



دانشگاه صنعتی شریف  
دانشکده مهندسی برق

عنوان:

## پروژه درس

اعضای گروه

سید محمد اجاق کزازی ۵۵۴۱۰۰۱۰۰۴۰۰

نام درس

سیستم‌های مخابراتی

نیمسال اول ۱۴۰۲-۱۴۰۳

نام استاد درس

محمد رضا پاکروان

# ۱ مقدمه

در این قسمت دیاگرام فرستنده و گیرنده ترسیم شده که برای فهم بهتر مساله نیازمند بررسی بوده و از آن استفاده کردیم!

## ۲ پیاده سازی بلوک ها به صورت مجزا

۱-۲

دوتابع Divide و Combine را مطابق خواسته مساله پیاده سازی کردیم. تنها نکته مهم این است که برای نزدیک شدن به حالت Real-Time، درایه های فرد  $b_1$  و درایه های زوج را در  $b_2$  قرار دادیم. اینگونه میتوان فرض کرد که دو پردازش همزمان و موازی هستند و سرعت ما دوبارابر شده و از طرفی همزمانی نیز رعایت میشود. در تابع Combine نیز همین منطق را در پیش گرفتیم.

۲-۲

تابع PulseShaping را مطابق خواسته سوال طراحی کردیم. به این صورت که به ازای هر بیت ۱ در آرایه باینری، یک شکل موج مختص به بیت برابر با ۱ و به ازای هر بیت ۰ در آرایه باینری، یک شکل موج مختص به بیت برابر با ۰ قرار دادیم. ترتیب این ها را نیز رعایت کردیم.

۳-۲

تابع AnalogMod را مطابق خواسته سوال طراحی کردیم. به این صورت که سیگنال ورودی را در توابع سینوس و کسینوس با فرکانس مرکزی  $f_c$  ضرب کردیم و پس از جمع با یکدیگر آن را خروجی دادیم. توابع سینوسی را با فرکانس  $f_s$  نمونه برداری کردیم.

۴-۲

برای پیاده سازی کانال، فیلتر هارا به صورت دستی خودمان تولید و استفاده کردیم! میدانیم که فیلتر حوزه زمان  $sinc(t)$  فیلتر پایین گذر در حوزه زمان است. از طرفی طبق قوانین فوریه ای میدانیم

بنابر این با ضرب دو نمایی در سیگنال  $sinc(t)$  میتوانیم  $g(t) \leftrightarrow G(f) \leftrightarrow g(t - t_0) \leftrightarrow e^{-j2\pi f t_0} G(f)$  فیلتر میانگذر دلخواه را تولید کنیم. ما در این تابع ابتدا سیگنالمان را به حوزه فرکانس بردیم و در فوریه‌ی  $sinc(t)$  که یک پالس مربعی است ضرب کردیم. سپس حاصل را به حوزه زمان برگرداندیم. با عبور سیگنال  $x_{transmit}$  از فیلتر ساخته شده سیگنال دریافت شده در دریافت کننده یعنی  $x_{receive}$  را تولید میکنیم.

۵-۲

کلیت این قسمت مانند تابع AnalogMod میباشد و لازم است که سیگنال را در یک سری سینوس و کسینوس ضرب کنیم و خروجی را تولید کنیم. مشابه همان توضیحات قسمت قبل برای تولید فیلتر پایین گذر از یک تابع  $sinc(t)$  تبدیل fft گرفته و سیگنال را از آن عبور میدهیم.

۶-۲

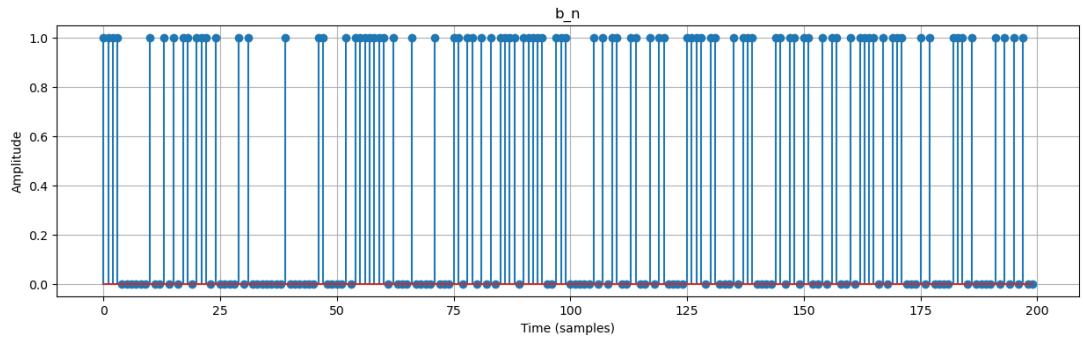
تابع اصلی و تصمیم گیرنده این تابع است! مطابق تعریف میدانیم که Filter Matched برای هر کدام از پالس‌های ما به صورت روبرو تعریف میشود:  $MatchedFilter = h(t) = pulse(T - t)$  بنابر این با flip کردن شکل پالس مختص به بیت ۱ و ۰، MatchedFilter های مربوطه تولید میشوند. پس از آن باید سیگنالی که در ورودی دریافت کرده ایم را با MatchedFilter ها کانوالو کنیم و هر  $T$  ثانیه نمونه برداری کنیم. حال برای تصمیم گیری که هر نمونه نشان دهنده بیت ۱ بوده یا ۰ نیز کافی است که خروجی MatchedFilter متناظر با ۱ را با خروجی متناظر با ۰ مقایسه کنیم و هر کدام بزرگتر بود، خروجی همان است! بهتر است به کدی که زده ایم نیز نگاهی بیندازید.

### ۳ انتقال دنباله تصافی ۰ و ۱

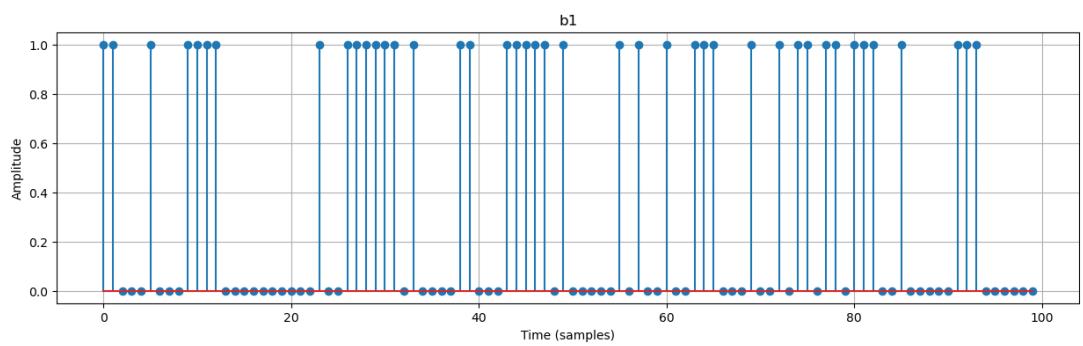
۱-۳

۱-۱-۳

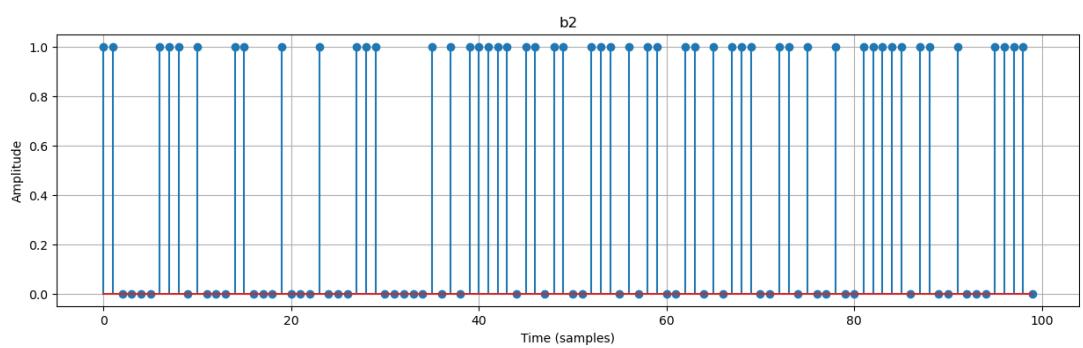
خروجی تابع‌های گفته شده در بخش ۲ را در اینجا نشان می‌دهیم. (به ازای یک ورودی ۲۰۰ عضوی از ۱۰۰)



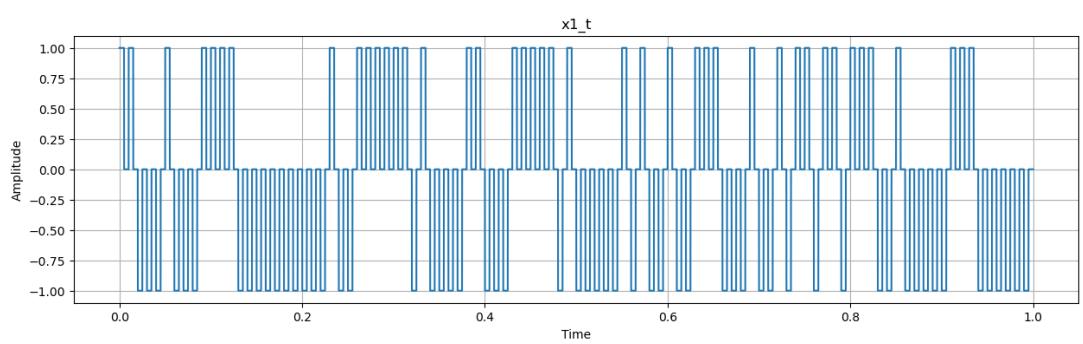
ورودی فرستنده  $b_n$



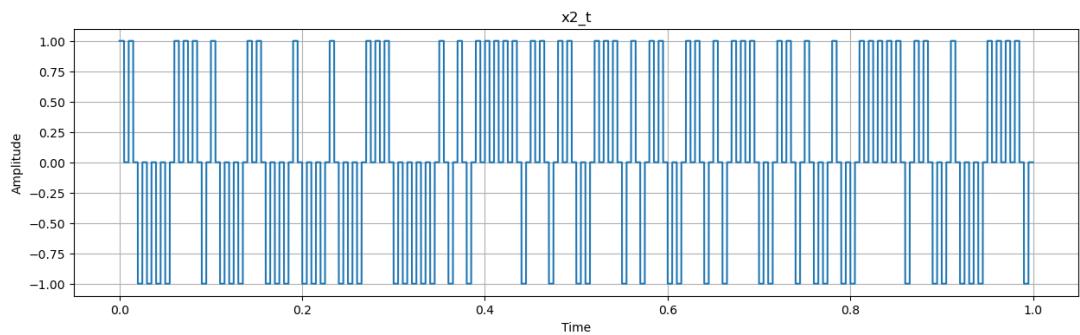
خروجی اول تابع Divide



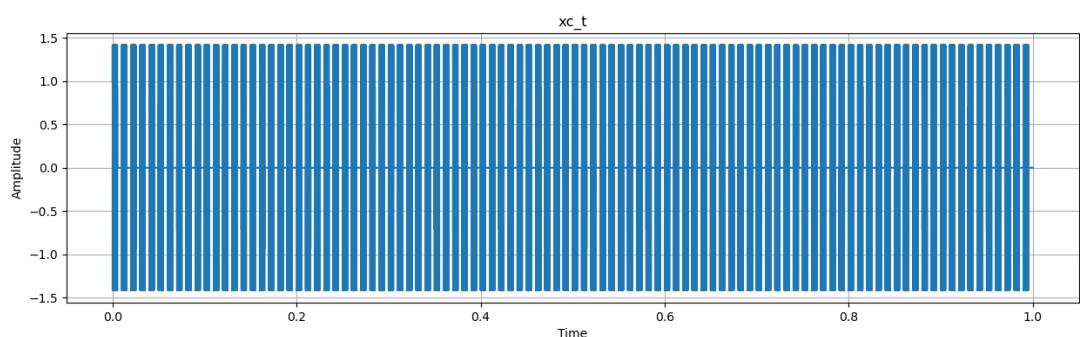
خروجی دوم تابع Divide



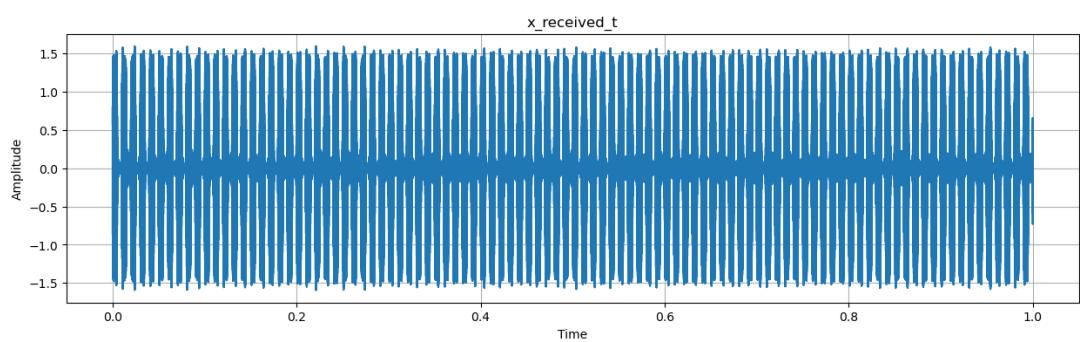
خروجی اول تابع PulseShaping



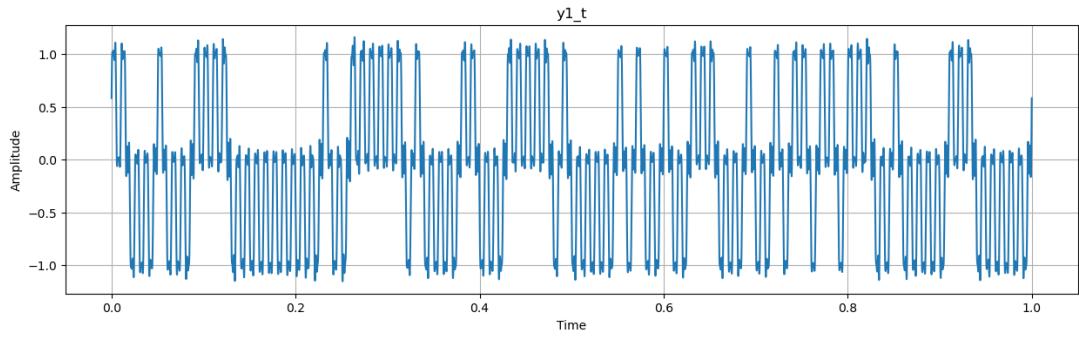
خروجی دوم تابع PulseShaping



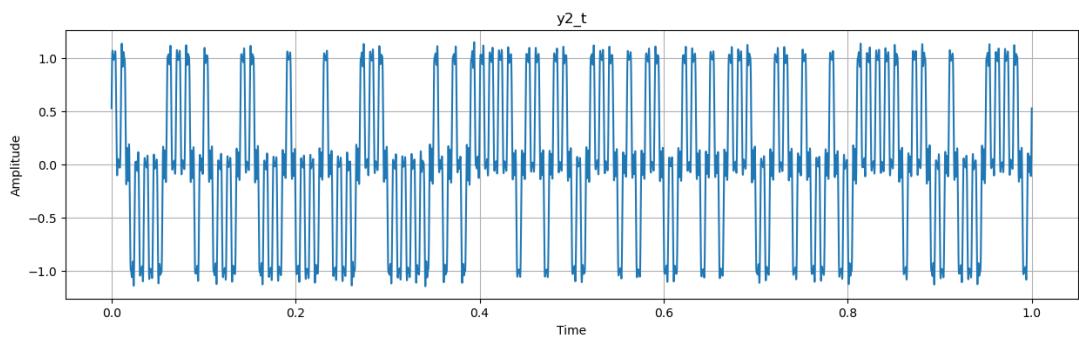
خروجی تابع AnalogMod



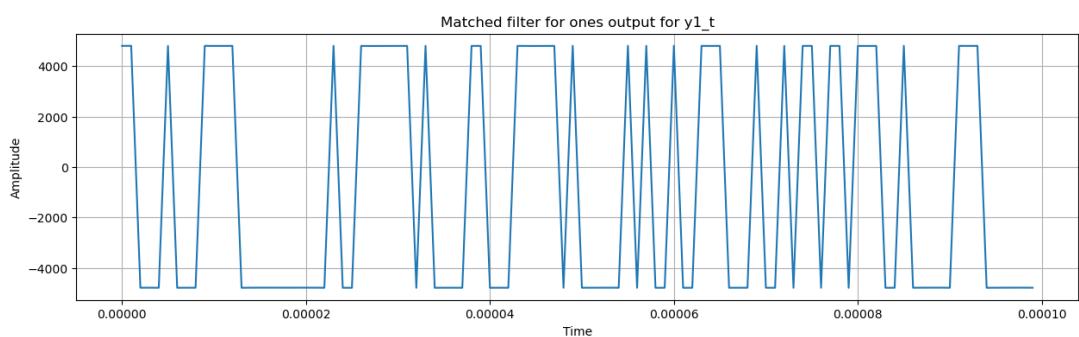
خروجی تابع Channel



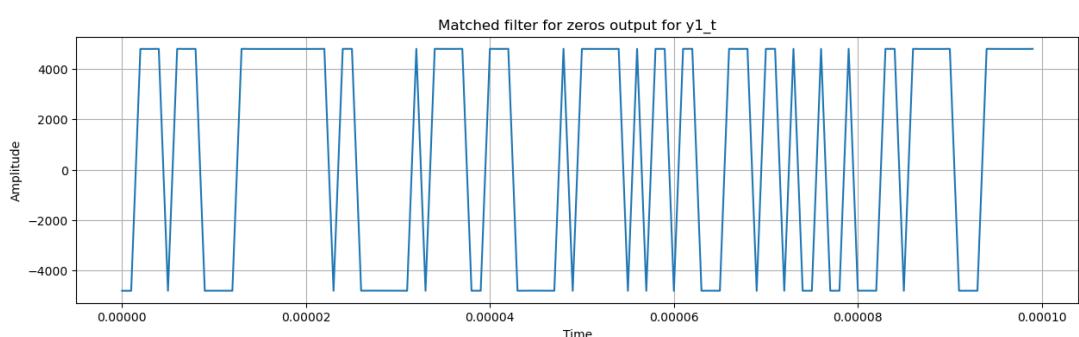
خروجی اول تابع AnalogDemod



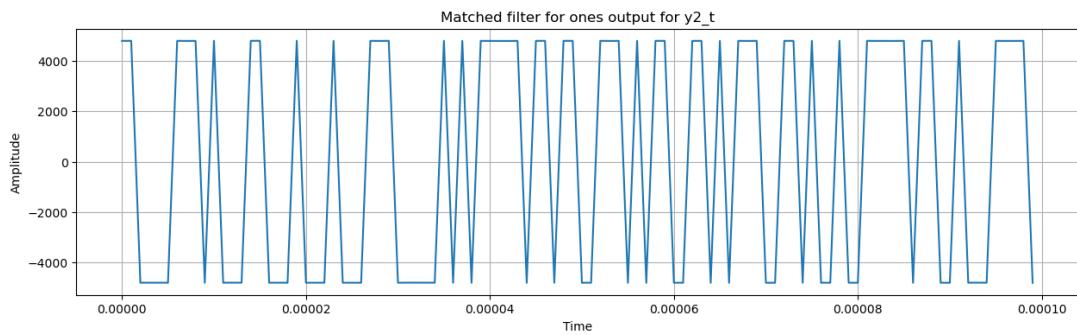
خروجی دوم تابع AnalogDemod



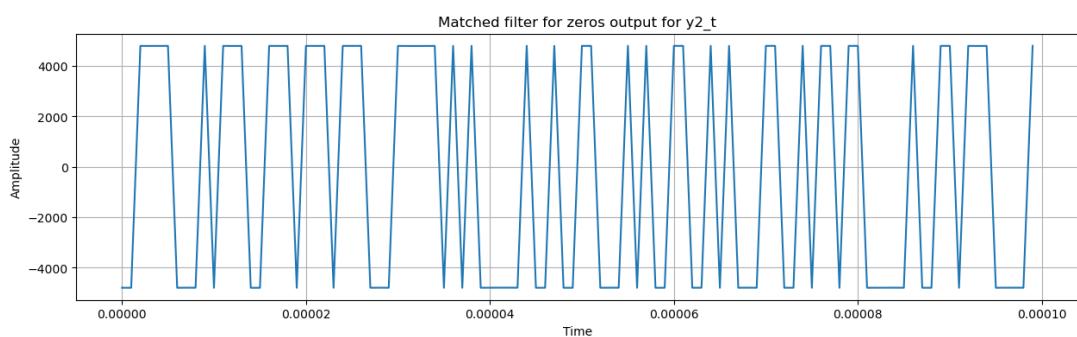
خروجی تابع MatchedFilt برای لول های یک رشته ای اول



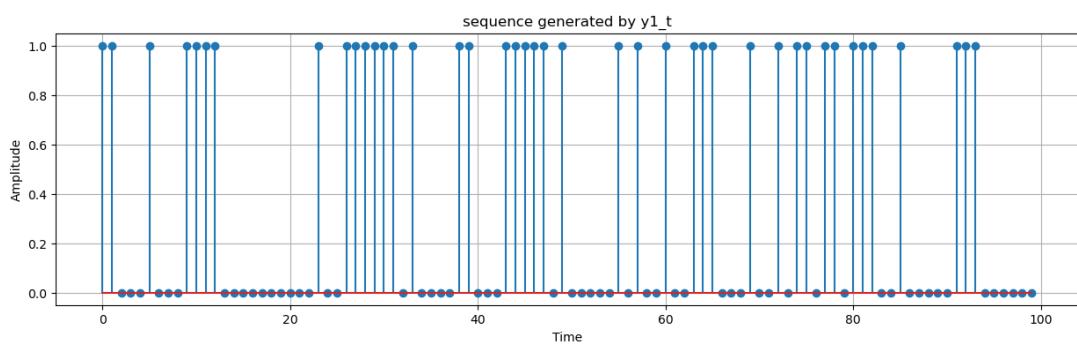
خروجی تابع MatchedFilt برای لول های صفر رشته ای اول



