به نام خدا

دانشگاه تهران

پردیس دانشکده فنی

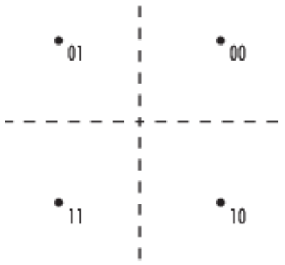
دانشکده برق و کامپیوتر

گزارش تمرین کامپیوتری 1

ریحانه گلی

810196545

هدف از انجام این تمرین کامپیوتری، شبیه سازی و پیاده سازی یک سیستم وایرلس ساده می باشد که به این منظور در ابتدا با استفاده از فایل generate_data.py رشته های ورودی که رشته ای ا ز 0 و 1 ها می باشد تولید شده و در فایل input.txt ریخته می شوند (10000 رشته دو بیتی تولید شده و در این فایل قرار گرفته است) .در مرحله اول فرض می شود که از مدل مدلاسیون QPSK استفاده می شود در نتیجه مانند شکل زیر رشته های "00" و "01"و"10"و"11" به نقاطی با مختصات مشخص در صفحه مختصات map می شوند. به منظور پیاده سازی این کار کلاسی با نام transmitter در نظر گرفته شده است



که در آن عملیات map صورت می گیرد.

برای پیاده سازی کانال وایرلس صرفا اثر محوشدگی را درنظر می گیریم و فرض می شود که اگر ورودی کانال عدد مختلط x باشد ابتدا این عدد در gain کانال که با متغیر h نمایش داده می شود و عددی مختلط است (hI,hQ) ضرب می شود. هم چنین می دانیم که توزیع hI و hQ به صورت نرمال با میانگین 0 و واریانس 1 می باشد و سپس حاصل آن با نویز AWGN که با متغیر n نمایش داده می شود و دارای توزیع نرمال با میاگین 0 و واریانس δ^2 می باشد جمع می شود و ارسال می شود. برای پیاده سازی کانال وایرلس که وظایف گفته شده را برعهده دارد کلاسی با نام wireless_channel تعریف شده است. و با استفاده از متد های applyChannelGain() و applyAWGN() این کار صورت می گیرد.

سیگنالی که در گیرنده دریافت می شود به صورت y=hx+n می باشد و برای به دست آوردن x می بایست اثر کانال را با تقسیم y بر h خنثی کنیم. به این منظور کلاسی با نام receiver در نظر گرفته شده که با استفاده از تابع removeChannelImpact() می توان این کار را انجام داد.

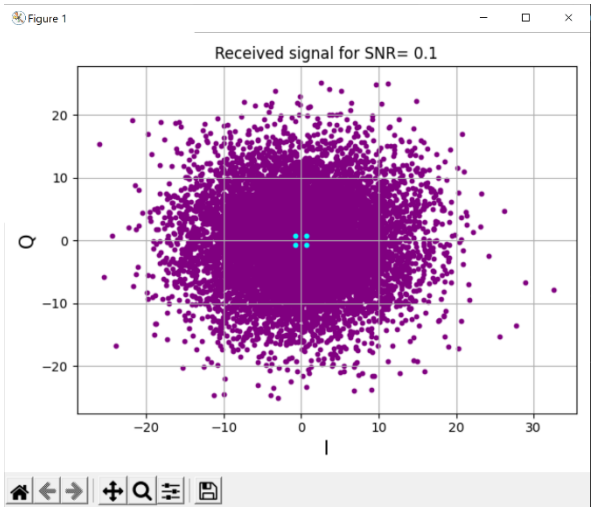
پس از خنثی کردن اثر کانال نقطه ای با مختصاتی داریم که برای بازیابی داده آن می بایست از Maximum Likelihood استفاده کنیم که قاعده ی ML در این جا همان یافتن نزدیک ترین constellation point به نقطه مورد نظر ماست. در این قسمت از آن جایی که از مدلاسیون QPSK استفاده می شود با توجه به مثبت یا منفی بودن هر یک از قسمت های حقیقی و موهومی نقطه به دست آمده می توان آن را بازیابی کرد. به این منظور در کلاس receiver متدی با نام demodulate() طراحی شده است.

