {'Bit Rate: 3'}
    {'signal'     }    {'signal'     }    {'signal'     }
```

همانطور که مشاهده می شود با این میزان نویز برای هیچ کدام خطایی ایجاد نمی شود.

بخش هفتم

قدرت نویز را هر بار به اندازه ۰.۱ افزایش می دهیم

- اولین بار در قدرت نویز ۰.۲ در مقدار خروجی با نرخ بیت ۳ خطا ایجاد می شود:

```
{'Bit Rate: 1'}    {'Bit Rate: 2'}    {'Bit Rate: 3'}
{'signal'     }    {'signal'     }    {'signam'     }
```

- در قدرت ۰.۴ برای اولین بار در مقدار خروجی با نرخ بیت ۲ خطا ایجاد می شود:

```
  {'Bit Rate: 1'}    {'Bit Rate: 2'}    {'Bit Rate: 3'}
  {'signal'     }    {'signb;!'    }    {'sioqqt'     }
```

- در قدرت ۱.۲ برای اولین بار در مقدار خروجی با نرخ بیت ۱ خطا ایجاد می شود:

```
{'Bit Rate: 1'}    {'Bit Rate: 2'}    {'Bit Rate: 3'}
{'si;!tic'    }    {'smgnal'     }    {'wh;ug;"f'   }
```

بنابراین نرخ بیت ۱ از همه مقاوم تر بود که با مطالب بیان شده در صورت پروژه تطابق دارد.

بخش هشتم

همانطور که در بخش قبل بیان شد:

- نرخ بیت سه: ۰.۰۴
- نرخ بیت دو: ۰.۱۶
- نرخ بیت یک: ۱.۴۴

بخش نهم

همانطور که در صورت پروژه بیان شد در صورتی که میزان دامنه افزایش یابد فاصله ی بین treshold های تعیین شده در تابع decode و corrolate افزایش می یافت در نتیجه تاثیر نویز در نتیجه رمزگشایی کاهش می یافت. بنابراین برای حل این مشکل باید دامنه ی سیگنال سینوسی را افزایش دهیم که این کار با افزایش قدرت فرستنده ممکن است.

بخش دهم

اگر هیچ گونه نویزی در سیگنال وجود نداشته باشد می توان نرخ بیت را تا جایی افزایش داده که فاصله ی بین treshold های کد های متفاوت برای دستگاه رمز گشا قابل تشخیص باشد برای مثال برای داده با نرخ بیت ۶ بیت حداقل دقت مورد نیاز ۱/۶۳ است.

بخش یازدهم

خیر نسبت به نویز مقاوم تر نخواهد شد زیرا در هنگام corrolate مقدار ۱۰ به جای ۲ بر روی نویز هم تاثیر خواهد گذاشت و دامنه ی آن را نیز افزایش خواهد داد بنابراین تاثیری بر مقاومت در برابر نویز ندارد و تنها افزایش دامنه ی فرستنده می تواند تاثیر نویز را خنثی کند.

بخش دوازدهم

معمولا اینترنت خانگی اطلاعات را با سرعتی بین ۸ تا ۱۶ مگابیت بر ثانیه ارسال می کند اما ما در اینجا در بهترین حالت با سرعت ۳ بیت بر ثانیه اطلاعات را ارسال کردیم !