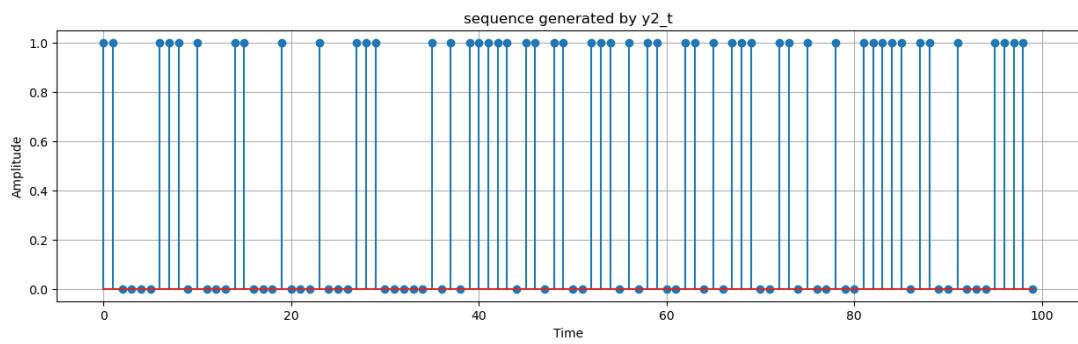
خروجی تابع MatchedFilt برای لول های یک رشته ای دوم



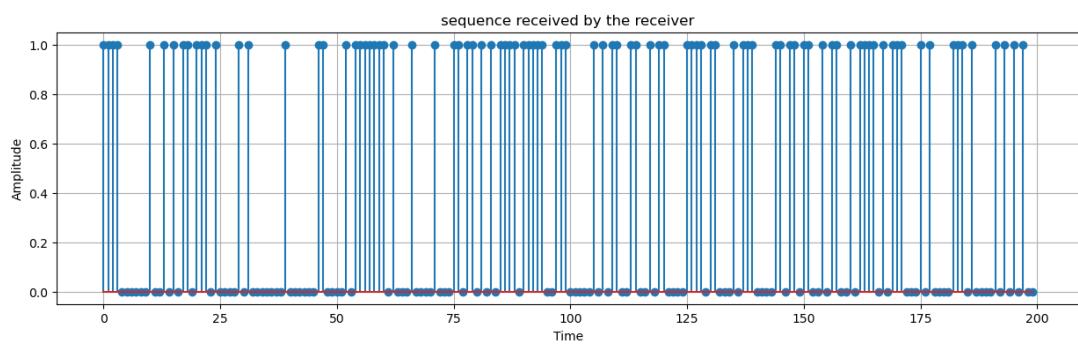
خروجی تابع MatchedFilt برای لول های صفر رشته ای دوم



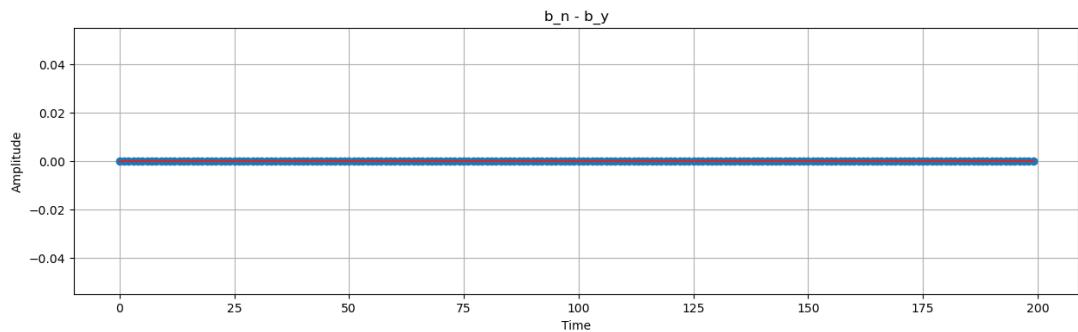
خروجی تابع MatchedFilt برای تخمین رشته اول



خروجی تابع MatchedFilt برای تخمین رشته دوم



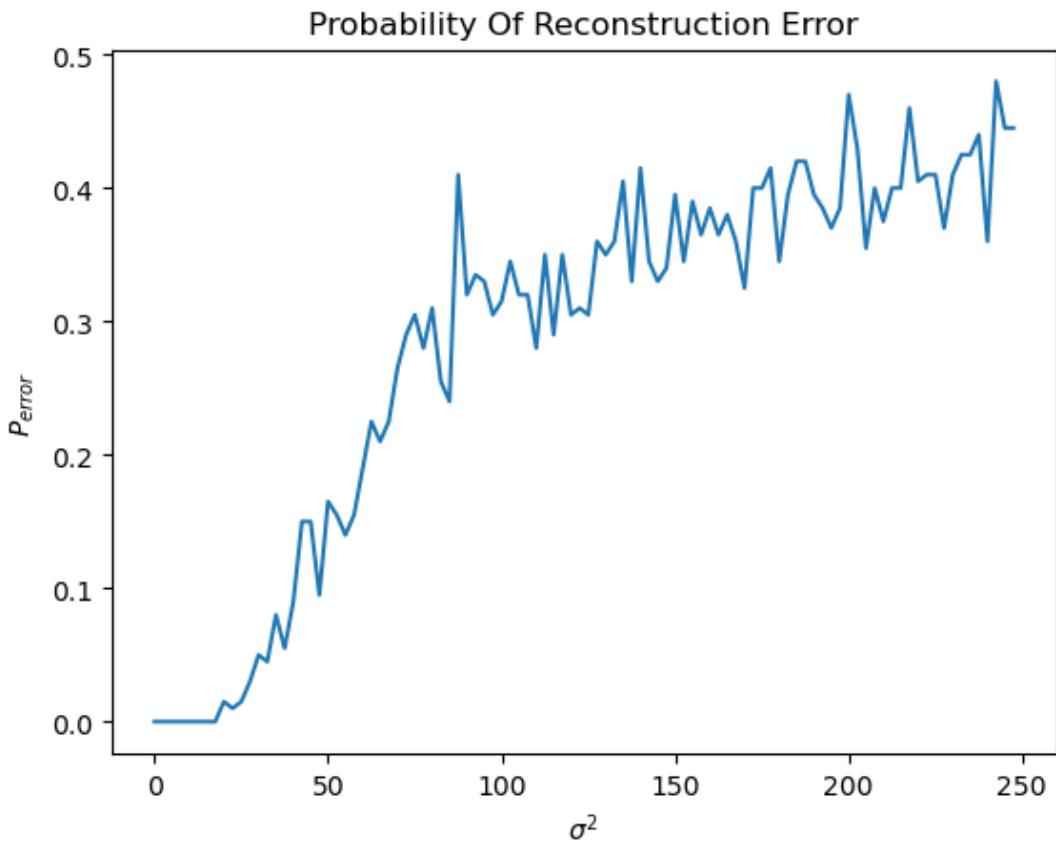
خروجی تابع Combine



اختلاف رشته‌ی خروجی و ورودی

۲-۱-۳

با استفاده از دستور `np.random.normal` در پایتون که متغیر رندوم با توزیع گوسی و میانگین و واریانس دلخواه ما تولید میکند، به سیگنال خروجی از تابع Channel نویز می افزاییم. نتیجه خواسته شده به شرح زیر میباشد:



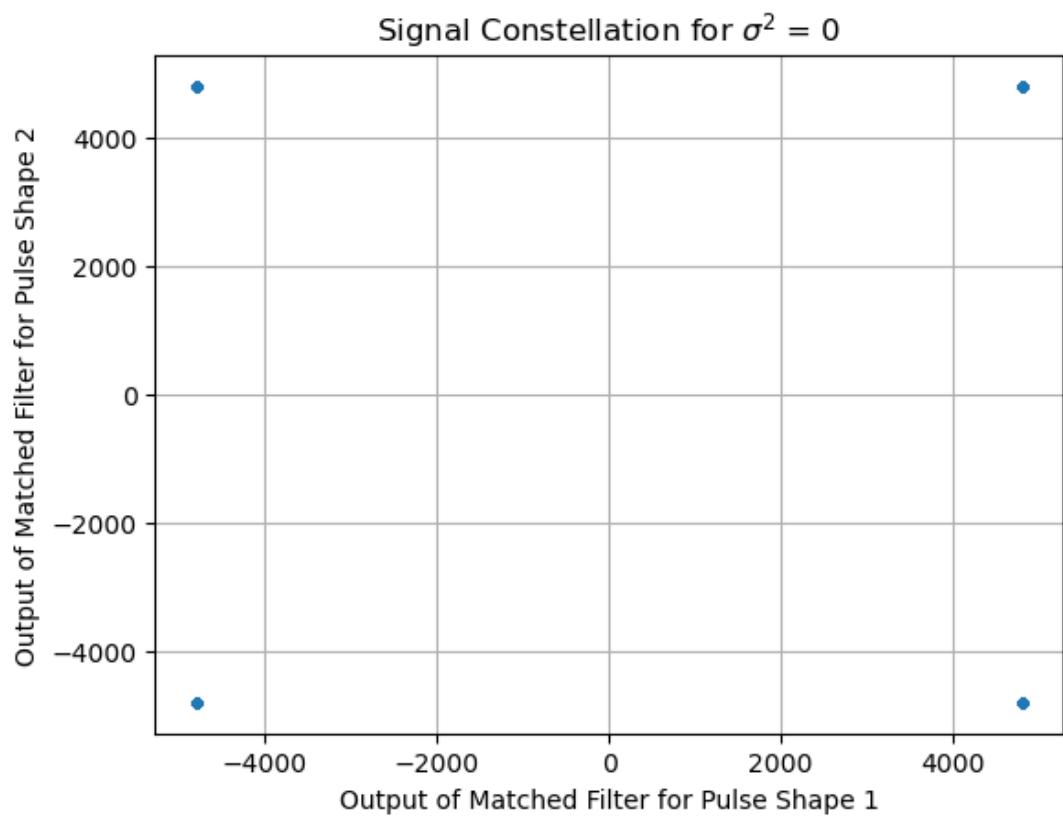
### احتمال خطای بازسازی بر حسب واریانس ویز کانال

واضح است که این رفتار کاملاً مطابق با انتظار ماست. همانطور که در تصویر هم واضح است خطای بازسازی به ازای واریانس های کم نویز صفر است، یعنی دریافت کننده می تواند سیگنالی که آلووده به نویز کوچکی است را با دقت ۱۰۰ درصد بازسازی کند. اما رفته رفته با افزایش شدت نویز و افزایش واریانس آن، خطای بازسازی افزایش میابد و مطابق انتظار مان، وقتی که واریانس نویز بیشتر و بیشتر میشود خطای ما به ۵۰ درصد میل میکند! چون گویی کلا اعداد رندوم تولید کرده که یا با بیت ارسال شده برابر هست یا نیست! بنابر این احتمال خطأ به ۵۰ درصد میل میکند. لازم به ذکر است رفتار کمی غیرخطی شکل فوق به این خاطر است که نمودار با ۱۰۰ نمونه ترسیم شده تا زمان زیادی برای پردازش صرف نشود. (با همین صد نمونه نیز ۱ دقیقه طول کشید برنامه تا اجرا بشود).

۳-۱-۳

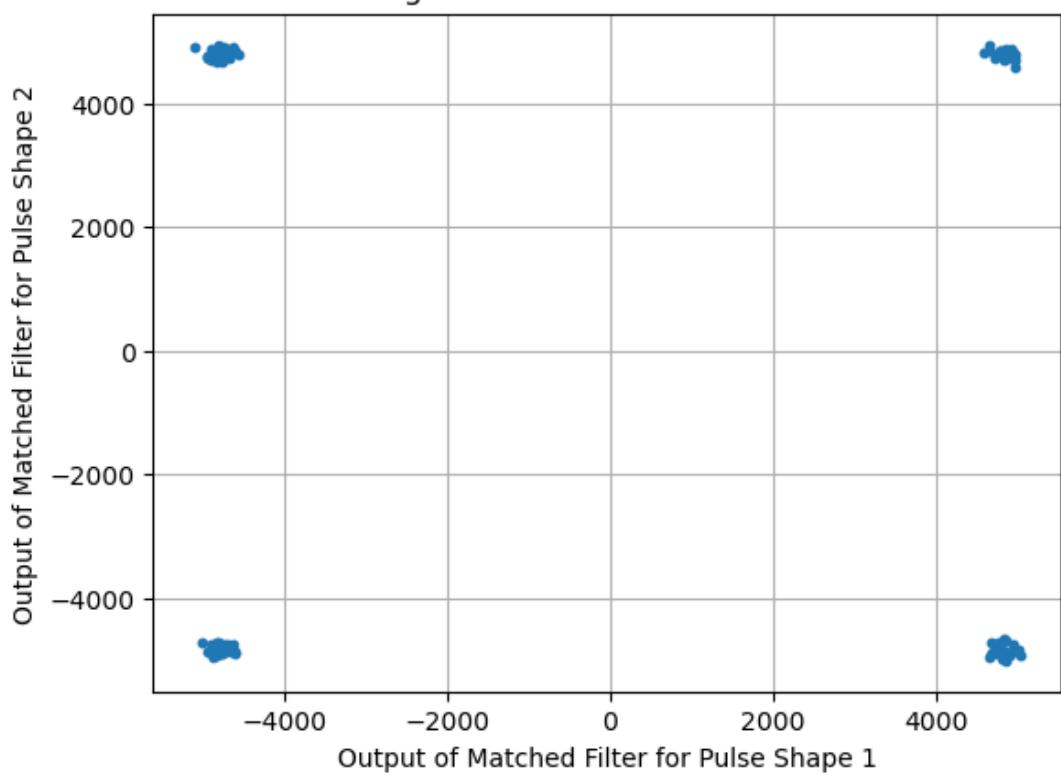
برای این قسمت از خروجی دوم تابع MatchedFilt استفاده کرده ایم. شکل های زیر Signal Constellation میباشند که بر اساس تعریفی که در متن پژوهه است رسم شده اند. این منظمه را

به ازای ۶ مقدار مختلف واریانس نویز رسم کرده ایم که نتیجه مانند زیر میباشد:



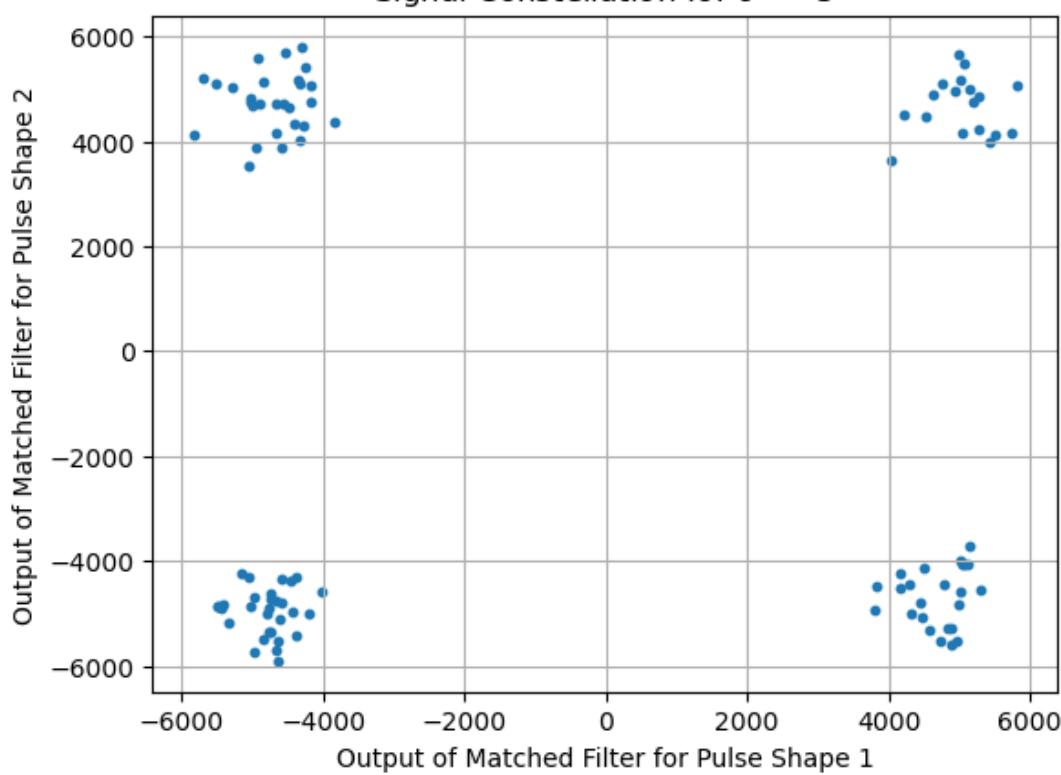
نویز با واریانس  $\sigma^2 = 0$

Signal Constellation for  $\sigma^2 = 1$

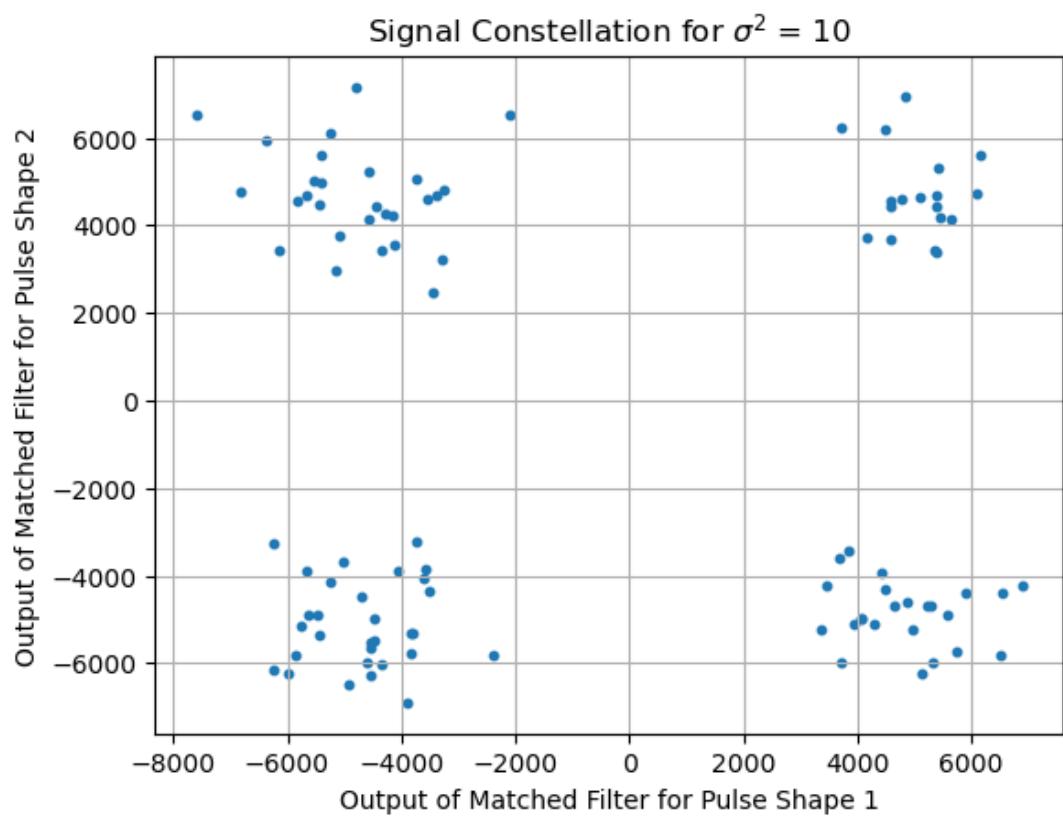


نویز با واریانس ۱

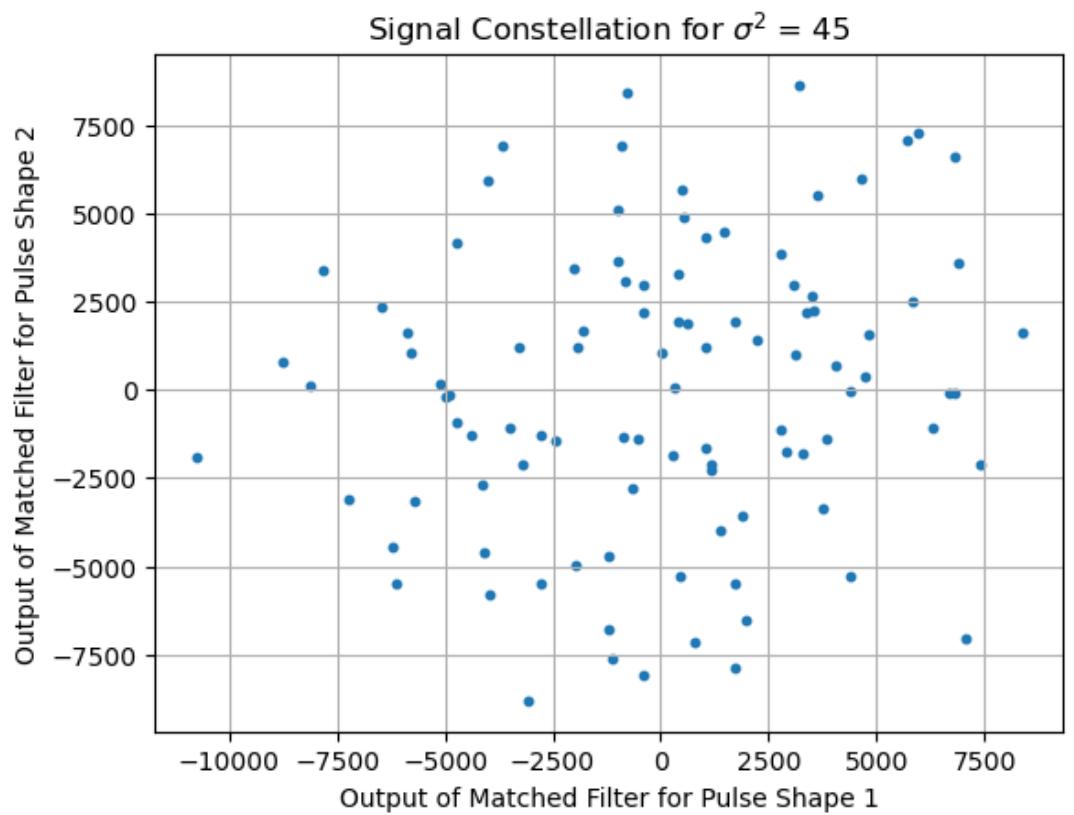
Signal Constellation for  $\sigma^2 = 5$