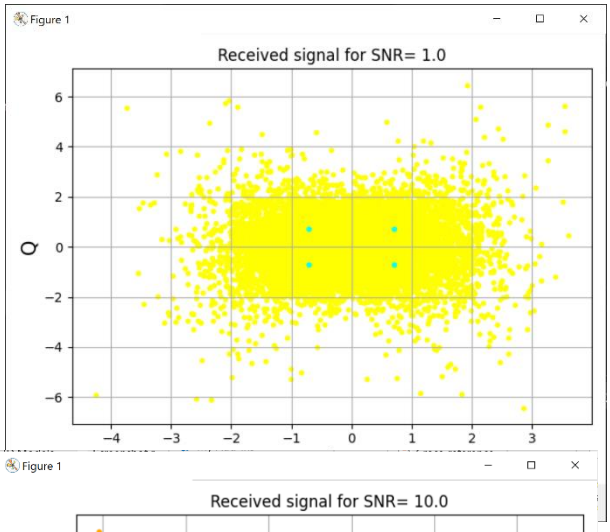
هم چنین فرض شده است که SNR=1/δ^2

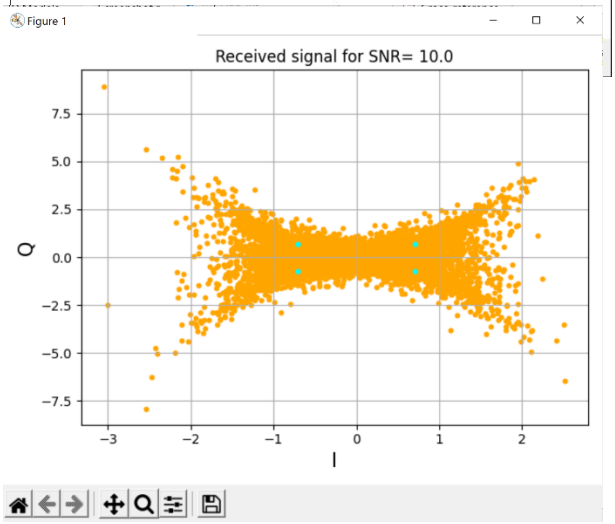
در بخش اول این تمرین scatter plot هایی برای SNR=0.1 , SNR=1 , SNR=10 رسم شده است که این کار با فراخوانی متد runForScatterPlot() که در کلاس wireless_system قرار دارد امکان پذیر است.

در این متد به ازای هر رشته دو بیتی که در فایل input.txt قرار دارد با استفاده از متد ()modulate به constellation point ای مپ شده است و این point به ترتیب به متد های ()applyChannelGain و ()applyAWGN داده شده و پس از آن ، نقطه حاصله در آرایه ای با نام ()generated_Points ، پوش می شود.

اشکال زیر نشان دهنده این scatter plot ها می باشند.