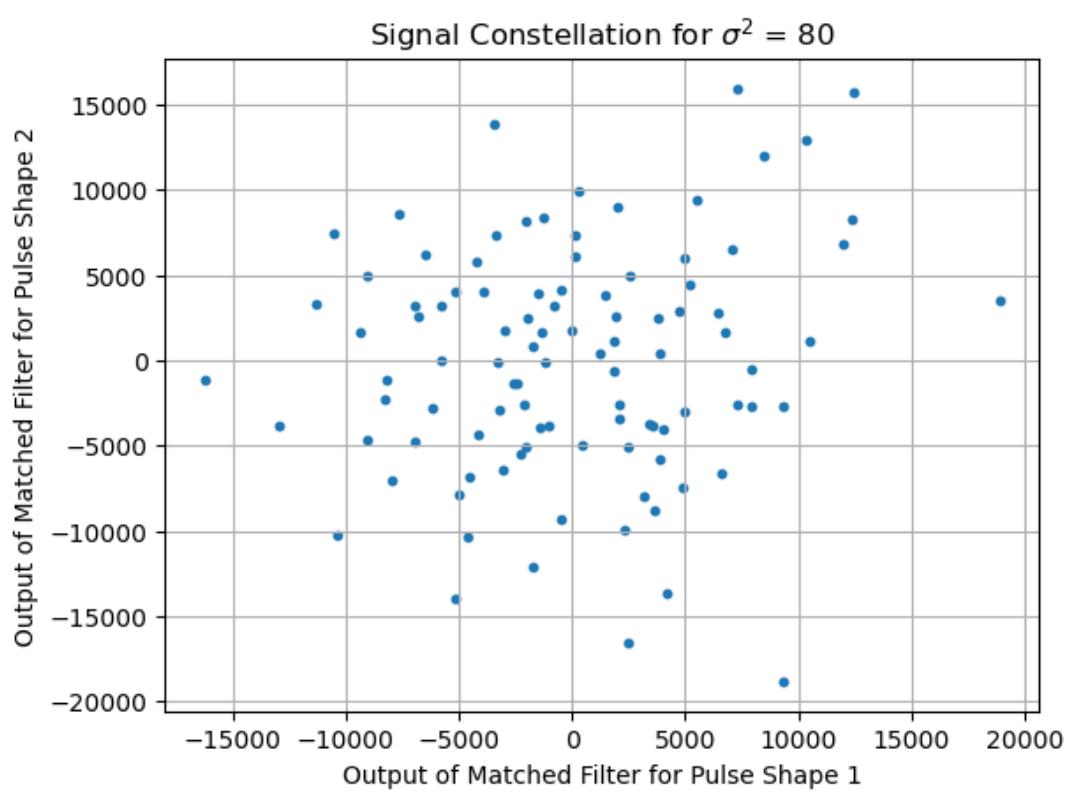
$\sigma^2 = 5$  نویز با واریانس



$\sigma^2 = 10$  نویز با واریانس



$\sigma^2 = 45$  واریانس با نویز



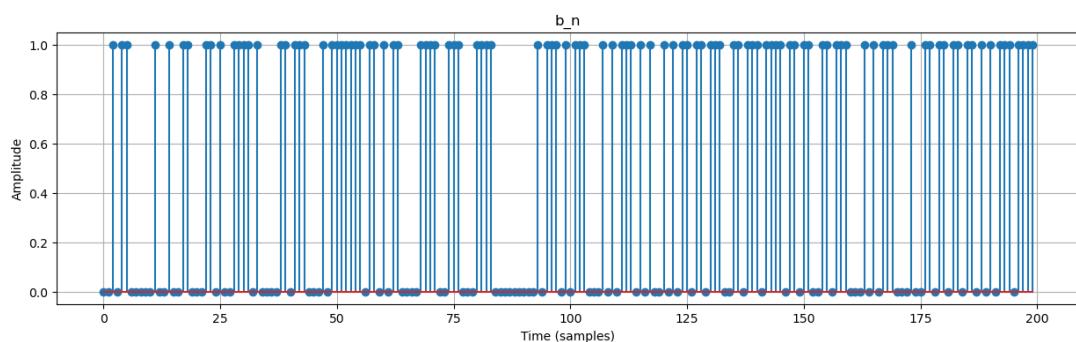
$$\text{نویز با واریانس } \sigma^2 = 8^{\circ}$$

واضح است مطابق انتظار، هرچه نویز کمتر باشد تراکم این اعداد بازیابی شده باید بیشتر باشد. تا جایی که به ازای  $\sigma^2$  نویز کاملاً بر هم منطبق هستند. رفته رفته با افزایش واریانس نویز این نقاط از هم فاصله میگیرند تا جایی که به نظر میرسد برای واریانس های بسیار بالا اصلاً هیچ تراکمی در منطقه خاصی وجود ندارد.

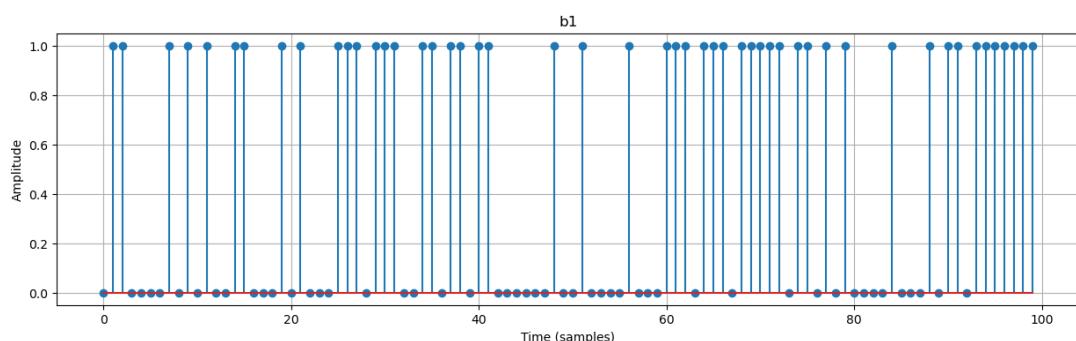
۲-۳

۱-۲-۳

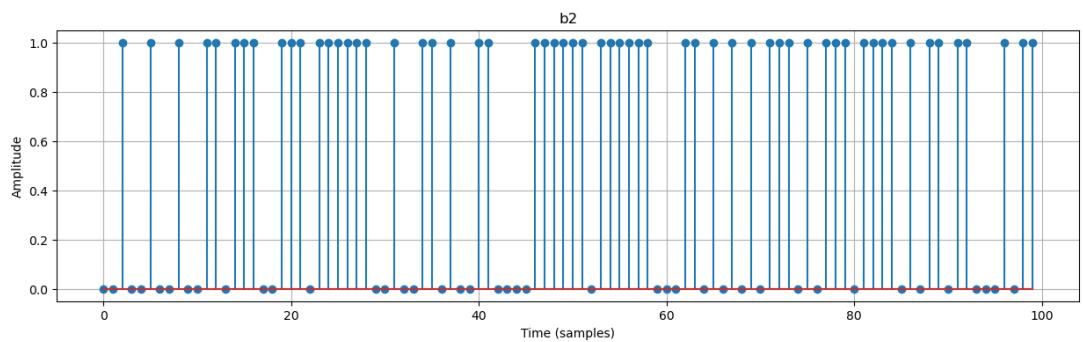
خروجی تابع های گفته شده در بخش ۲ را در اینجا نشان می دهیم. (به ازای یک ورودی  $200^{\circ}$  عضوی از  $(1, 0)$ )



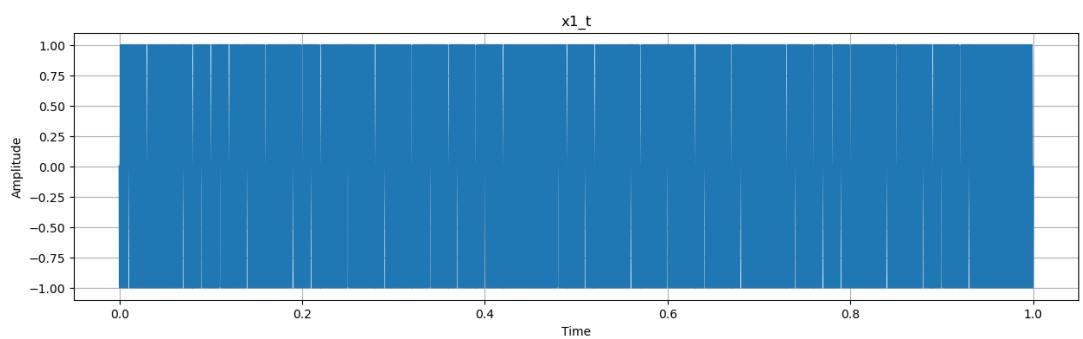
ورودی فرستنده  $b_n$



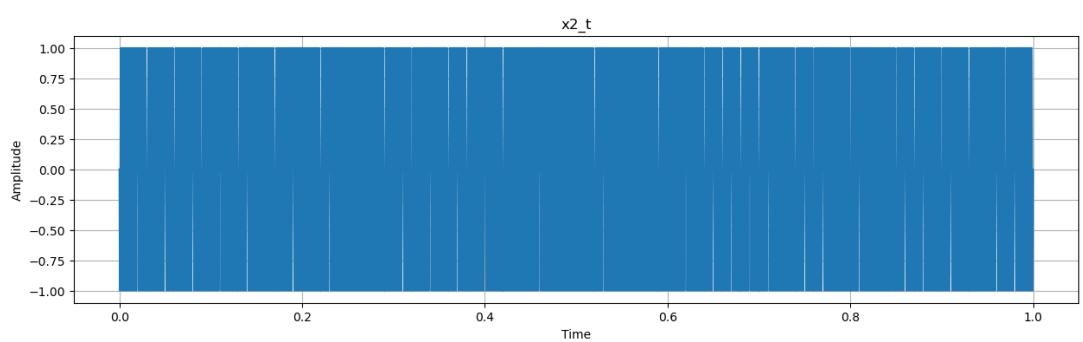
خروچی اول تابع Divide



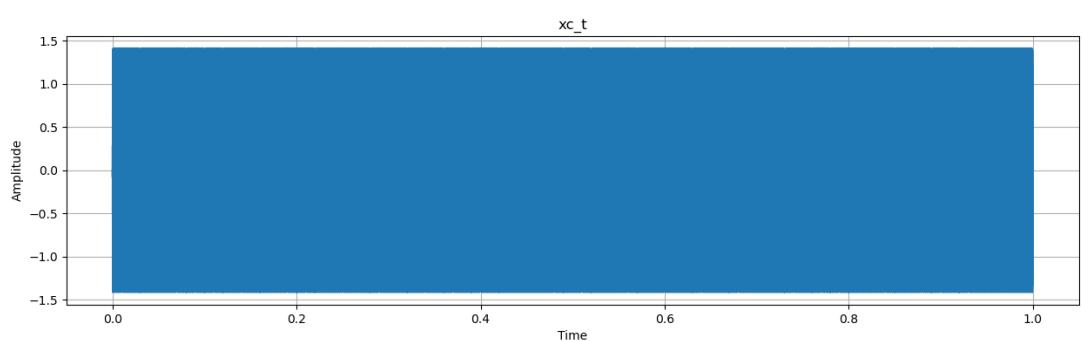
خروجی دوم تابع Divide



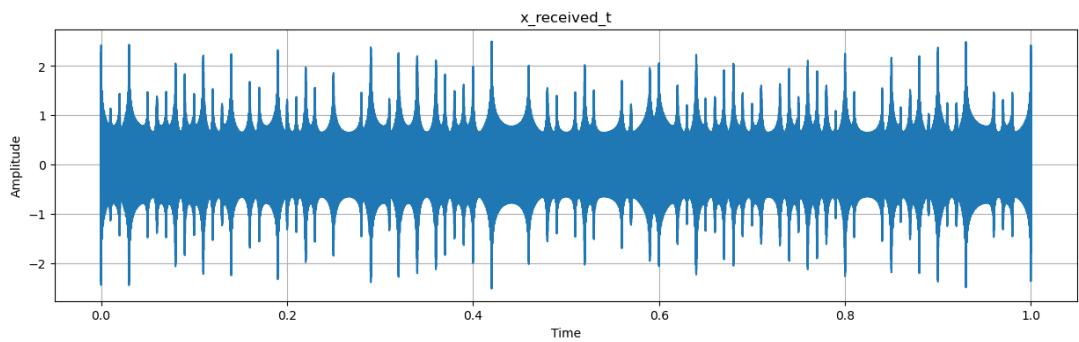
خروجی اول تابع PulseShaping



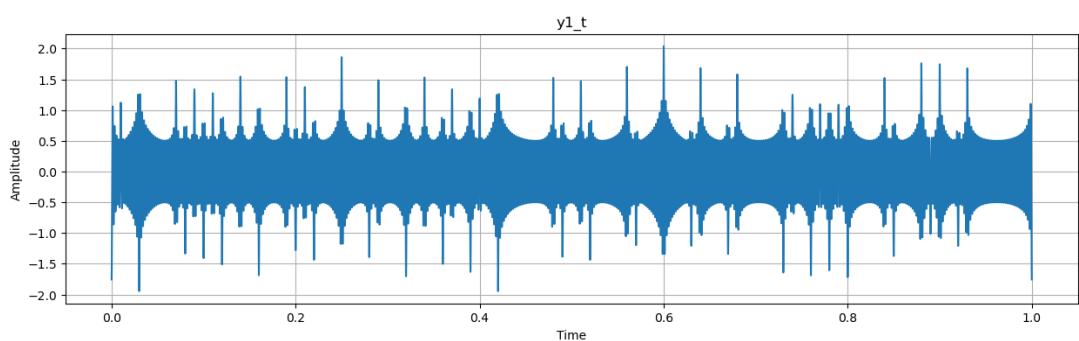
خروجی دوم تابع PulseShaping



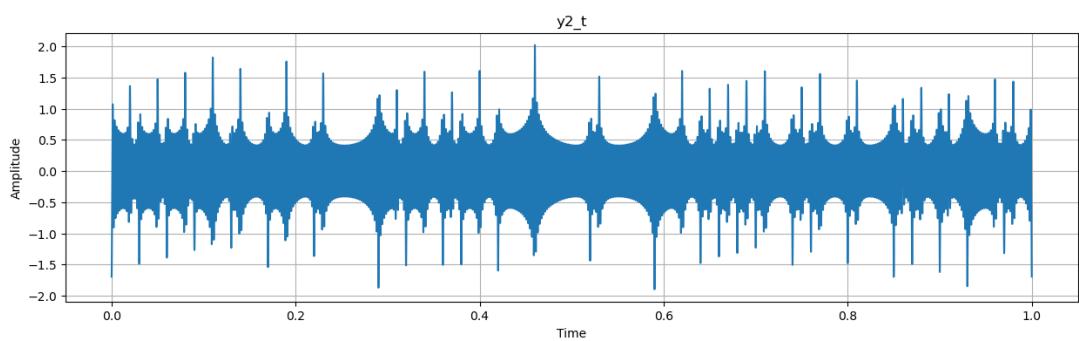
خروجی تابع AnalogMod



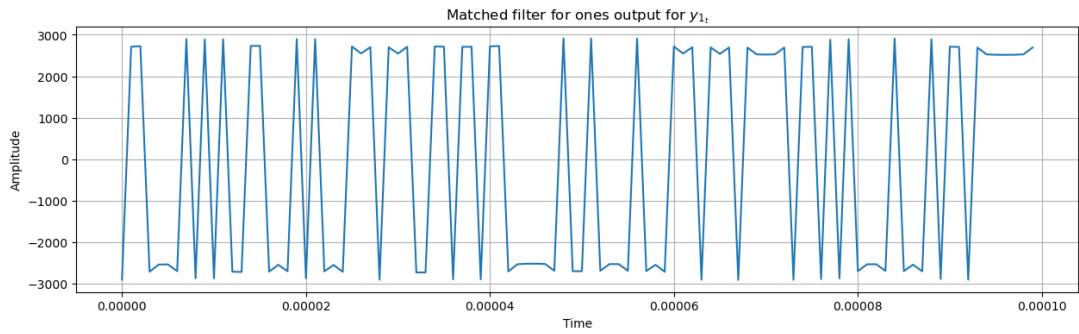
خروجی تابع Channel



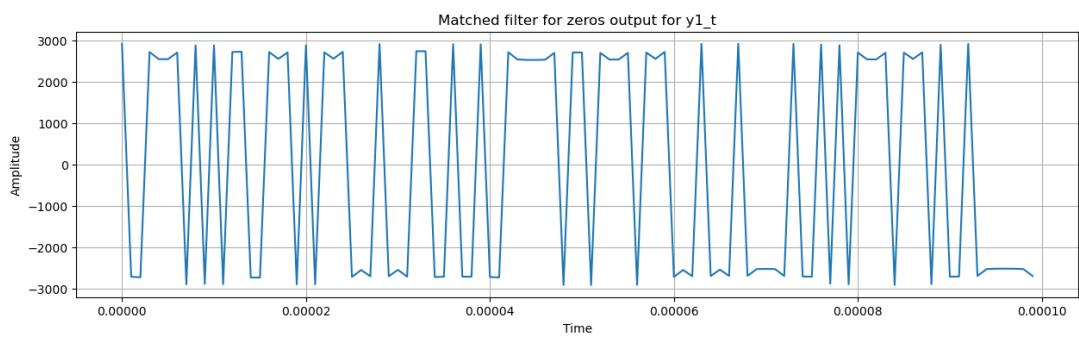
خروجی اول تابع AnalogDemod



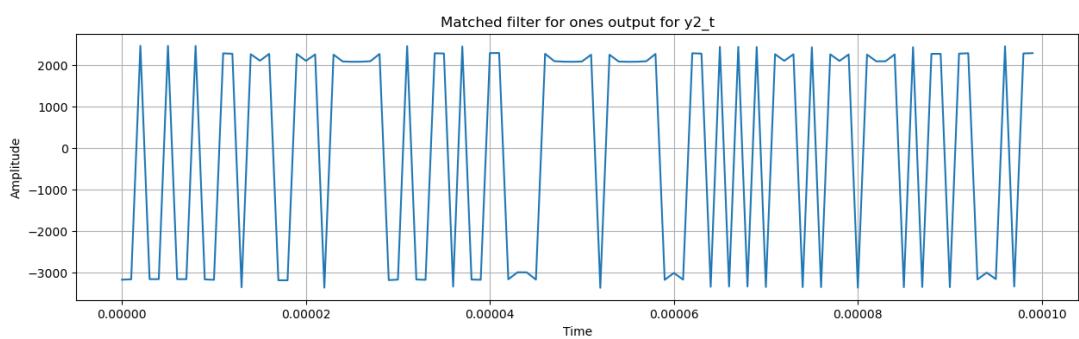
خروجی دوم تابع AnalogDemod



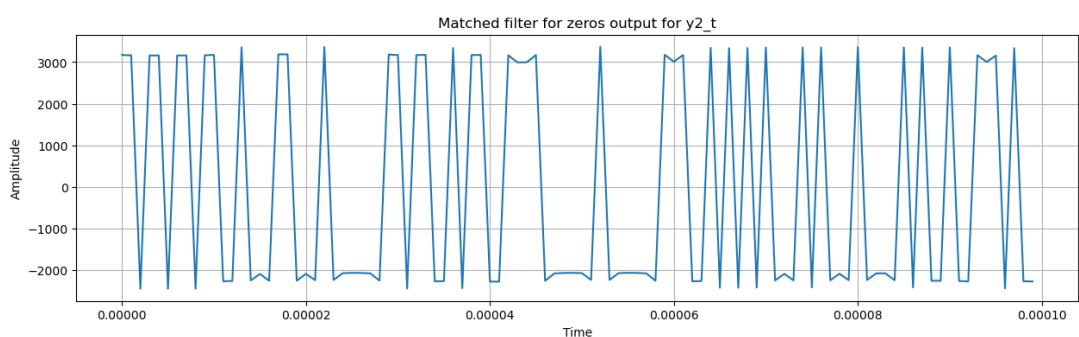
خروجی تابع MatchedFilt برای لول های یک رشته ای اول



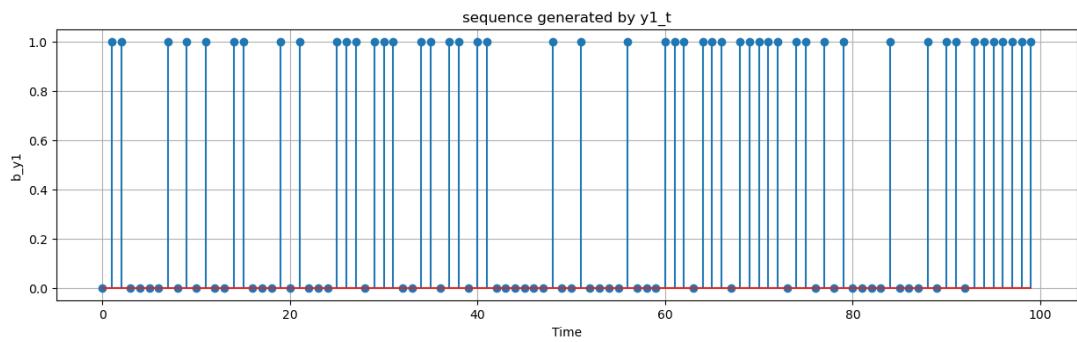
خروجی تابع MatchedFilt برای لول های صفر رشته ای اول



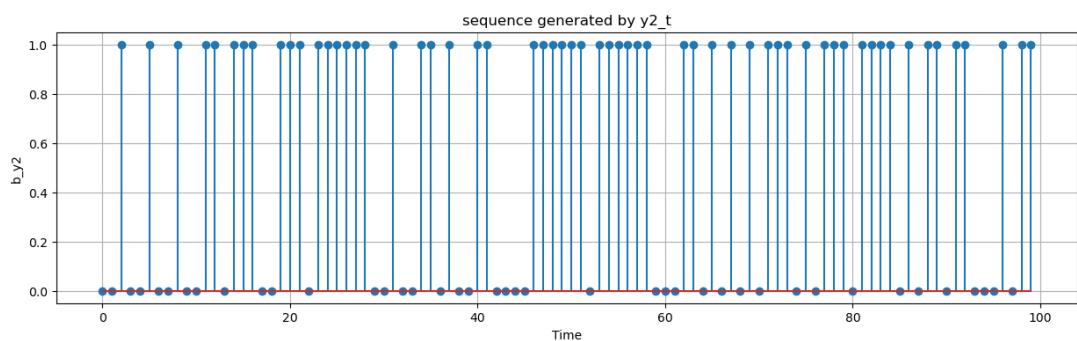
خروجی تابع MatchedFilt برای لول های یک رشته ای دوم



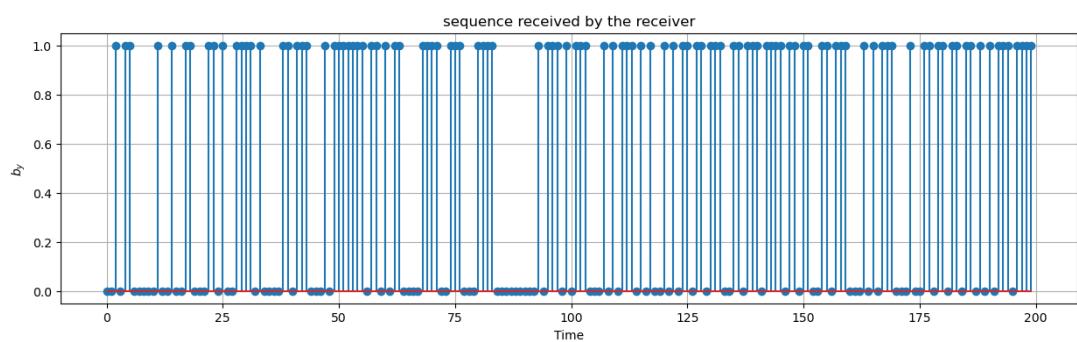
خروجی تابع MatchedFilt برای لول های صفر رشته ای دوم



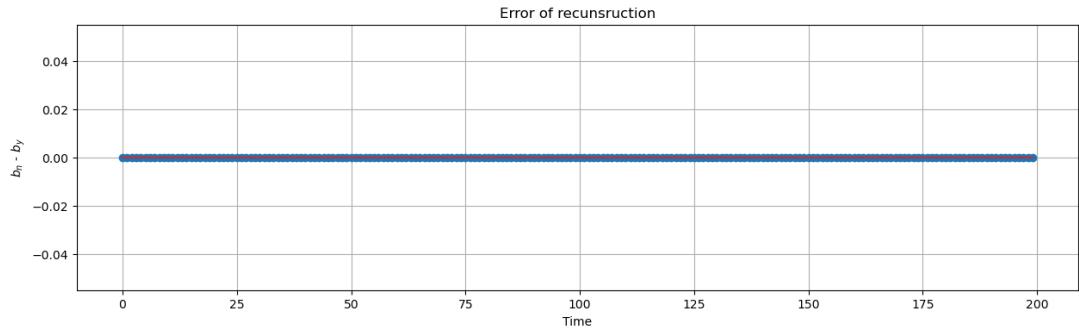
خروجی تابع MatchedFilt برای تخمین رشته اول



خروجی تابع MatchedFilt برای تخمین رشته دوم



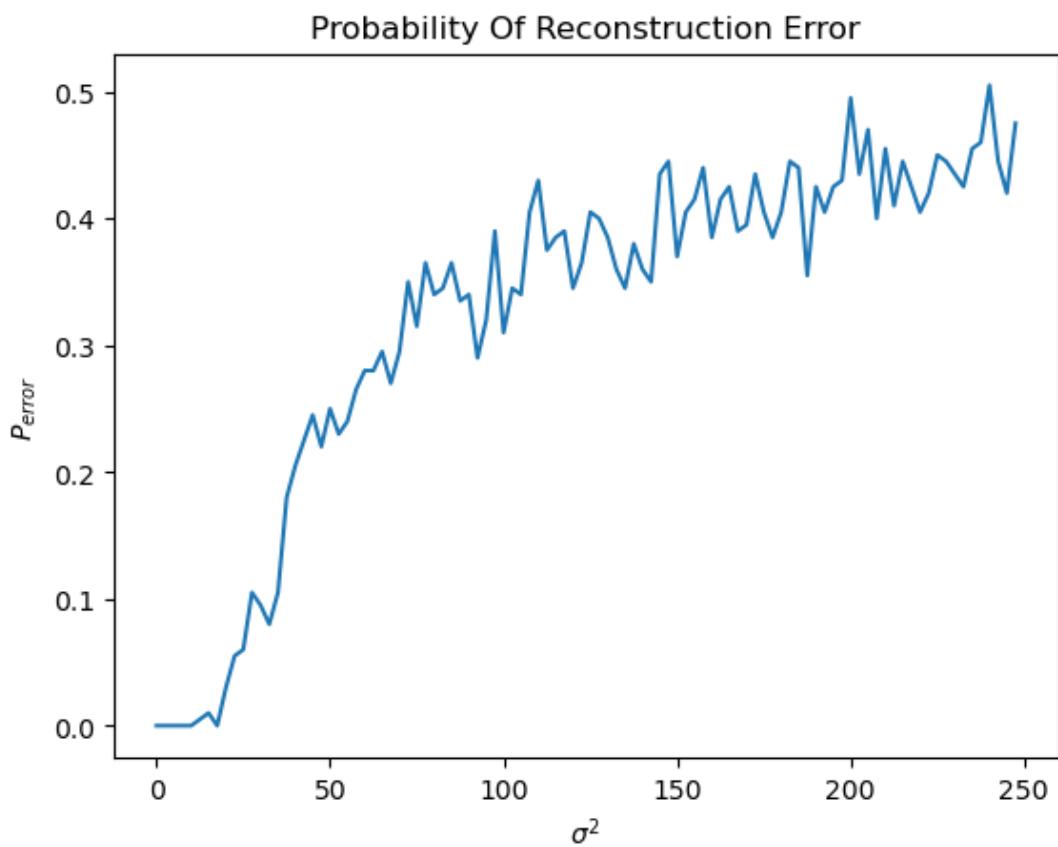
خروجی تابع Combine



### اختلاف رشته‌ی خروجی و ورودی

۲-۲-۳

مانند بخش ب بخش قبل پیش می‌رویم. خروجی‌های شبیه سازی به این صورت است:



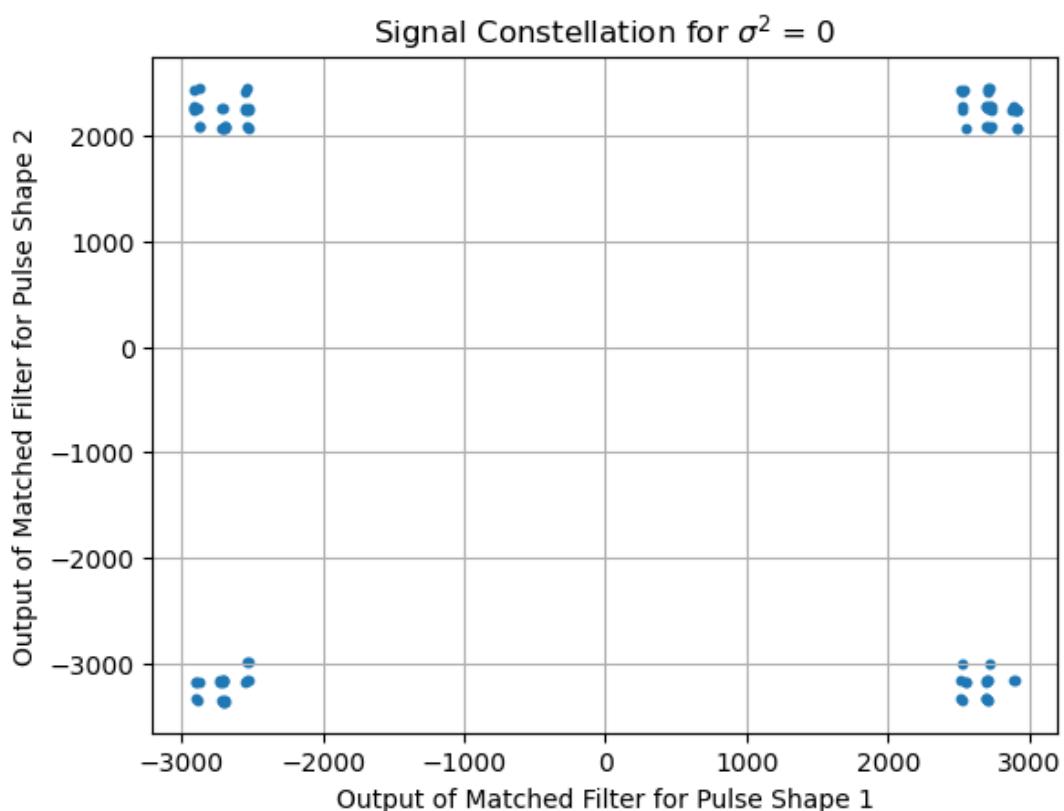
### احتمال خطای بازسازی بر حسب واریانس ویز کانال

واضح است انتظار داشتیم نتیجه‌ای مشابه با قسمت قبل را شاهد باشیم. البته تفاوت‌های ریزی نیز وجود دارد که در شکل واضح است. اما نکته مهم این است که همچنان مجموعه پروژه ما میتواند

تا حدی اثر نویز را خنثی کند و بدون هیچ اشتباہی سیگنال را بازیابی کند. اما رفته احتمال خطای بیشتر میشود تا اینکه به نزدیکی خطای  $5^\circ$  درصد میرسیم که یعنی اصلاً یک سیگنال رندوم جدید تولید شده است (مشابه با علتی که در قسمت قبل توضیح دادیم) لازم به ذکر است رفتار کمی غیرخطی شکل فوق به این خاطر است که نمودار با  $100$  نمونه ترسیم شده تا زمان زیادی برای پردازش صرف نشود.

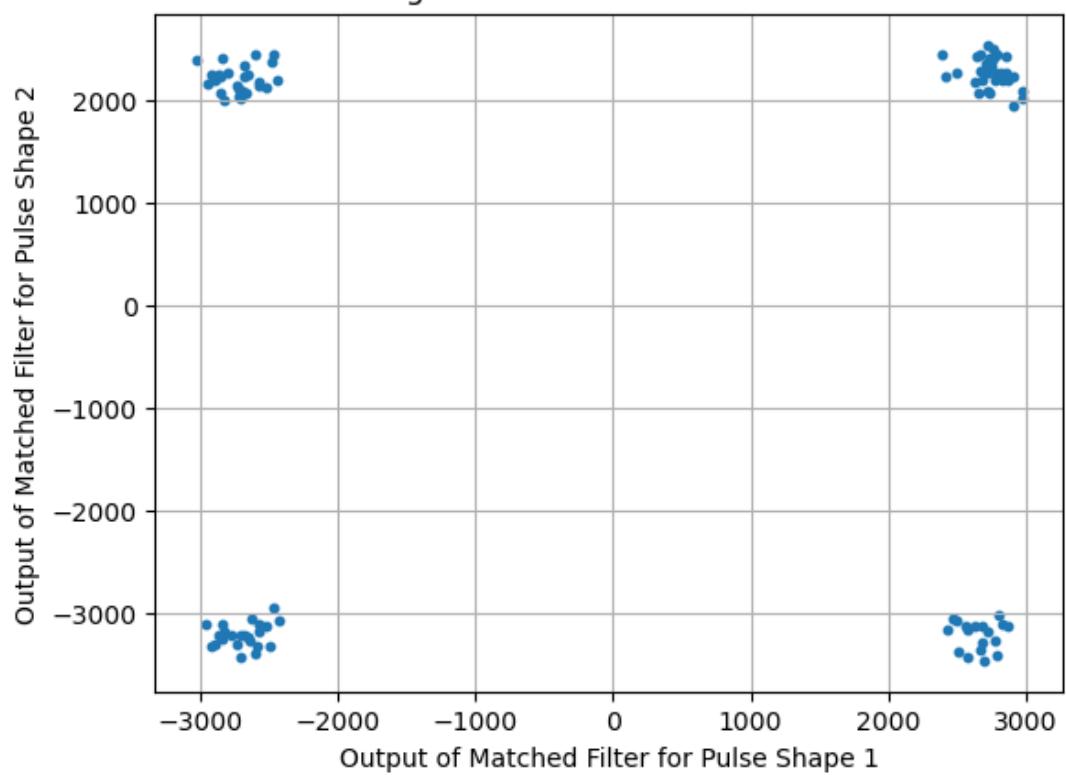
۳-۲-۳

مانند بخش ج بخش قبل پیش می رویم. خروجی های شبیه سازی به این صورت است :



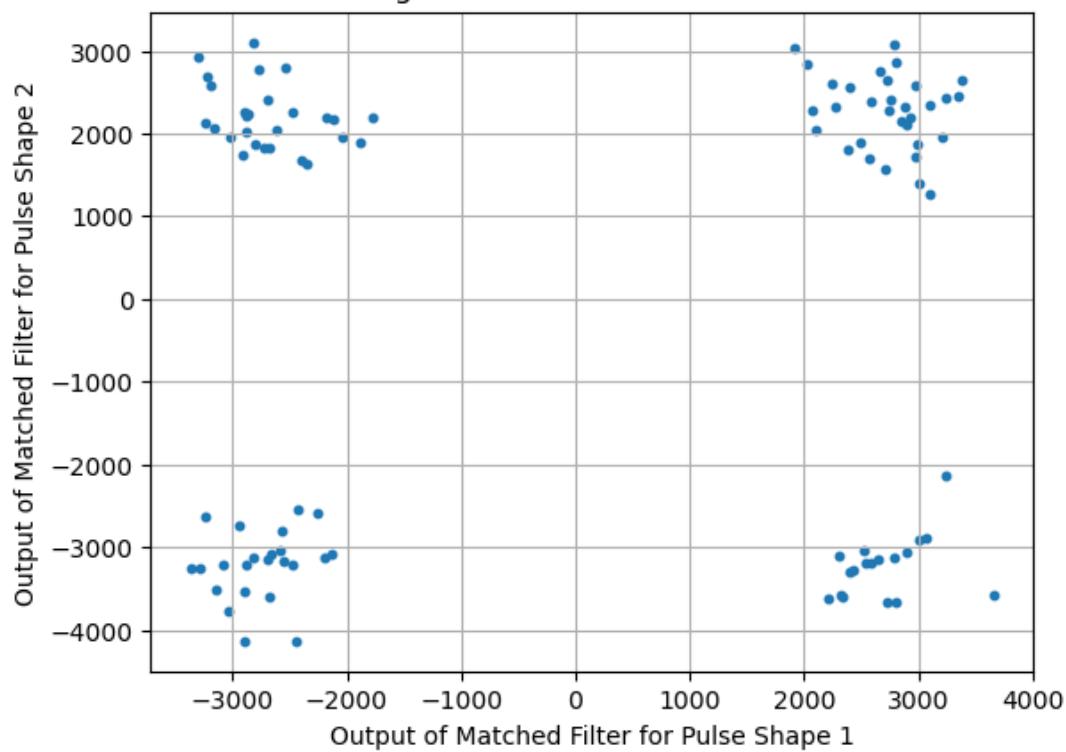
$$\sigma^2 = 0 \text{ نویز با واریانس } ۰$$

Signal Constellation for  $\sigma^2 = 1$

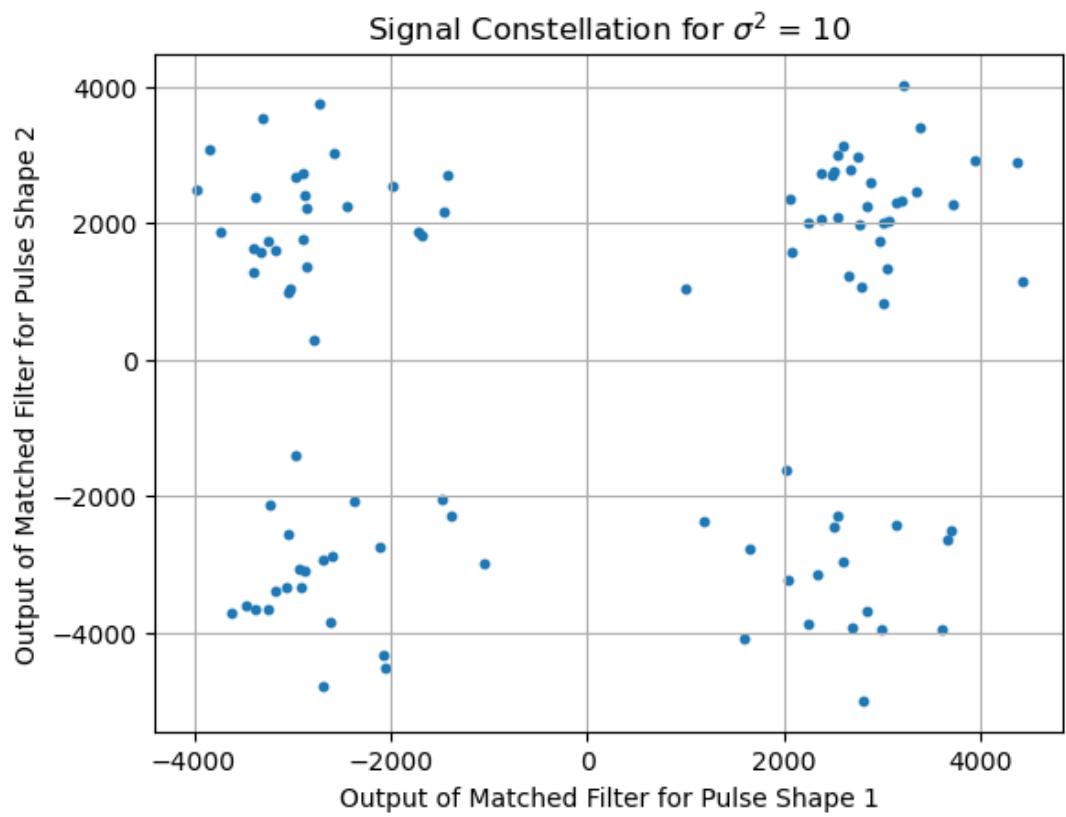


نویز با واریانس ۱

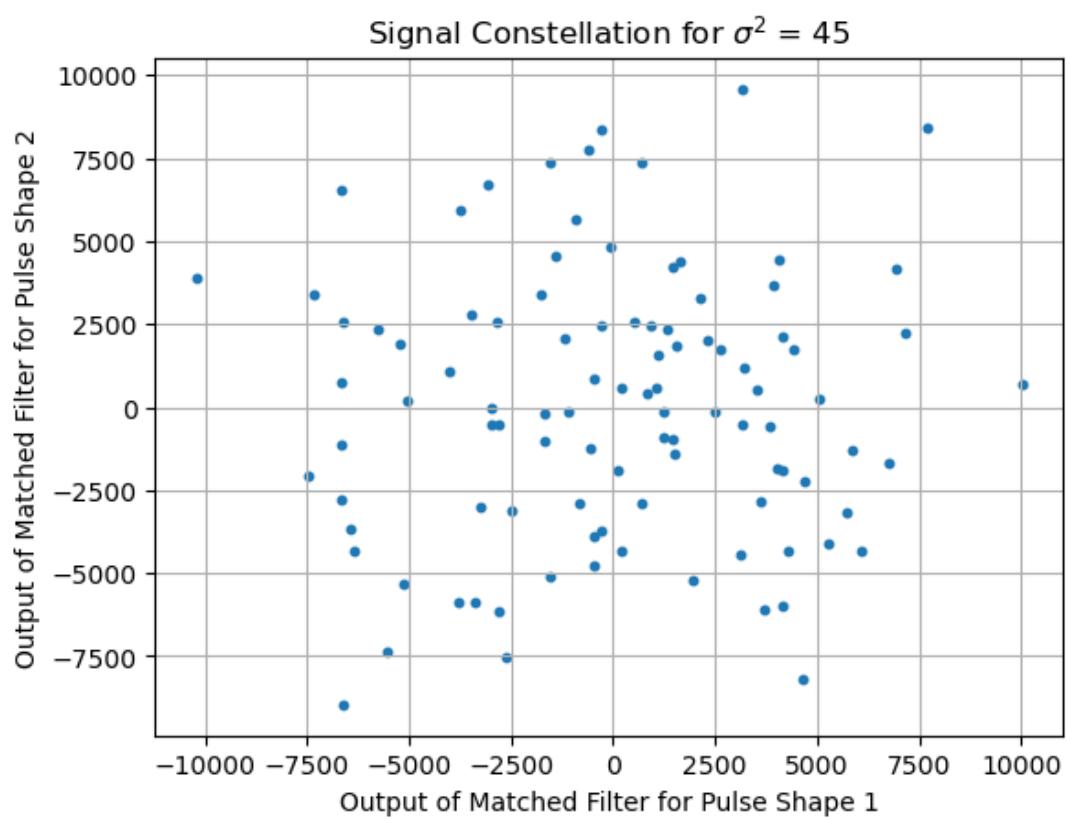
Signal Constellation for  $\sigma^2 = 5$



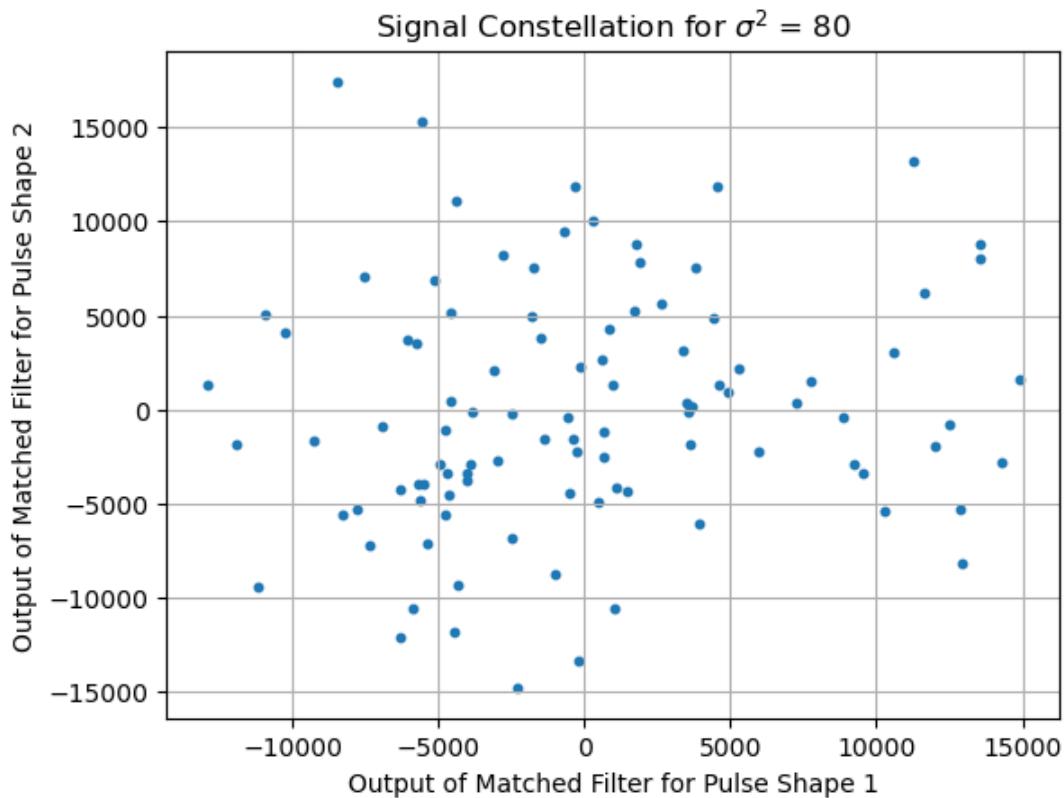
نویز با واریانس ۵



$\sigma^2 = 10$  واریانس با نویز



$$\text{نویز با واریانس } \sigma^2 = 45$$



$$\text{نویز با واریانس } \sigma^2 = 80$$

نکته مهم این است که منظومه سیگنال ترسیم شده برای حالت  $\sigma^2_{noise} = 0$  علی رغم اینکه تمام نقاط روی هم ترسیم نشده اند، اما همچنان از شکل میتوان استنباط کرد که فرایند بازسازی بدون خطأ بوده است. علت اینکه در هر ناحیه بجای تمرکز بر یک نقطه، تمرکز بر چهار قسمت است این است که سیگنال های متناظر با بیت های ما سینوسی هستند. از طرفی فیلتر های ما با توجه به سبکی که تعریف شان کرده ایم ایده آل نیستند و برای همین در ابتدای سیگنال کمی دامنه آن ها کمتر است. به نوعی میتوان گفت سیگنال های خروجی از فیلتر ها یک بخش transient دارند و یک بخش steady state که بخش گذرا باعث بوجود آمدن این پدیده شده اسم. همچنین میتوانیم ببینیم که در حالت PAM نویز تاثیر کمتری می گذارد نسبت به حالت PSK.