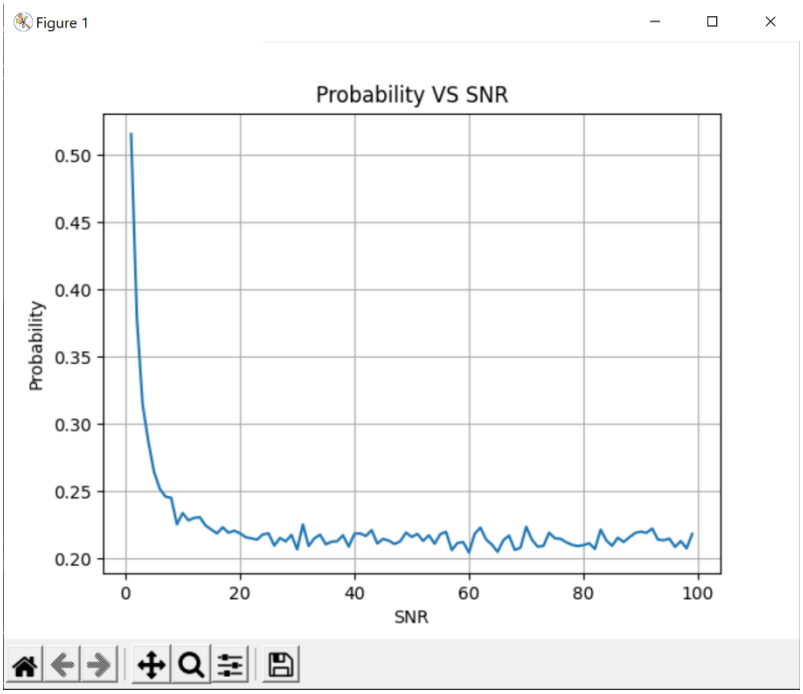






در بخش دوم این تمرین می خواهیم متوسط احتمال خطا را بر حسب SNR رسم کنیم.به این منظور از تابع runForLinePlot() استفاده شده است که در آن به ازای 100 مقدار متفاوت SNR ، در هر بار تکرار h ها و n های مختلف تولید شده و پس از از بین بردن اثر کانال با فراخوانی تابع removeChannelImpact() ، نقطه به دست آمده با استفاده از متد demodulate() به داده تبدیل می شود. سپس این داده بازیابی شده با داده اولیه مقایسه شده و در صورتی که با هم برابر باشد مقدار numOfCorrectOutputs یک واحد اضافه می شود. حالا با داشتن تعداد داده هایی که به درستی بازیابی شدند و تقسیم آن ها بر تعداد کل داده ها می توان احتمال عدم خطا را به ازای هر SNR محاسبه کرد. با داشتن این احتمال و کم کردن آن از 1 می توان به احتمال خطا رسید.

در شکل زیر می توان نمودار متوسط احتمال خطا را بر حسب SNR را مشاهده کرد



در بخش سوم این تمرین فرض شده که از کدینگ کانال Hamming(4,7) در فرستنده و گیرنده استفاده شده است. و می خواهیم مانند قسمت قبل نمودار احتمال خطا بر حسب SNR را برای آن رسم کنیم.

در Hamming(4,7) در کلاس transmitter ماتریس g که ماتریسی 7*4 می باشد داریم که برای تولید d1,d2,d3,d4,p1,p2,p3 مورد استفاده قرار می گیرد. در متد encodeAllWithHamming() ابتدا داده ها به صورت 4 تایی خوانده شده و به متد encodeHamming() داده می شوند این متد با ضرب داده ها که ماترسی 4*1 اند در ماتریس g به ازای هر داده 4 تایی یک داده 7 تایی تولید می کند که این داده های تولیدی به صورت پشت سر هم در فایلی با نام encoded.txt ریخته می شوند.

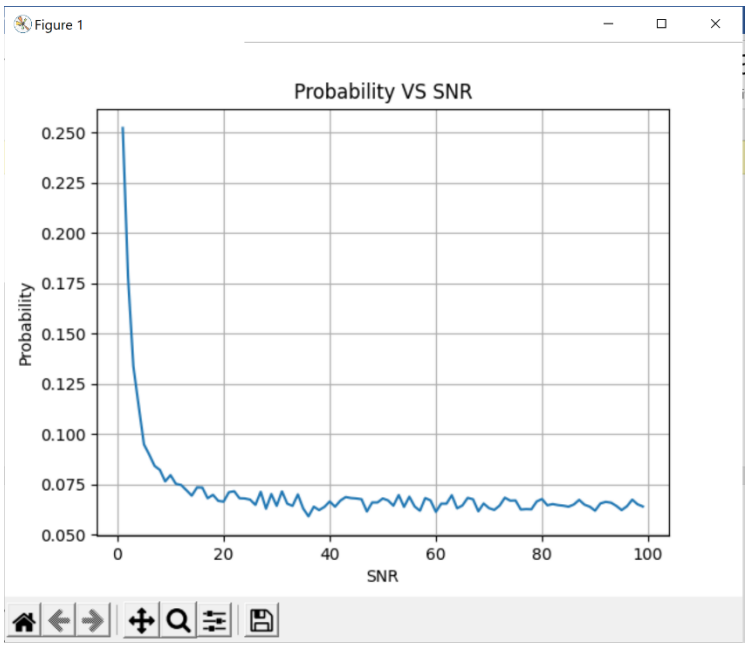
در تابع runWithHammingCode() از فایل encoded.txt دو تا دو تا خوانده و با رد کردن آن ها از کانال(اعمال gain کانال و جمع آن با نویز) و خنثی کردن اثر کانال در گیرنده داده ها را بازیابی کرده و پشت سر هم در فایلی با نام demodulated.txt می ریزیم.