برای اثبات متعامد بودن این دو سیگنال، باید رابطه‌ی زیر برقرار باشد:

$$\int_0^{T_s} \sin(2\pi \times 1500 \times t) \sin(2\pi \times 1000 \times t) dt = 0$$

$T_s$  دوره‌ی تناوب است. برای اثبات این رابطه از روابطی که در اسلاید های درس آمده است استفاده خواهیم کرد:

$$\int_0^{T_s} \sin(2\pi \times 1500 \times t) \sin(2\pi \times 1000 \times t) dt = \frac{1}{2} \int_0^{T_s} [\cos(2\pi \times 500 \times t) - \cos(2\pi \times 2500 \times t)] dt$$

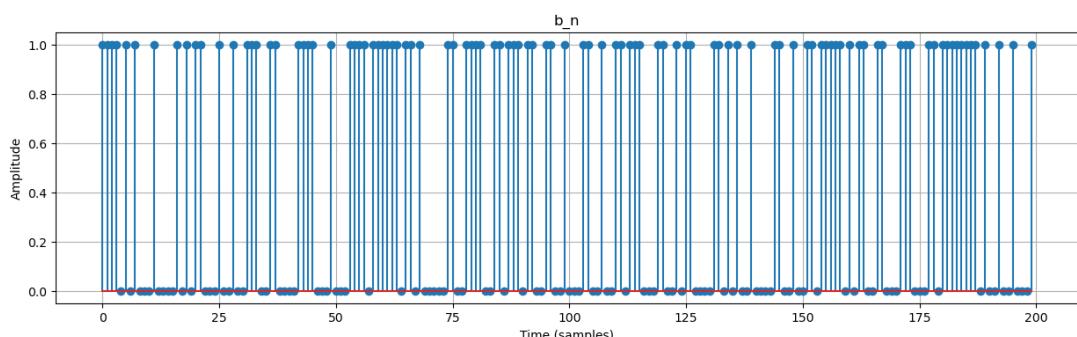
$$= \frac{1}{2} T_s \frac{\sin(2\pi \times 500 \times T_s)}{2\pi \times 500 \times T_s} - \frac{1}{2} T_s \frac{\sin(2\pi \times 2500 \times T_s)}{2\pi \times 2500 \times T_s}$$

و از آنجا که  $T_s = \text{integer}$  بنابراین هر دو سینوس برابر صفر خواهند بود. لذا:

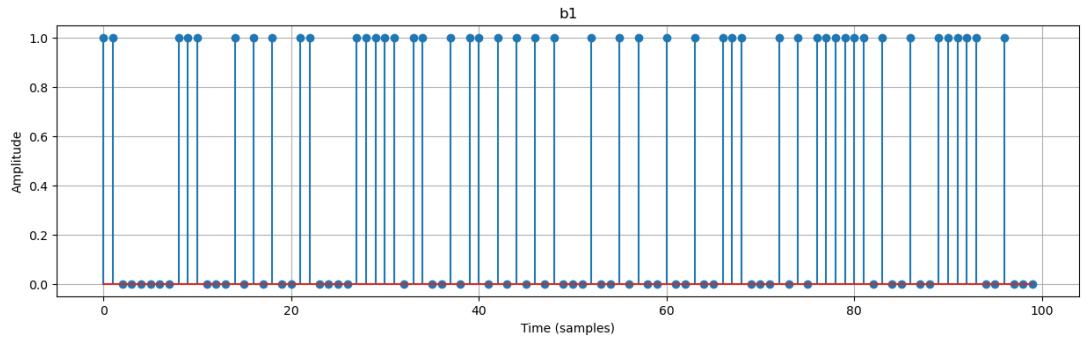
$$\int_0^{T_s} \sin(2\pi \times 1500 \times t) \sin(2\pi \times 1000 \times t) dt = 0$$

بنابراین میتوانیم به این نتیجه برسیم که این دو سیگنال با یکدیگر متعامد هستند.

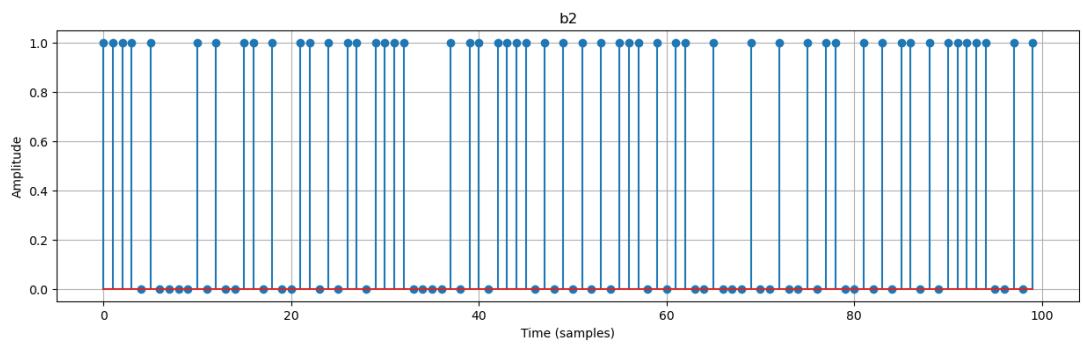
خروجی تابع‌های گفته شده در بخش ۲ را در اینجا نشان می‌دهیم. (به ازای یک ورودی ۲۰۰ عضوی از  $\{1, 0\}$ )



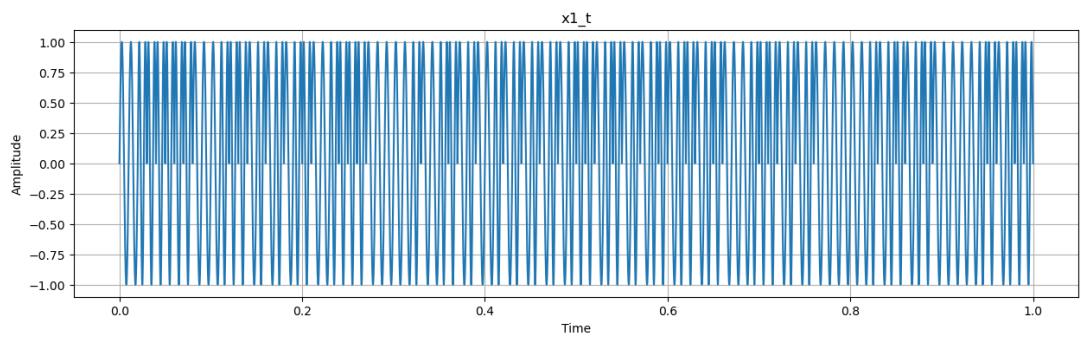
ورودی فرستنده  $b_n$



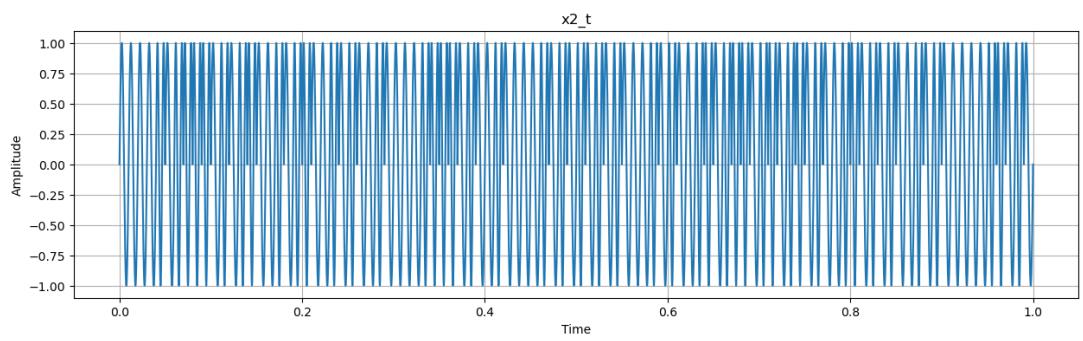
خروجی اول تابع Divide



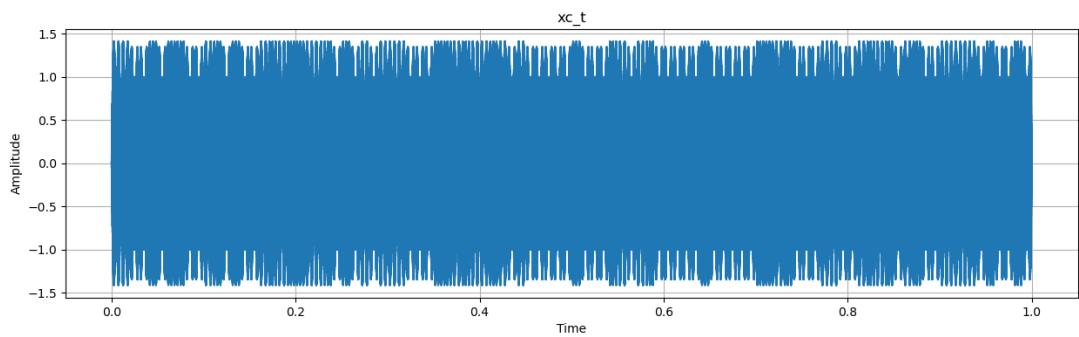
خروجی دوم تابع Divide



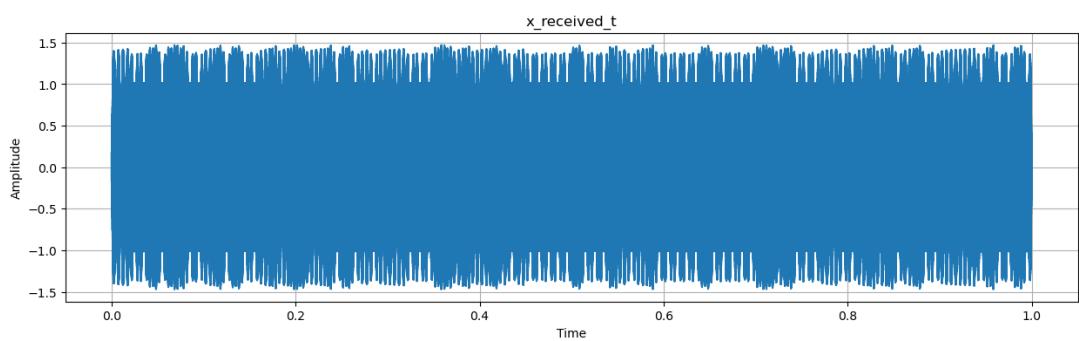
خروجی اول تابع PulseShaping



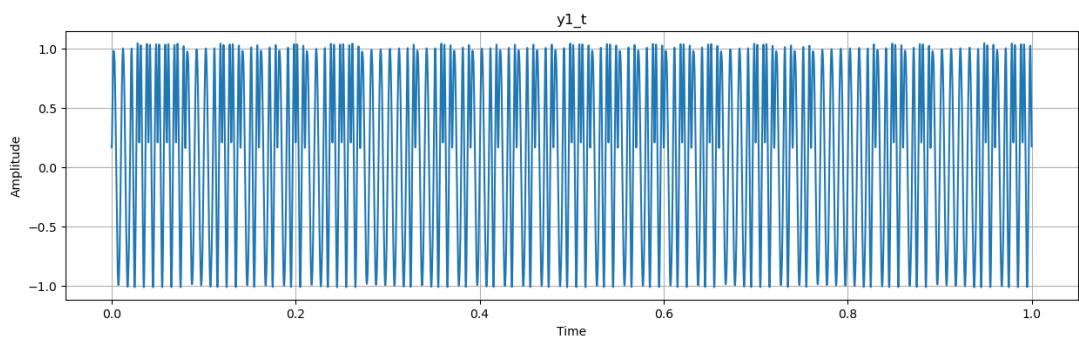
خروجی دوم تابع PulseShaping



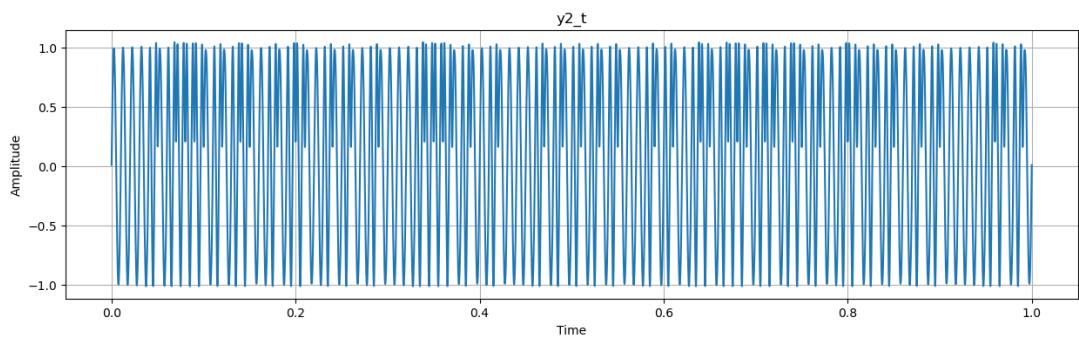
خروجی تابع AnalogMod



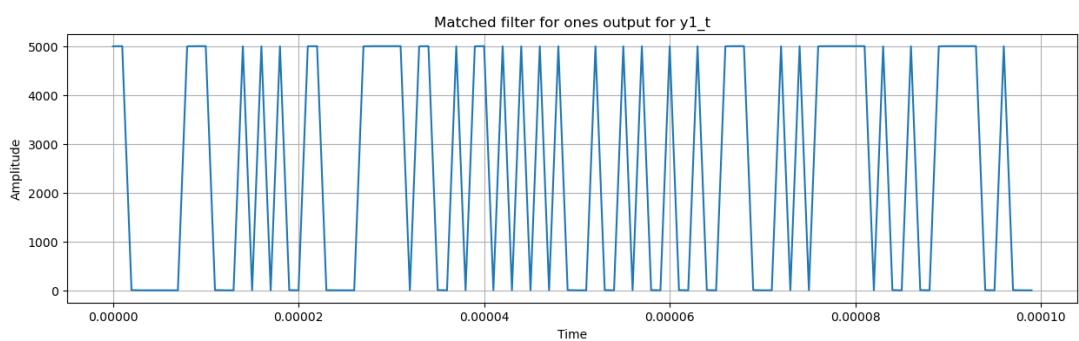
خروجی تابع Channel



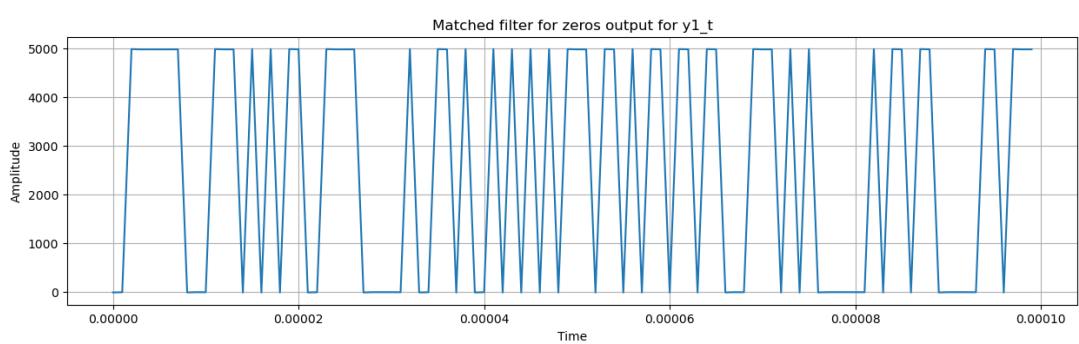
خروجی اول تابع AnalogDemod



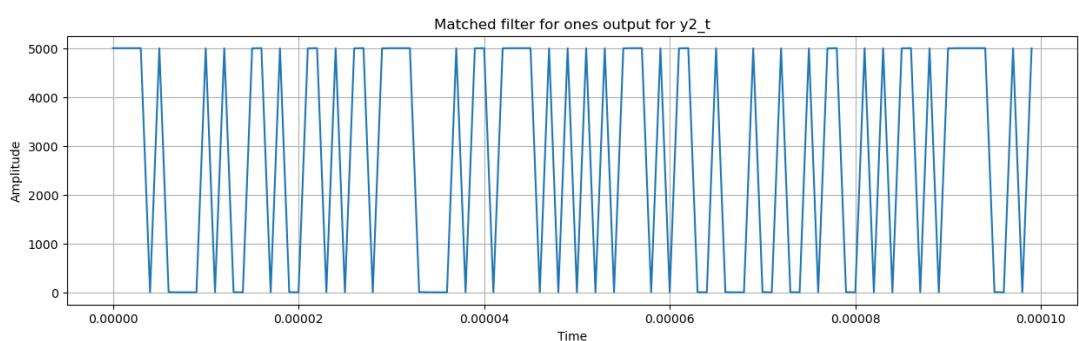
خروجی دوم تابع AnalogDemod



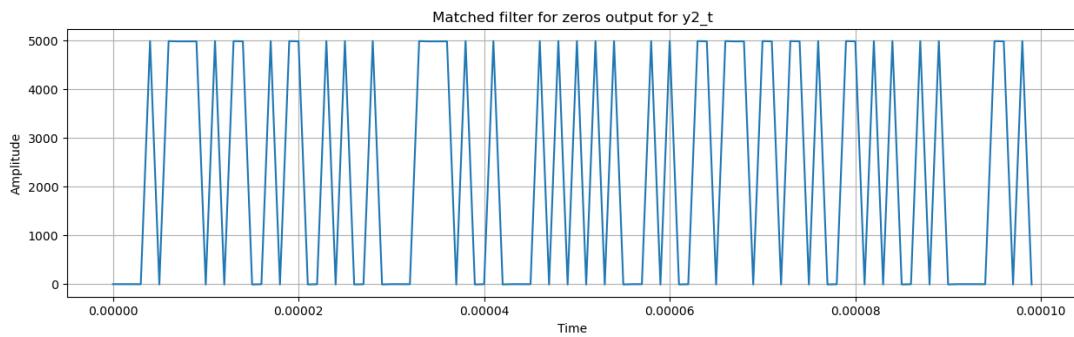
خروجی تابع MatchedFilt برای لول های یک رشته ای اول



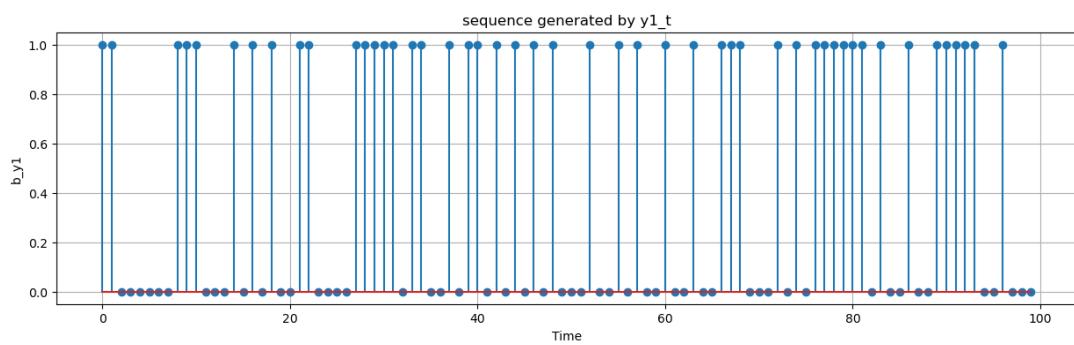
خروجی تابع MatchedFilt برای لول های صفر رشته ای اول



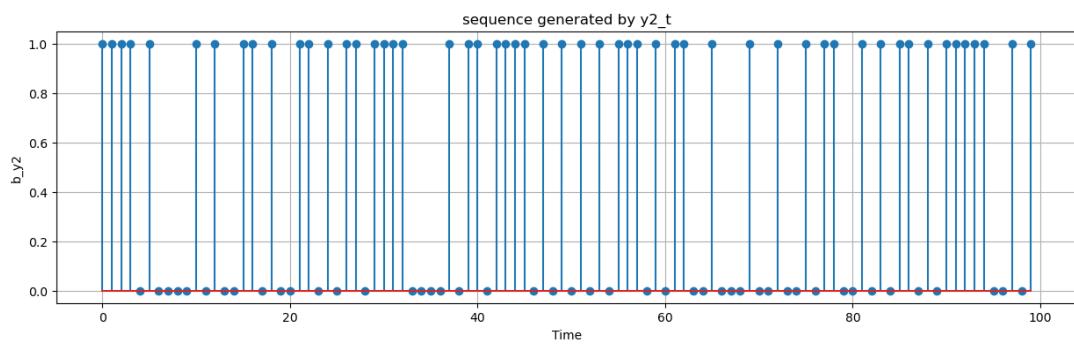
خروجی تابع MatchedFilt برای لول های یک رشته ای دوم



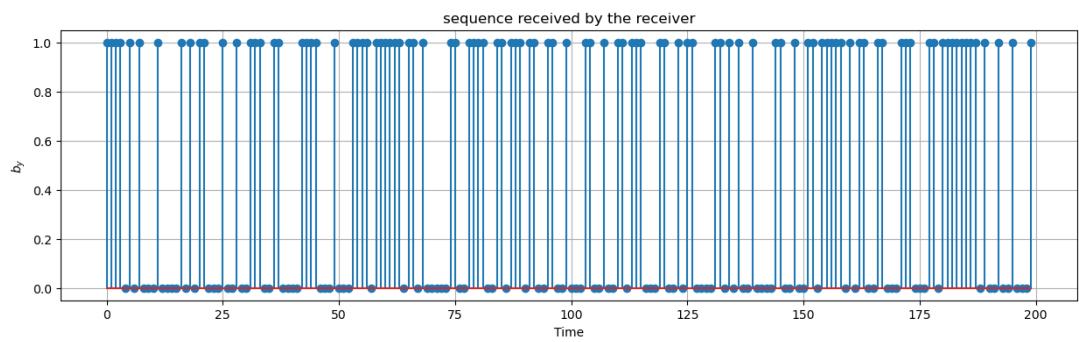
خروجی تابع MatchedFilt برای لول های صفر رشته ای دوم