با فراخوانی تابع decodeAll() داده های موجود در فایل demodulated.txt را 7 تا 7تا خوانده و به تابع decodeHamming() می دهیم. در کلاس receiver ماتریسی با نام h داریم که ماتریسی 7*3 می باشد. این تابع با ضرب داده ما که به صورت ماتریسی 1*7 در آمده است در ماتریس h ماتریسی 1*3 تولید می کند که نشان دهنده parity های ماست. از آن جایی که مس دانیم به ازای هر 7 تایی که از demodulated.txt خوانده می شود ، 4 بیت اول آن داده واقعی می باشد، در هر مرحله داده واقعی(4 بیت) را در کنار parity تولید شده قرار داده (3 بیت) و در فایلی با نام decoded.txt پشت سر هم می نویسیم.

در تابع reconstructAndCalcCorrectOutputs() داده های فایل decoded.txt را 7 تا 7 تا خوانده و می دانیم 4 بیت اول آن ها داده واقعی و 3 بیت بعدی آن ها parity است هر یک از این هفت بیت را به تابع findAndCorrectError() می دهیم. در این تابع با توجه به سه بیت parity مشخص می شود که ارور در داده دریافتی در کدام بیت بوده است. و اگر اروری در d1 یا d2 یا d3 یا d4 رخ داده باشد با flip کردن آن می توان خطا را برطرف کرد.

پس از رفع خطا توسط این تابع و مقایسه آن با داده ای که اول فرستاده شده بود(داده های در فایل input.txt) می توان تعداد بیت های صحیح را محاسبه کرد و از روی مکمل آن احتمال خطا را به دست آورد.

نمودار احتمال خطا بر حسب SNR در این قسمت:



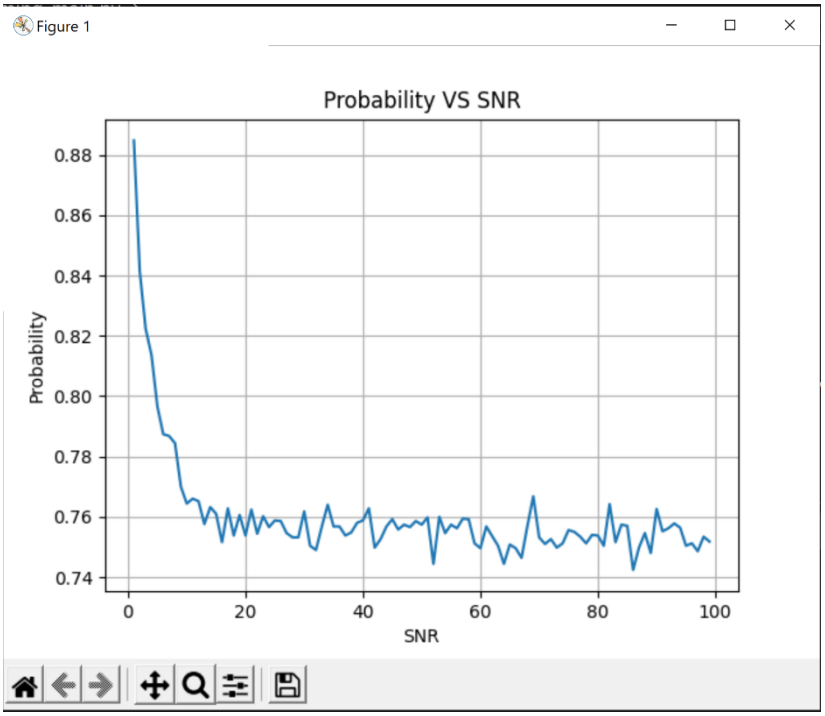
در بخش 4 این تمرین خواسته شده است که به جای مدلاسیون QPSK از 16QAM استفاده کنیم و احتمال خطا برحسب SNR را در حالت عادی و در حالتی که از کدینگ Hamming(4,7) در فرستنده و گیرنده استفاده شده است رسم کنیم.

مراحل انجام کار دقیقا مانند دو قسمت قبل می باشد با این تفاوت که برای مورد اول از فراخوانی تابع runForLinePlot16() استفاده شده است که از نظر پیاده سازی با runForLinePlot() یکسان است و تفاوت آن در فراخوانی توابع modulate16QAM() و demodulate16() می باشد.

در modulate16QAM() با توجه به qam16Dict ای که در کلاس transmitter تعریف شده ، رشته های 4 تایی را به constellation point هایی map می کند . در demodulate16() با توجه به regionpoints ها که در receiver تعریف شده اند (که نشان دهنده ناحیه 1 و 4 نقطه مختص آن و ناحیه 2 و 4 نقطه مختص آن و...می باشد) و مثبت یا منفی بودن بخش های real و موهومی نقطه مورد نظر، ابتدا ناحیه ای که نقطه به آن تعلق دارد حدس زده می شود و از بین 4 نقطه آن ناحیه، نقطه ای انتخاب می شود که نزدیک ترین به آن باشد(با پیاده سازی findNearest()) و اینگونه داده بازیابی می شود.

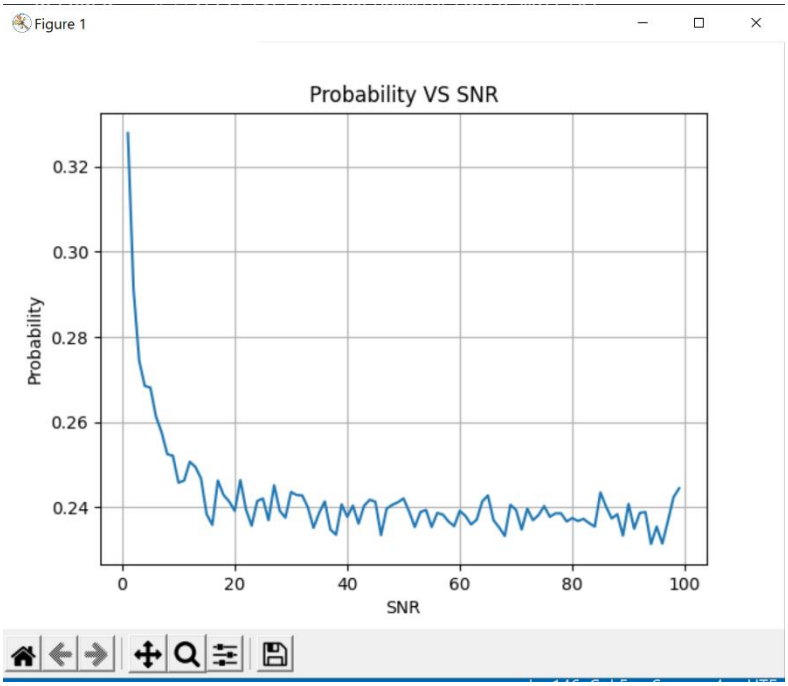
و با مقایسه با داده ارسالی و در ازای درست بودن آن numOfCorrectOutputs یک واحد اضافه می شود. و با داشتن numOfCorrectOutputs و تعداد کل داده ها می توان این احتمال خطا را به ازای هر SNR به دست آورد.

این نمودار به صورت زیر می باشد:



برای بخش Hamming نیز دقیقا مانند بخش سوم عمل شده با این تفاوت که در توابع لازم برای آن هر جا لازم بوده از توابع modulate16QAM() و demodulate16() به جای modulate() و demodulate() استفاده شده است.

که نتیجه به صورت زیر می باشد:



نتیجه گیری کلی: استفاده از کدینگ Hamming احتمال خطا را کاهش میدهد و هم چنین احتمال خطا در QPSK در حالت کلی کم تر از 16QAM است چرا که constellation point ها از هم فاصله زیادی دارند و در هنگام بازیابی داده ها با خطای کم تری رو به رو هستیم.