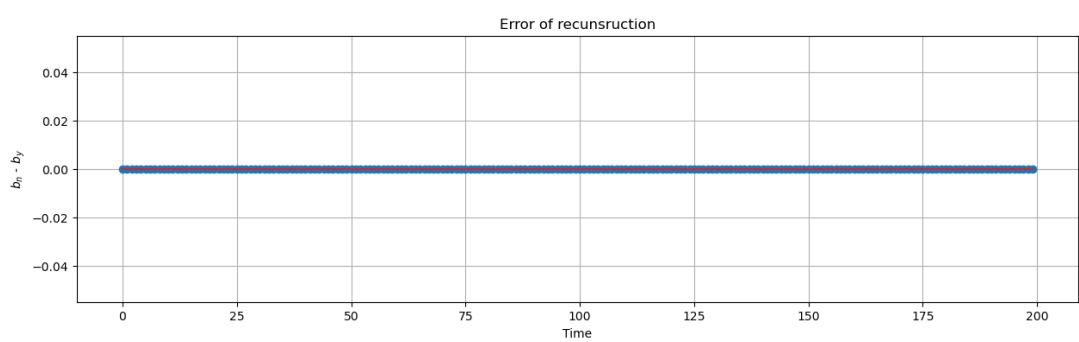
خروجی تابع MatchedFilt برای تخمین رشته اول



خروجی تابع MatchedFilt برای تخمین رشته دوم



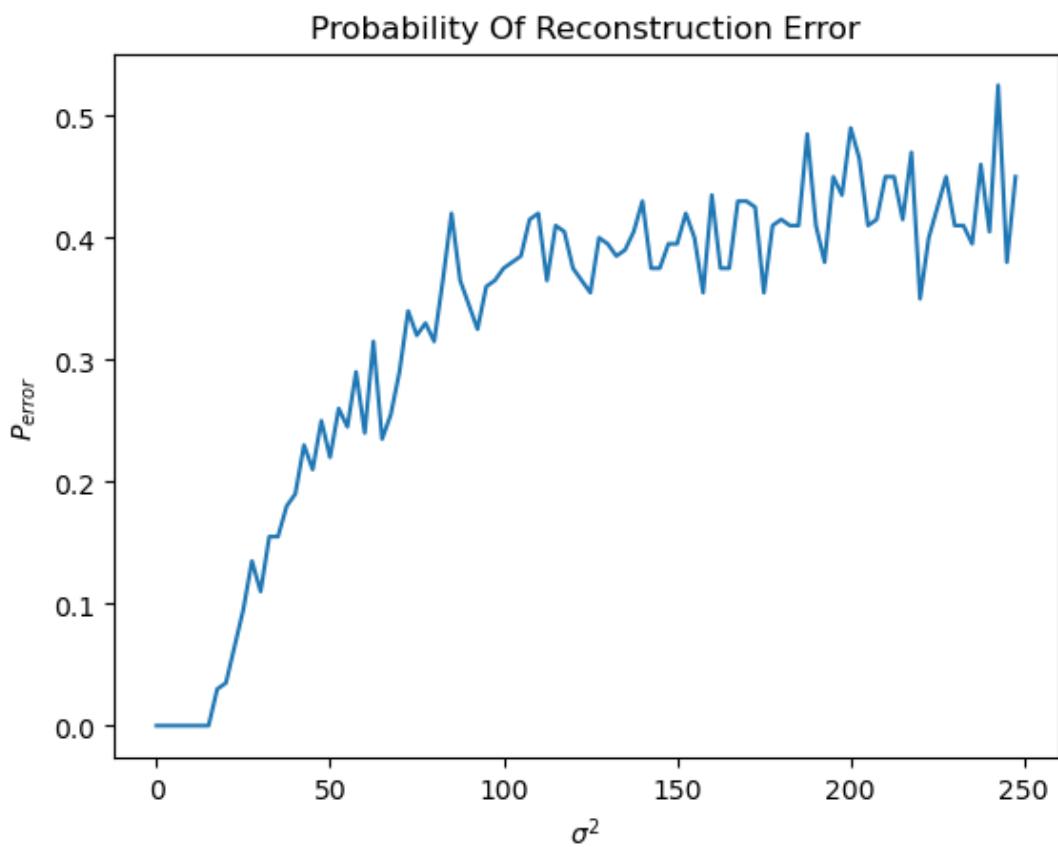
تابع خروجی Combine



اختلاف رشته‌ی خروجی و ورودی

۳-۳-۳

مانند بخش ب بخش قبل پیش می‌رویم. خروجی‌های شبیه سازی به این صورت است:



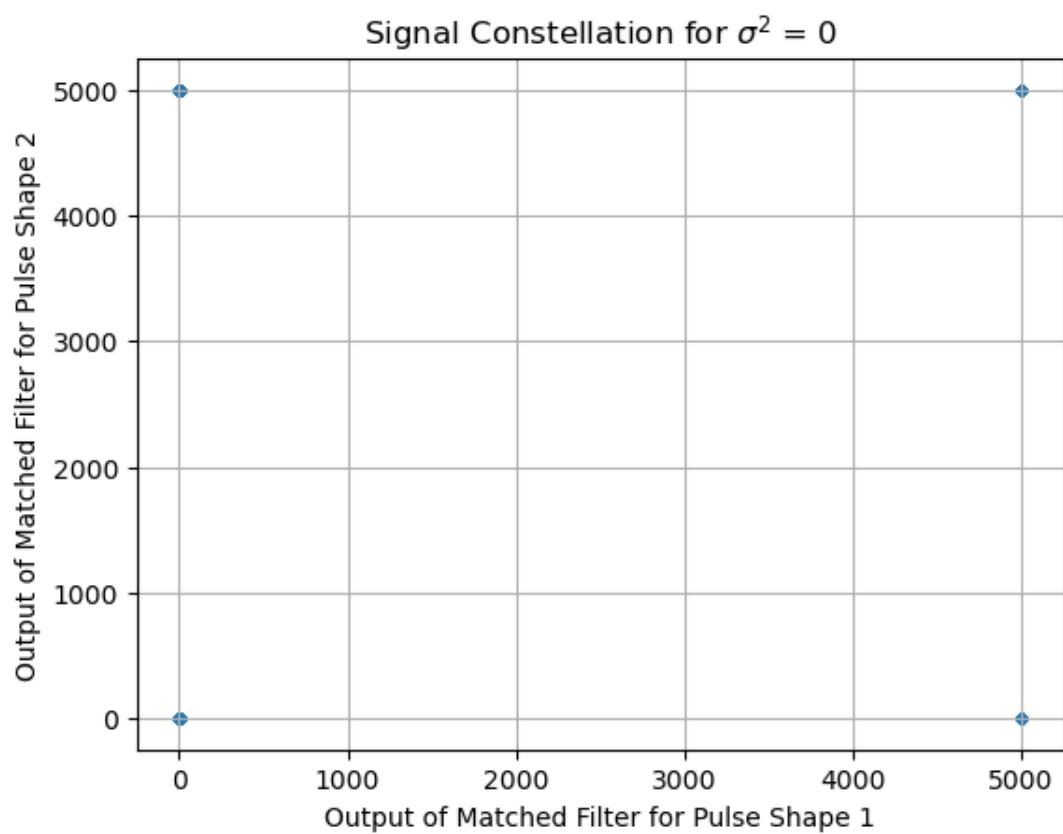
### احتمال خطای بازسازی بر حسب واریانس ویز کانال

واضح است انتظار داشتیم نتیجه ای مشابه با قسمت قبل را شاهد باشیم. البته تفاوت های ریزی نیز وجود دارد که در شکل واضح است. اما نکته مهم این است که همچنان مجموعه پروژه ما میتواند تا حدی اثر نویز را خنثی کند و بدون هیچ اشتباہی سیگنال را بازیابی کند. اما رفته احتمال خطأ بیشتر میشود تا اینکه به نزدیکی خطای  $5^\circ$  درصد میرسیم که یعنی اصلاً یک سیگنال رندوم جدید تولید شده است (مشابه با علتی که در قسمت قبل توضیح دادیم)

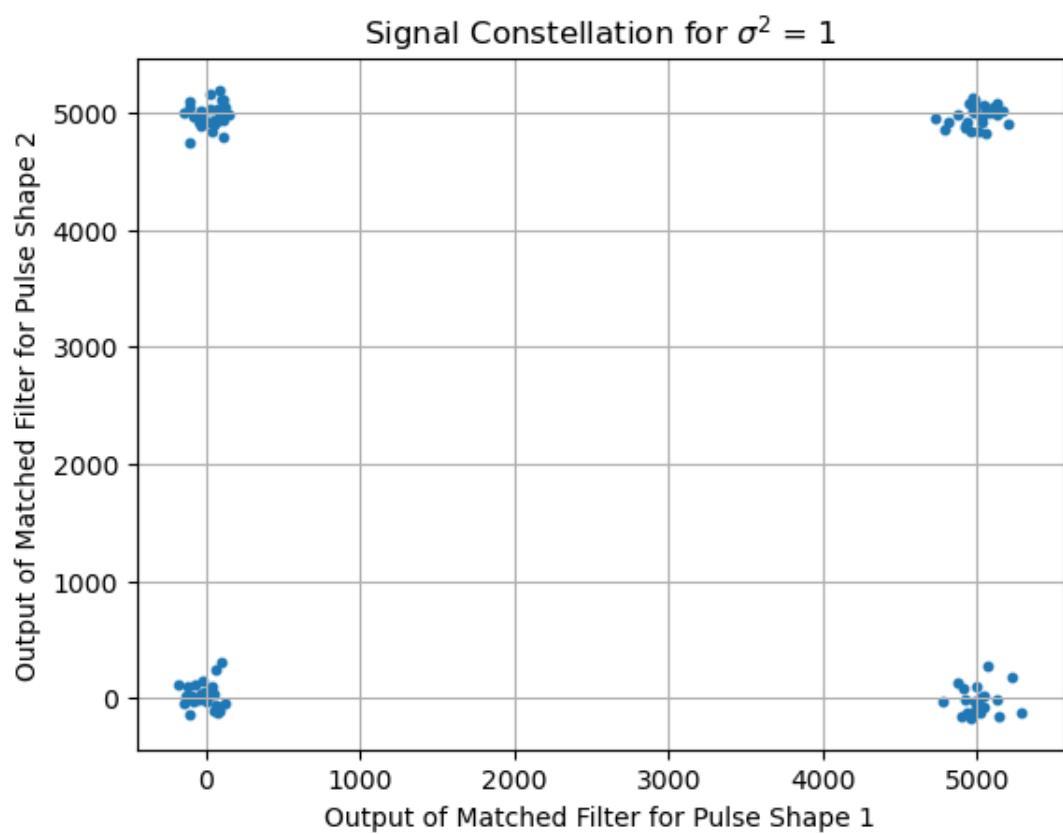
لازم به ذکر است رفتار کمی غیرخطی شکل فوق به این خاطر است که نمودار با  $100$  نمونه ترسیم شده تا زمان زیادی برای پردازش صرف نشود.

۴-۳-۳

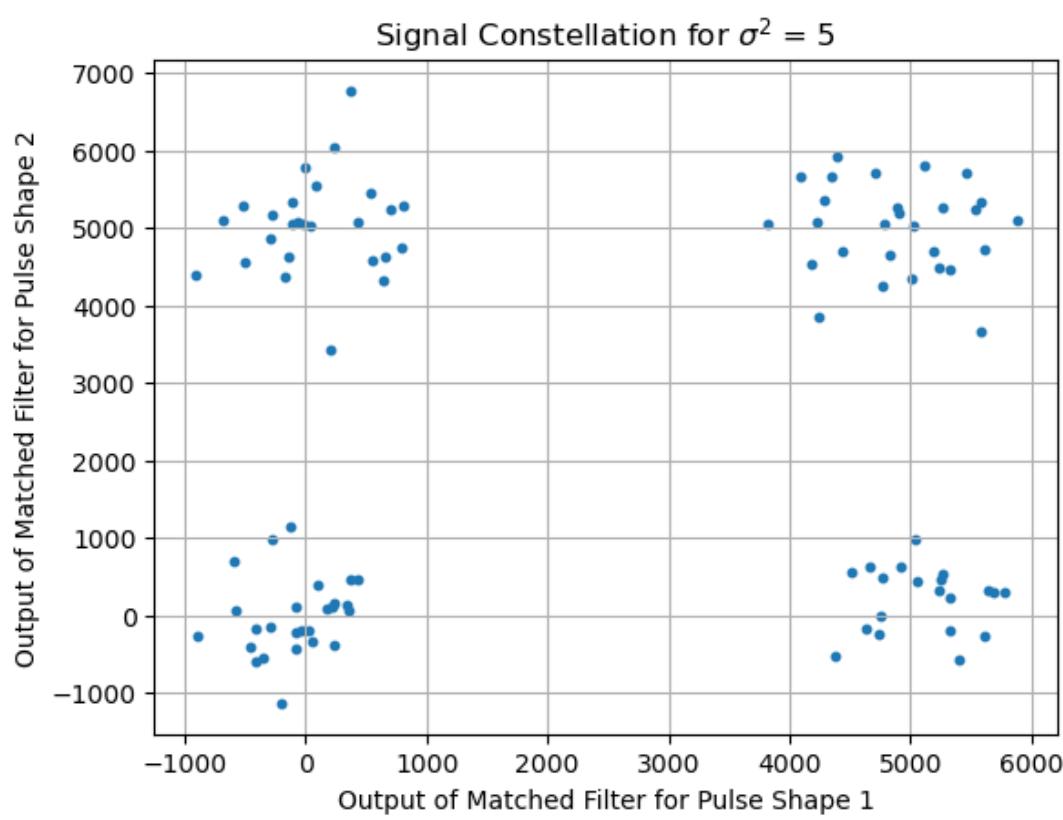
مانند بخش ج بخش قبل پیش می رویم. خروجی های شبیه سازی به این صورت است :



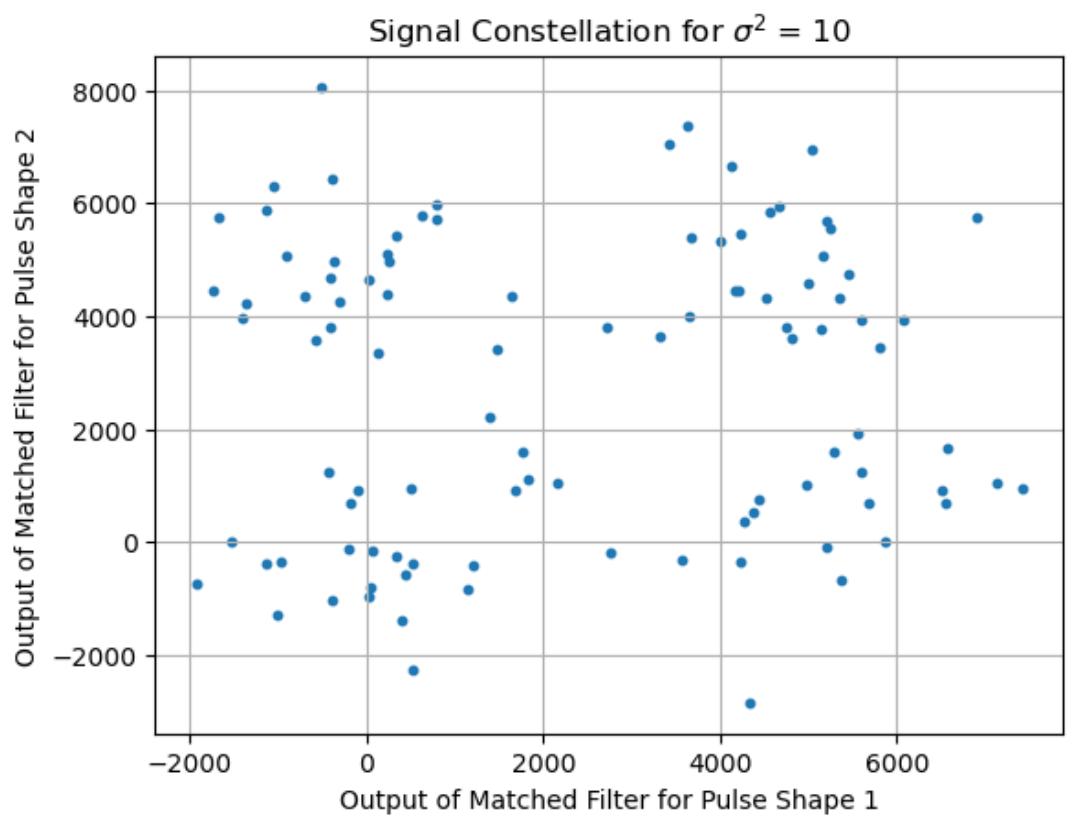
نویز با واریانس  $\sigma^2 = 0$



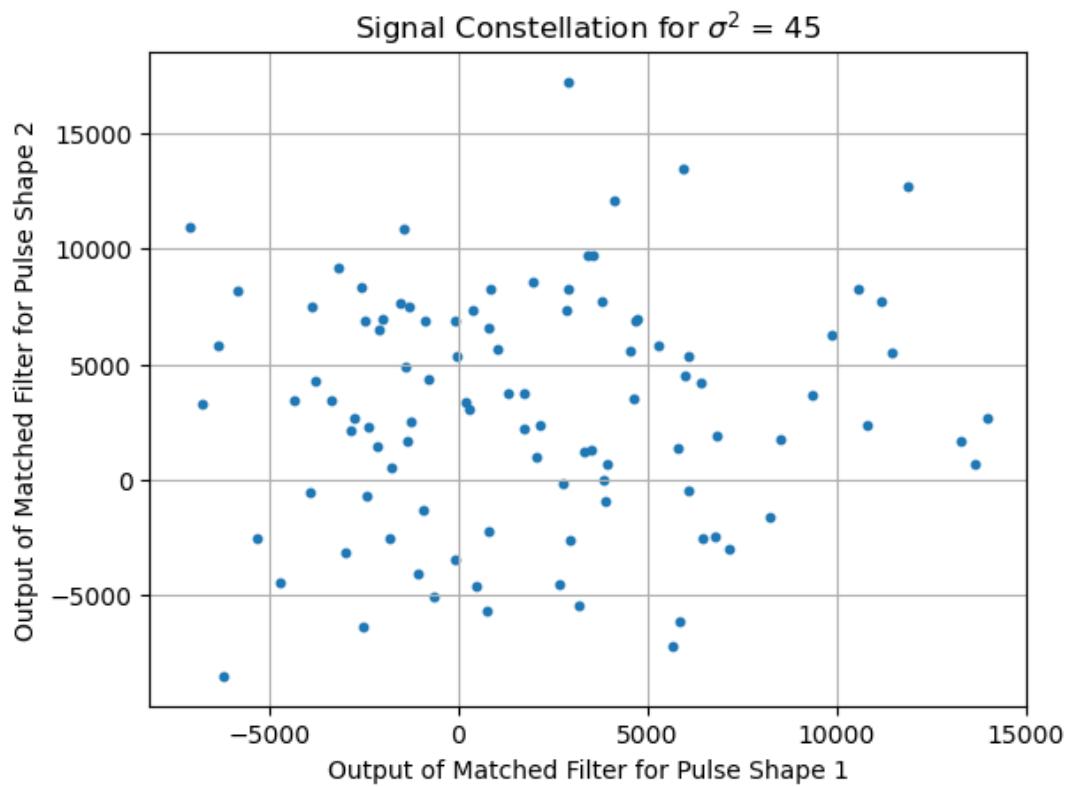
$\sigma^2 = 1$  واریانس با نویز



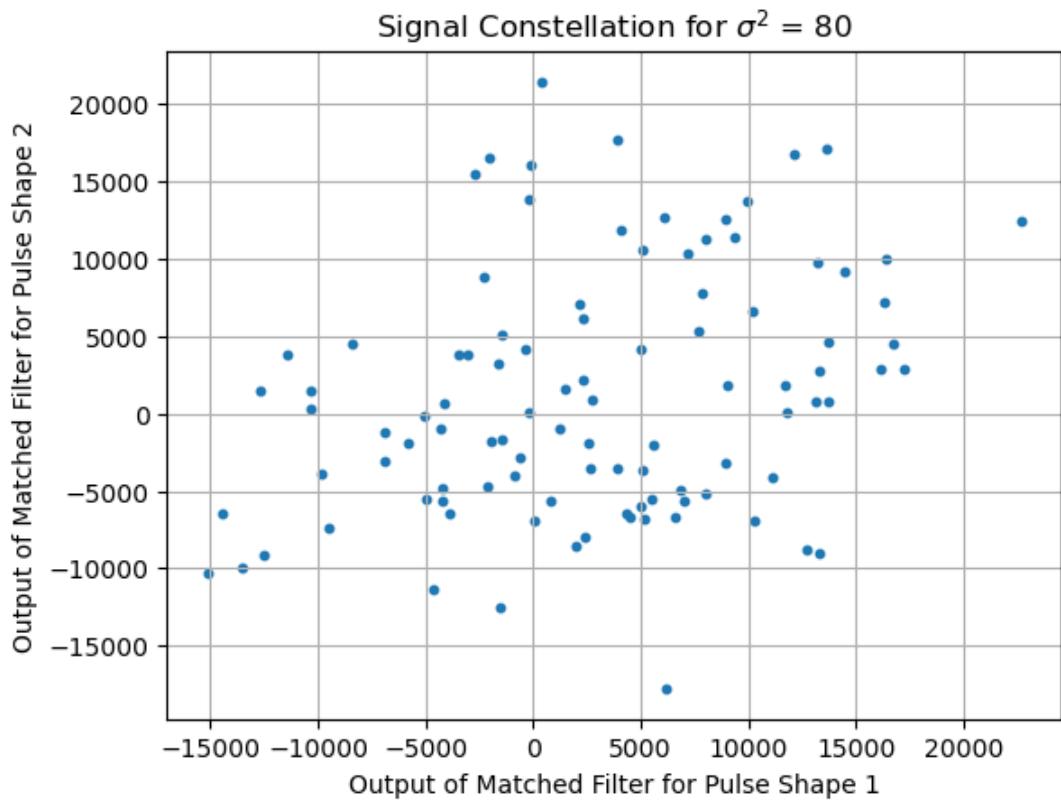
$\sigma^2 = 5$  نویز با واریانس



$\sigma^2 = 10$  نویز با واریانس



نویز با واریانس  $\sigma^2 = 45$



نویز با واریانس  $\sigma^2 = 80$

#### ۴-۳

میبینیم که مدولاسیون FSK از خود مقاومت خوبی نشان داده و منظومه های سیگنال شکل های خوب و متراکمی هستند. البته که بدیهتا در این حالت نیز به ازای واریانس نویز بزرگ، منظومه از تراکم خارج شده و در کل صفحه پخش میشود.

## ۴ انتقال دنباله ای از اعداد ۸ بیتی

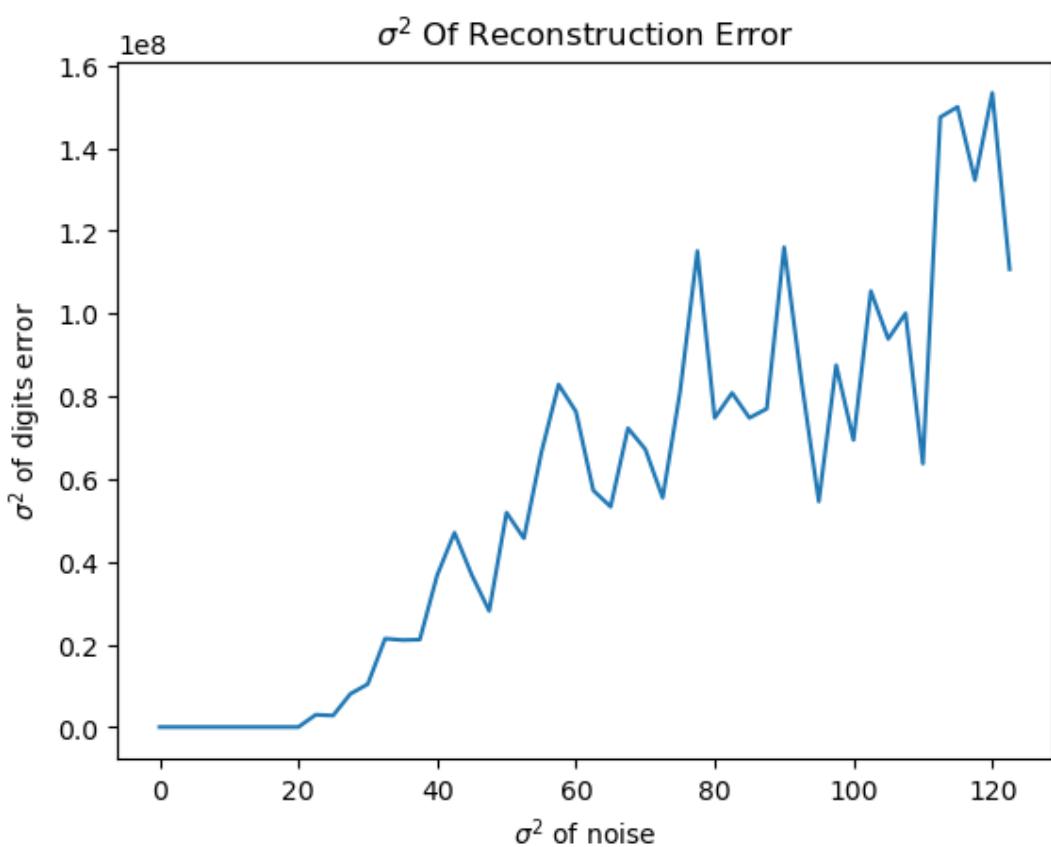
ما در این بخش ابتدا سیگنال هایی با دامنه در بازه  $[0, 255]$  هستند را دریافت می کنیم و این اعداد را به بازنری تبدیل میکنیم و از سیستم مخابراتی تشکیل داده در بخش های قبلی این رشته بازنری را عبور می دهیم و سپس دوباره به حالت دسیمال بر می گردانیم و انتظار داریم که با ورودیمان یکسان باشد.

۱-۴

در این بخش دو تابع خواسته شده‌ی `SourceGenerator` و `OutputDecoder` را به همان گونه که صورت سوال می‌خواست تشکیل می‌دهیم.

۲-۴

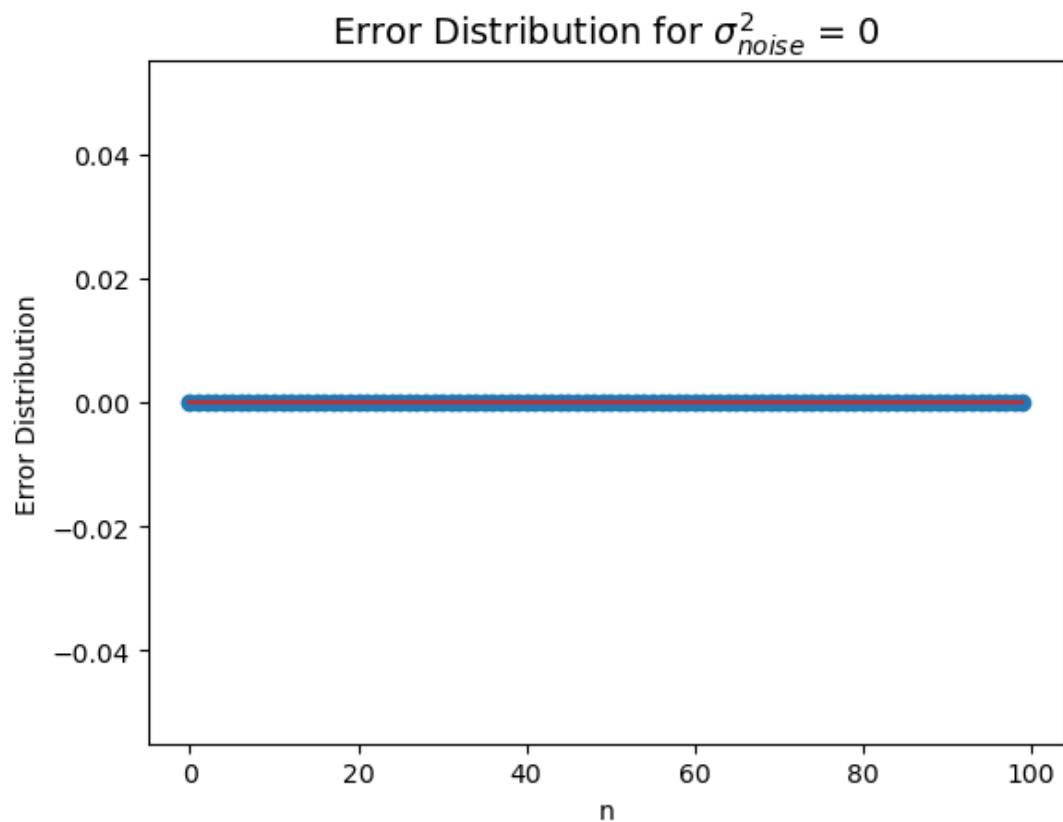
با توجه به صورت سوال، ما خطای بازسازی را برابر با مجدد اخلاف اعداد ورودی و خروجی در نظر می‌گیریم و طبق این تعریف داریم:



### واریانس خطأ بحسب واریانس نویز

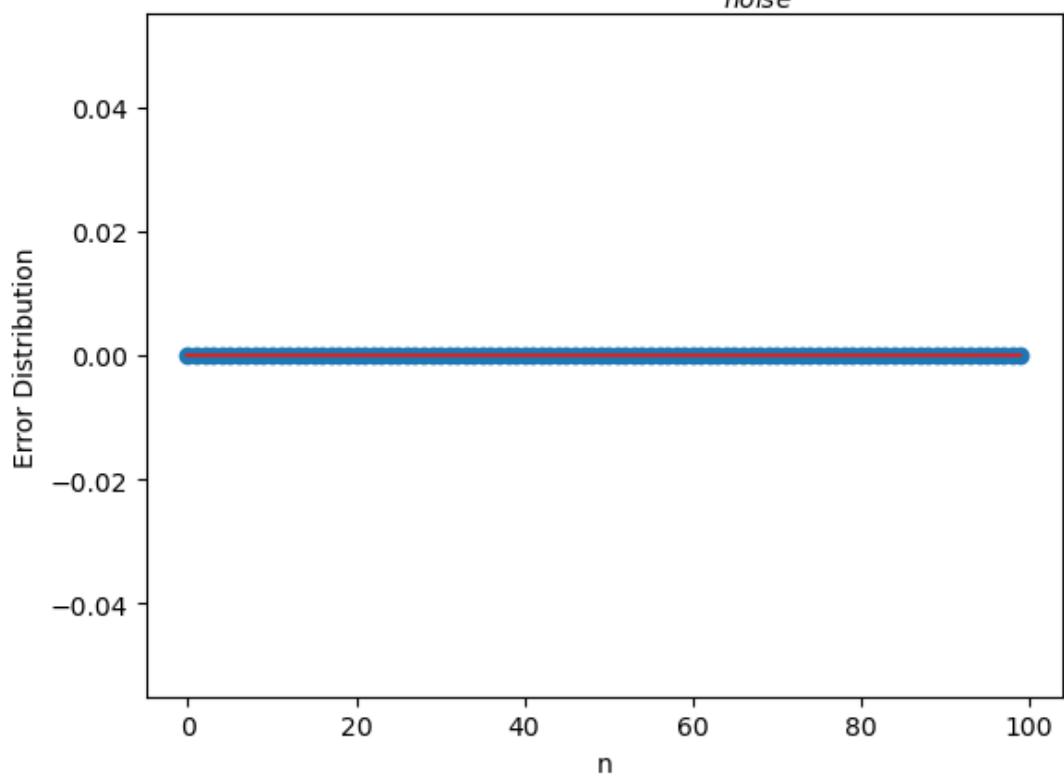
همانطور که حدس می‌زدیم، با افزایش رفته واریانس نویز، تعداد و میزان اختلاف اعداد دسیمال بیشتر می‌شود. البته در واریانس‌های کم و نزدیک به صفر، سیستم ما با دقت ۱۰۰ درصدی می‌تواند سیگنال فرستاه شده را دریافت کند. از طرفی به ازای واریانس‌های بسیار بزرگ باعث می‌شود که تقریباً عدد به صورت رندم بین [۲۵۵،۰] تشکیل می‌شود و همین باعث می‌شود که اختلاف‌های بسیار بزرگی به وجود بیاید.

به ازای واریانس های مختلف خطأ را رسم می کنیم :



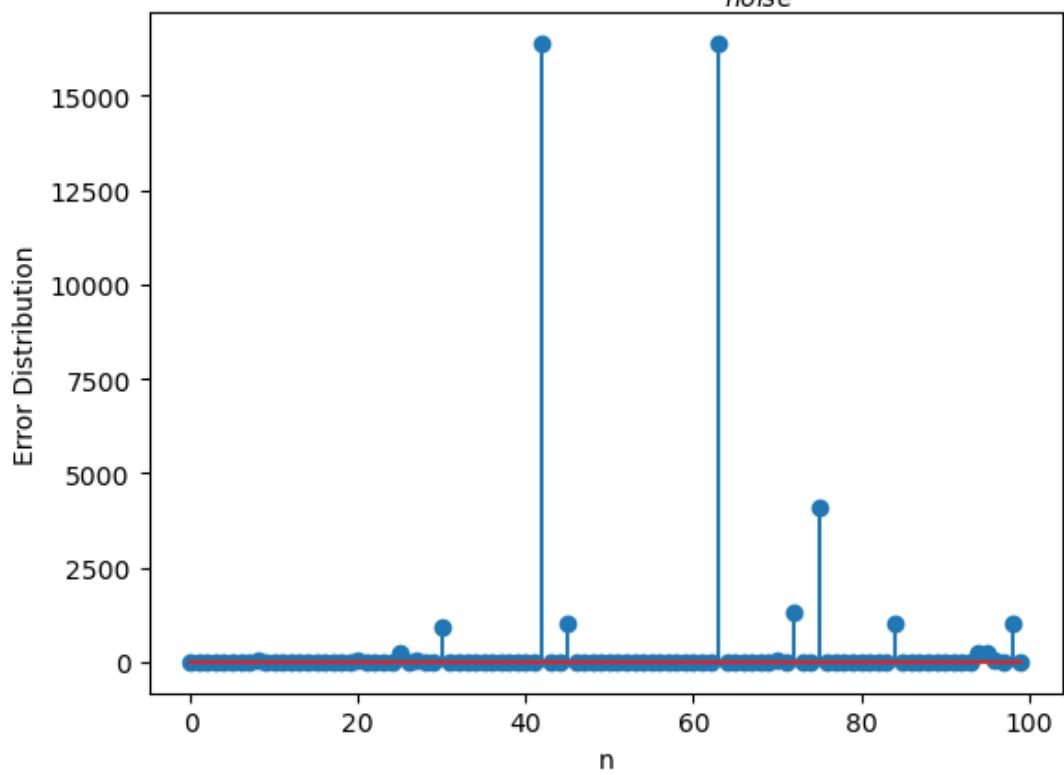
تابع توزیع خطأ برای  $\sigma_{noise}^2 = 0$

Error Distribution for  $\sigma_{noise}^2 = 10$

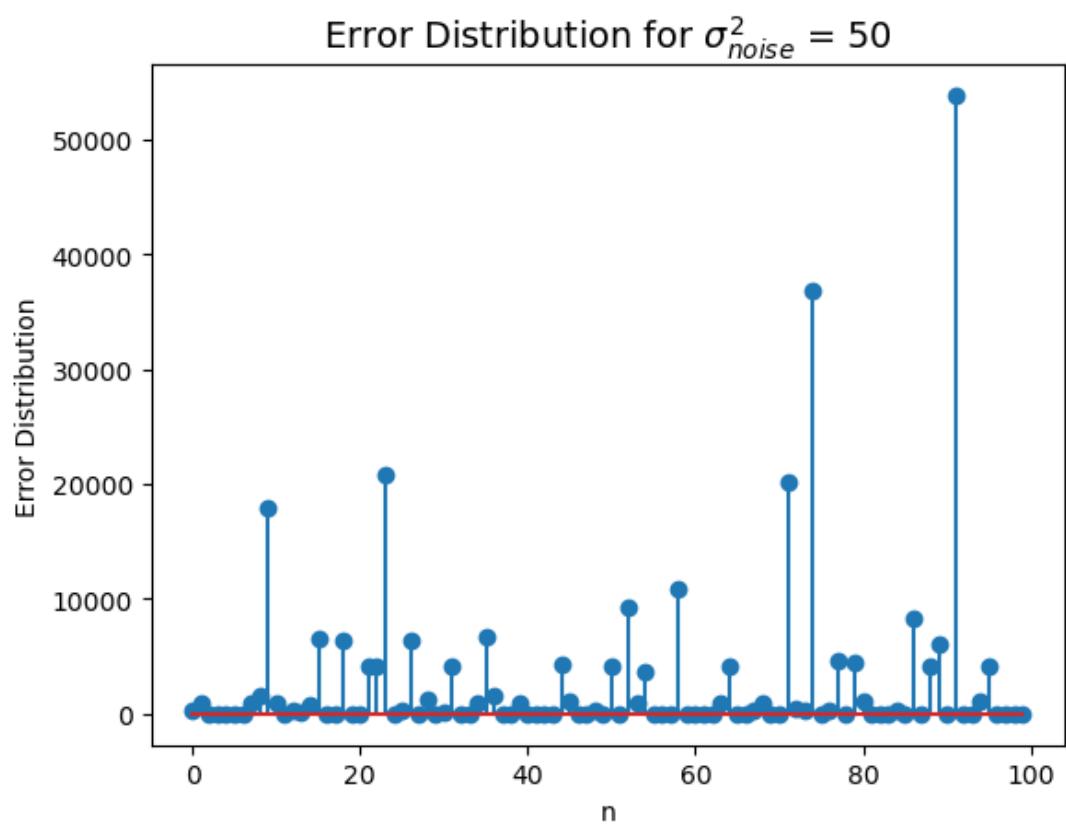


تابع توزیع خطای برای  $\sigma_{noise}^2 = 10$

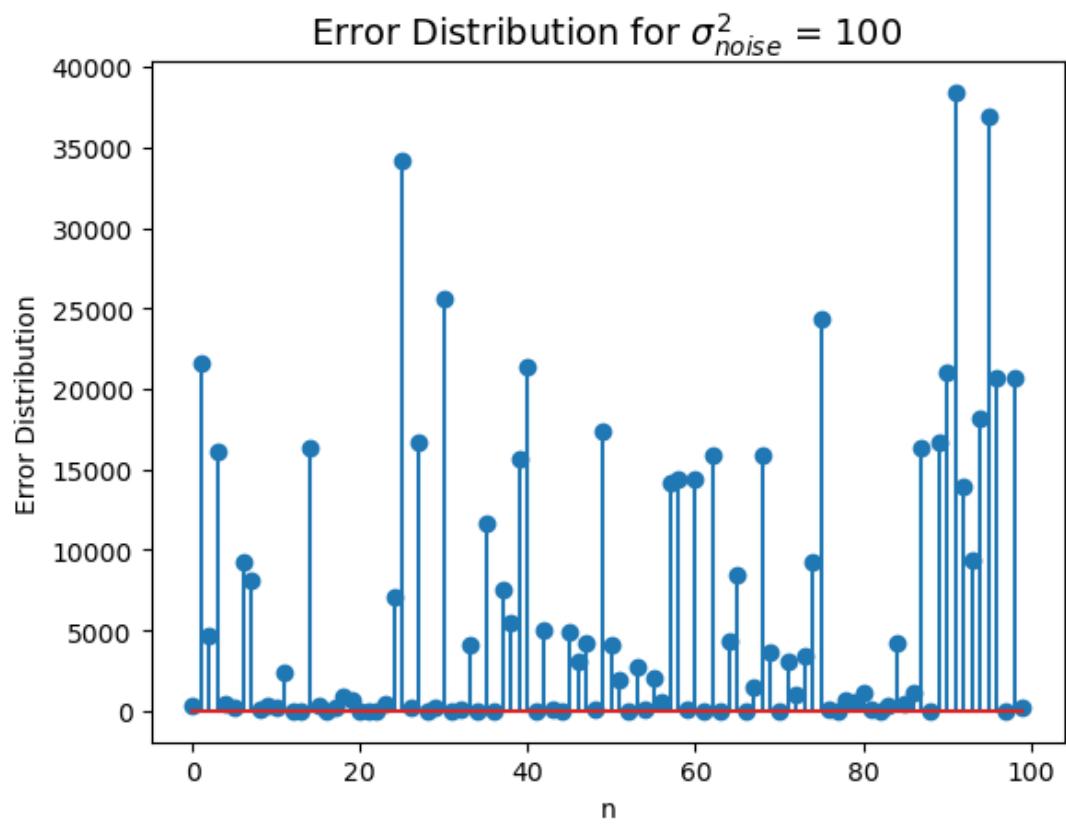
Error Distribution for  $\sigma_{noise}^2 = 25$



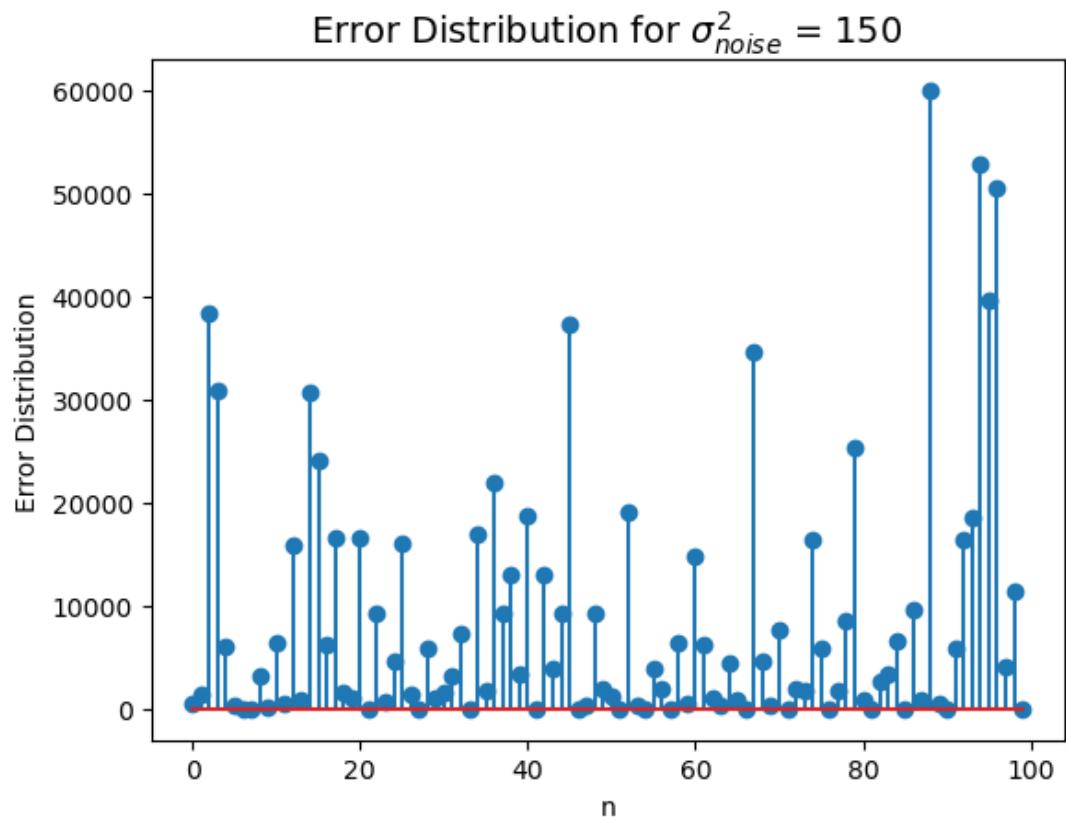
تابع توزیع خطای برای  $\sigma_{noise}^2 = 25$



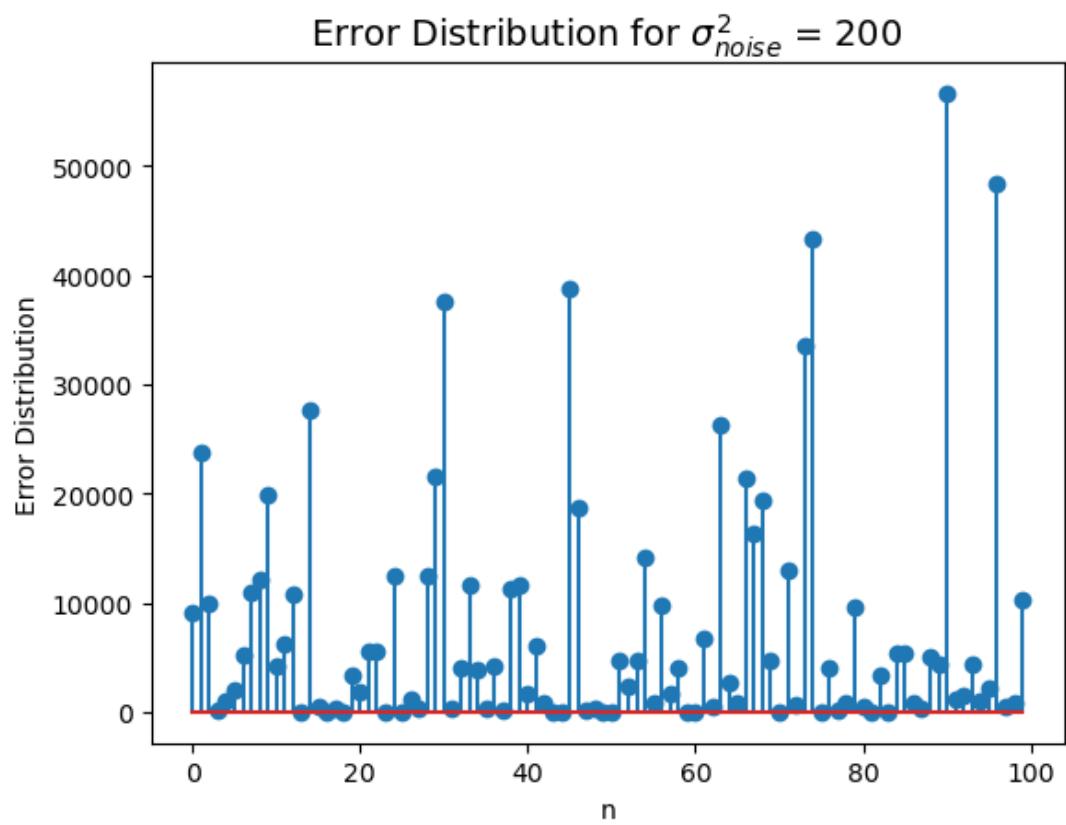
تابع توزیع خطای برای  $\sigma_{noise}^2 = 50^\circ$



تابع توزیع خطای برای  $\sigma_{noise}^2 = 100$

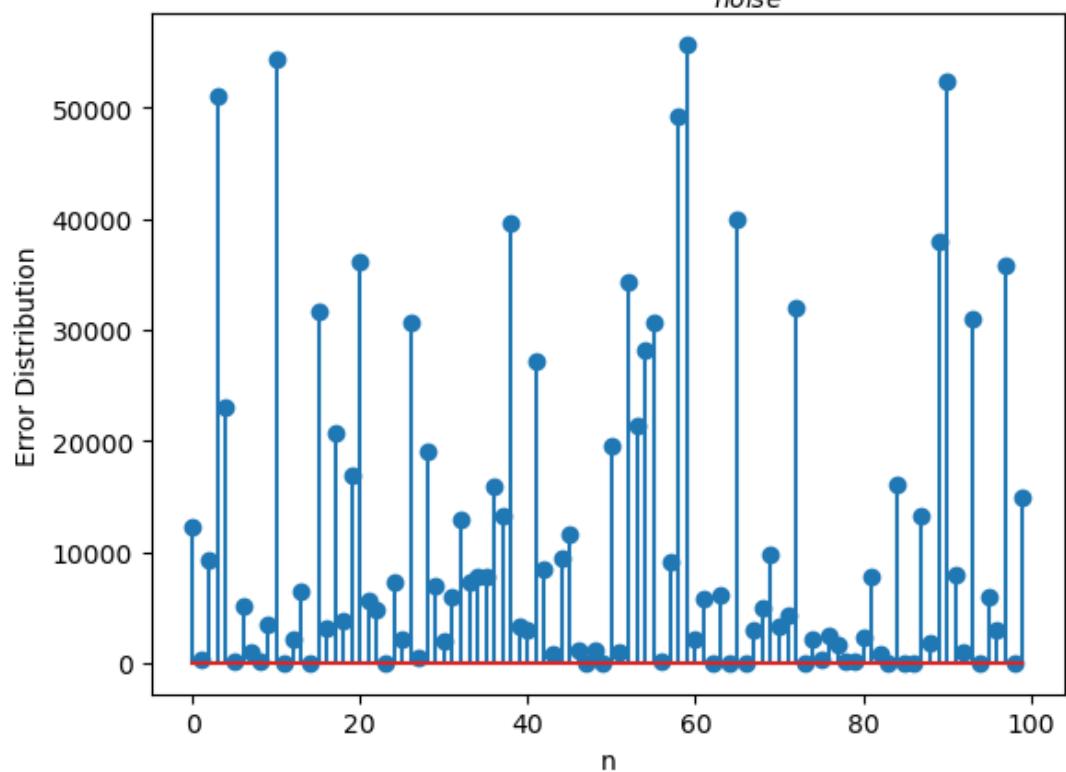


تابع توزیع خطای برای  $\sigma_{noise}^2 = 15^{\circ}$



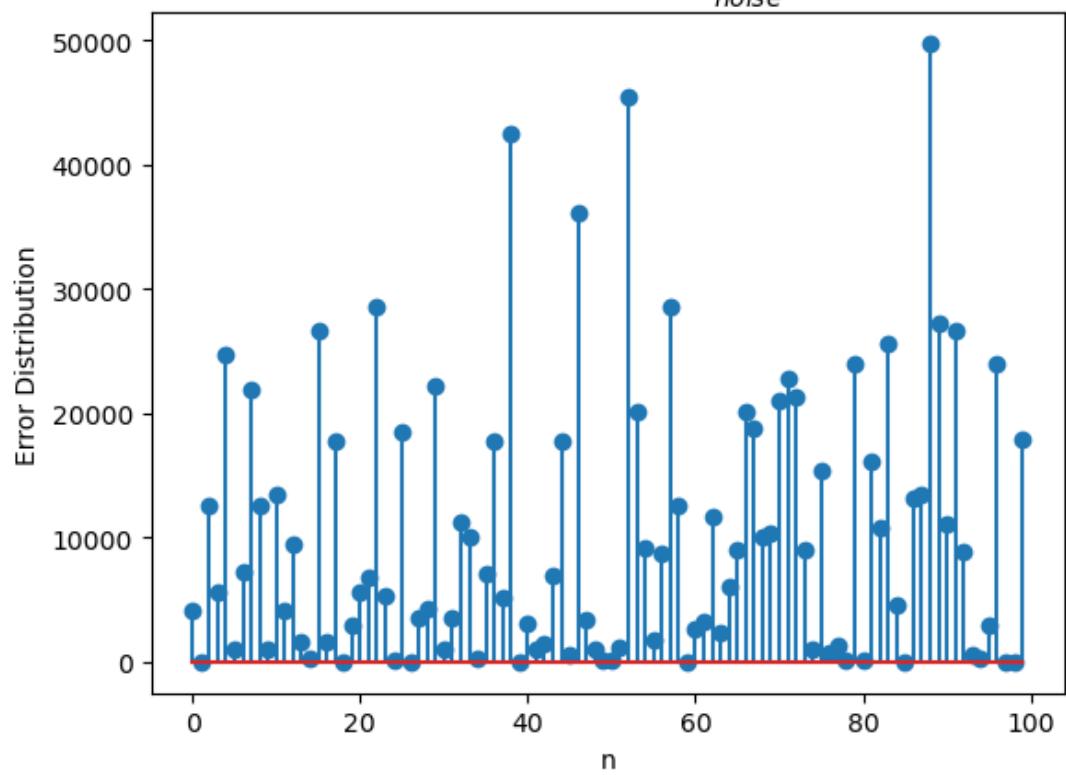
تابع توزیع خطای برای  $\sigma_{noise}^2 = 20^{\circ}$

Error Distribution for  $\sigma_{noise}^2 = 250$



تابع توزیع خطای برای  $\sigma_{noise}^2 = 250$

Error Distribution for  $\sigma_{noise}^2 = 300$



۴۰

$$\sigma_{noise}^2 = 300$$

همانگونه که در بخش قبل نیز گفتیم، در نویز های کم توزیع خطأ صفر است و پیام به درستی انتقال می یابد. اما با افزایش نویز خطأ کم در باز سازی ظاهر میشود تا جایی که برای نویز های بزرگ، عملاً توزیع خطأ را میتوان حاصل تفريق دو عدد رندوم میان  $0$  تا  $250$  دانست. برای همین خود تابع توزیع برای واریانس های بزرگ نیز میتواند مجدد تمام مقادیر این بازه را اختیار کند. پس انتظار داریم هر عددی در بازه  $0$  تا  $255$  را ملاحظه کنیم که همینطور نیز هست.

۴-۴

همانگونه که در بخش های قبل گفتیم، به ازای نویز های بسیار بزرگ عملاً انگار یک عدد رندم به ازای هر عدد ورودی تشکیل می شود که خب این باعث می شود عدد ورودی و خروجی از یکدیگر مستقل باشد و با استفاده از همین نکته میتوانیم بدست بیاوریم که :

$$X_n(x) \sim Uniform(0, 255)$$

$$Y_n(x) \sim Uniform(0, 255)$$

$$X \text{ and } Y \text{ are independent} \implies Z = X - Y \sim \Lambda(-255, 255)$$

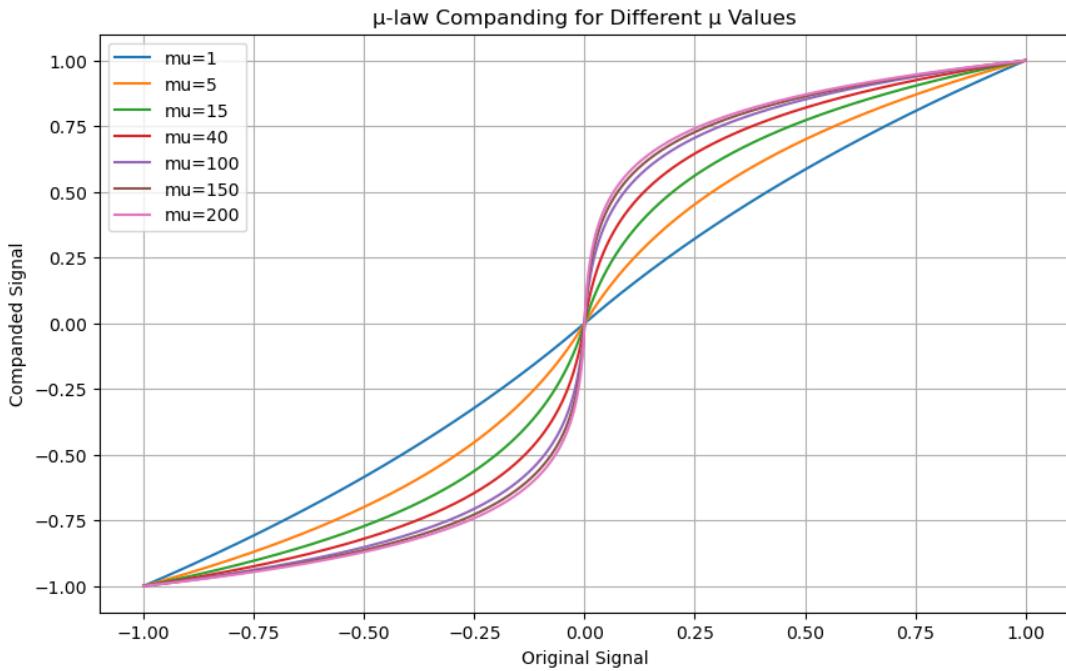
با توجه به توزیع خطأ، واریانس با رفتاری که در نمودار های فوق مشاهده کردیم مطابقت دارد.

## ۵ فشر گستر

ما در این بخش می خواهیم که نتیجه Comandig را بدست آوریم. در این بخش به جای یک کانال مخابراتی، از یک کوانتايزر استفاده خواهیم کرد. کاری که قرار است انجام دهیم این است که ابتدا سیگнал را از یک کانال مخابراتی عبور می دهیم و SNR آن را بدست می آوریم و بعد از آن توسط یک compander برای کاهش اثر اعوجاج خطی استفاده خواهیم کرد.

۱-۵

خروجی تابع بع ازای  $\mu$  های مختلف به صورت زیر است :



### تابع Compander به ازای $\mu$ های مختلف

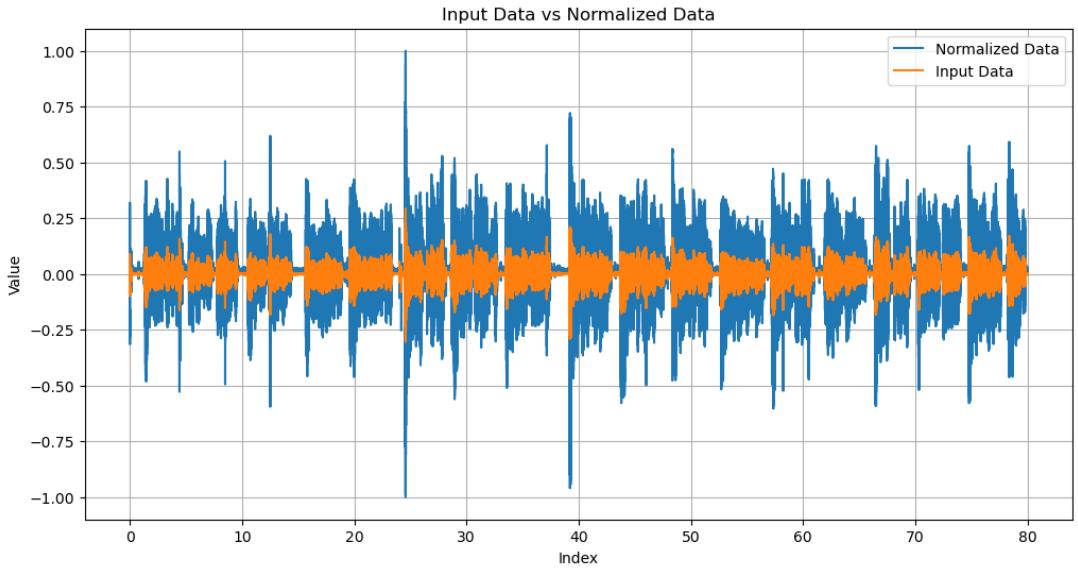
همانگونه که انتظار داشتیم، می بینیم که این تابع به ازای  $\mu$  برابر با صفر، خود ورودی را در خروجی می دهد، اما هرچه رفته  $\mu$  بزرگتر می شود، باعث می شود که بخش های سیگنال که نزدیک تر به صفر هستند، فاصله شون بیشتر شود و عملاً گسترش یابد، ولی بخش های بزرگ تابع به همان اندازه باقی بمانند. این باعث می شود که محدوده‌ی دینامیکی سیگنال صوتی ما کاهش یابد.

۲-۵

در این بخش ما با استفاده از کتابخانه sounddevice صدای خود را ضبط و با استفاده از scipy.io.wavfile آن را ذخیره میکنیم و همچنین آن را میخوانیم.

۳-۵

در این بخش ما سیگنال صدا را نرمالایز می کنیم و در بازه‌ی [۱،۱] قرار می دهیم. برای بدست آوردن توان سیگنال از رابطه امید ریاضی مجازور آن استفاده می کنیم که همانطور که در خروجی های کد مشخص است، مقدار آن  $20 \log_{10} \frac{S}{N}$  است. شکل سیگنال را در تصویر زیر مشاهده می کنید:



سیگنال اصلی و سیگنال نرمالایز شده.

#### ۴-۵

در این بخش با کمک تابع ساخته شده در بخش یک، سیگنال را به ورودی می دهیم و سیگنال compress شده را دریافت می کنیم.

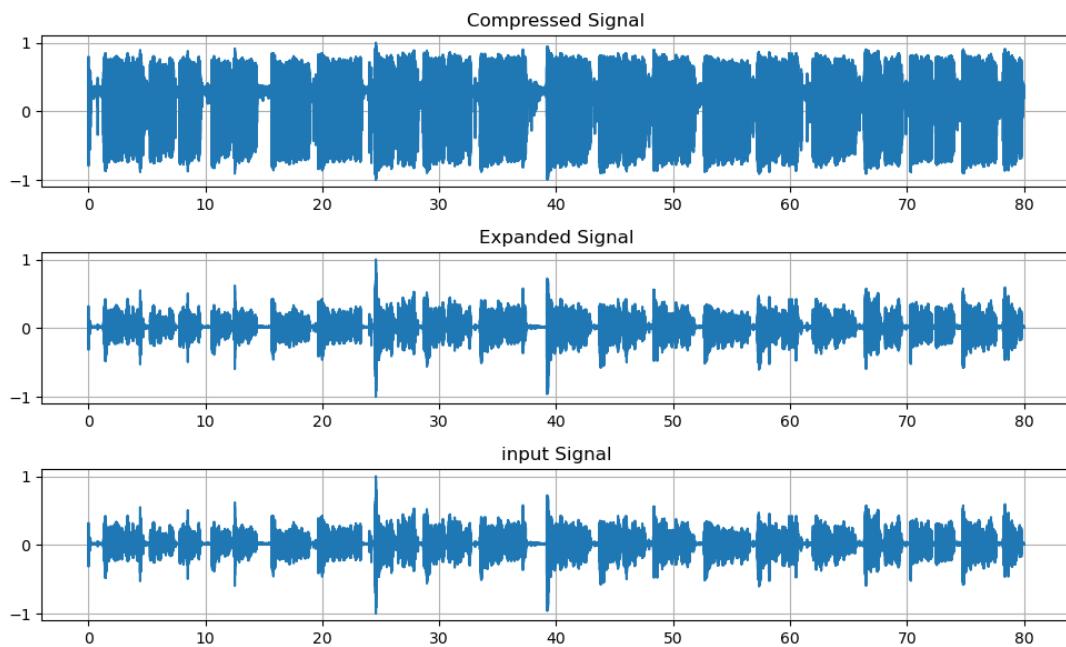
#### ۵-۵

در این بخش سیگنال فشرده شده را به معکوس تابع گفته شده در بخش اول میفرستیم و سیگنال گسترش یافته را به عنوان خروجی دریافت می کنیم. تابع معکوس به این صورت تعریف می شود:

$$F^{-1}(y) = \text{sgn}(y) \frac{(1 + \mu)^{|y|} - 1}{\mu}, \quad -1 \leq y \leq 1.$$

#### ۶-۵

ابتدا سیگنال نرمالایز شده را به تابع compressor می دهیم و سپس خروجی این تابع را به ورودی تابع expander می دهیم. باید سیگنال خروجی این تابع به سیگنال اصلی ما تا حد ممکن نزدیک باشد تا بتوانیم بگوییم تابع های ما به خوبی کار کرده اند و ما به خوبی آن ها را طراحی کرده ایم.



### سیگنال فشرده شده، گسترش یافته، سیگنال اصلی

```

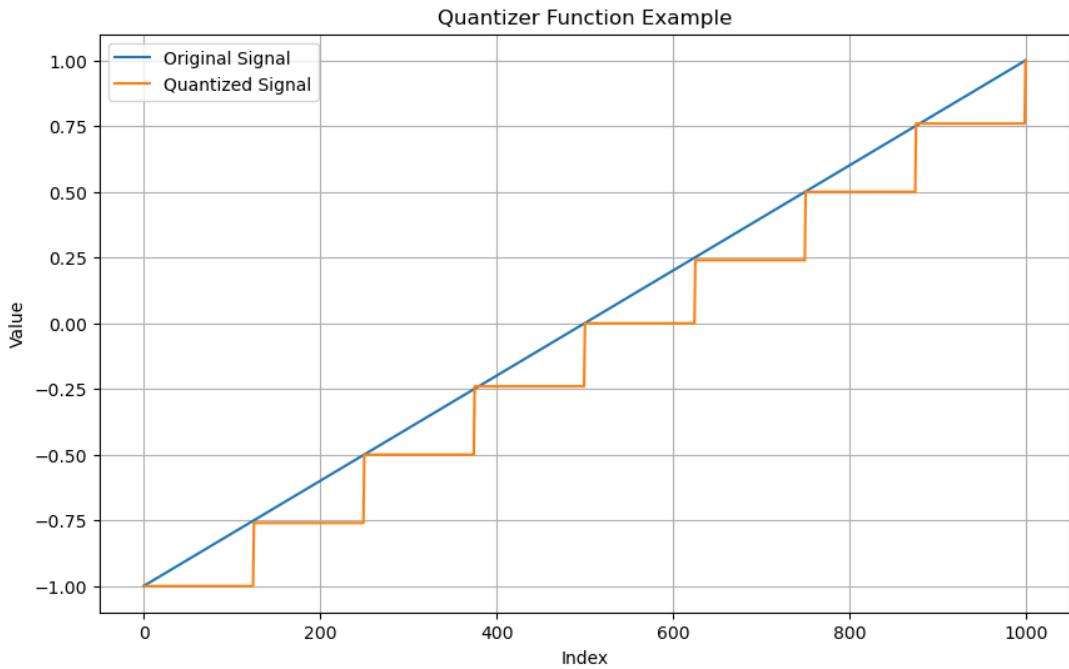
The RMS error of the input and output signals for μ = 1e-05 is: 6.410592722320119e-12
The RMS error of the input and output signals for μ = 1 is: 1.4953261511274048e-18
The RMS error of the input and output signals for μ = 10 is: 1.0848438067093695e-17
The RMS error of the input and output signals for μ = 50 is: 1.6533264755753483e-17
The RMS error of the input and output signals for μ = 150 is: 2.140671832349053e-17
The RMS error of the input and output signals for μ = 200 is: 2.220855417134377e-17
The RMS error of the input and output signals for μ = 255 is: 2.727674142721904e-17
The RMS error of the input and output signals for μ = 600 is: 3.018639231788286e-17
The RMS error of the input and output signals for μ = 1000 is: 3.20749084477865e-17

```

### محاسبه‌ی خطای RMS به ازای مقادیر مختلف $\mu$

۷-۵

در این بخش ما یک تابع quantizer تولید می‌کنیم که در ابتدا سیگنال را میان  $10^0$  نرمالایز می‌کنیم، سپس آن را به صورت گستته کوانتیزه می‌کنیم و سپس سیگنال را در این مرحله‌های کوانتیزه به اعداد هر لول گرد می‌کنیم. سپس دوباره از  $10^0$  به  $-10^0$  اسکیل می‌کنیم. یک نمونه از کارکرد این quantizer در شکل زیر را مشاهده می‌کنید.



### کوانتیزه کردن یک سیگنال دلخواه

۸-۵

ما در این بخش پس از عبور دادن سیگنال مان از کوانتايزر، نسبت سیگنال به نویز خروجی را بدست می آوریم که همانطور که انتظار داشتیم، با افزایش تعداد لول های کوانتیزه کردن، نسبت سیگنال به نویز ما افزایش می یابد.

```
The SNR of the output signal for L = 4 is: -0.3663884852371499 dB
The SNR of the output signal for L = 5 is: 0.22040357194712978 dB
The SNR of the output signal for L = 6 is: 0.8428078432696493 dB
The SNR of the output signal for L = 7 is: 1.074029621505397 dB
The SNR of the output signal for L = 8 is: 1.527499590564409 dB
```

### SQNR سیگنال خروجی به ازای لول های مختلف

۹-۵

در این بخش همان کار بخش قبل را انجام می دهیم، با این تفاوت که در ابتدای آن ابتدا سیگنال را فشرده و بعد از کوانتیزه کردن آن را گستردۀ می کنیم. این کار باعث می شود که میزان خطای ما

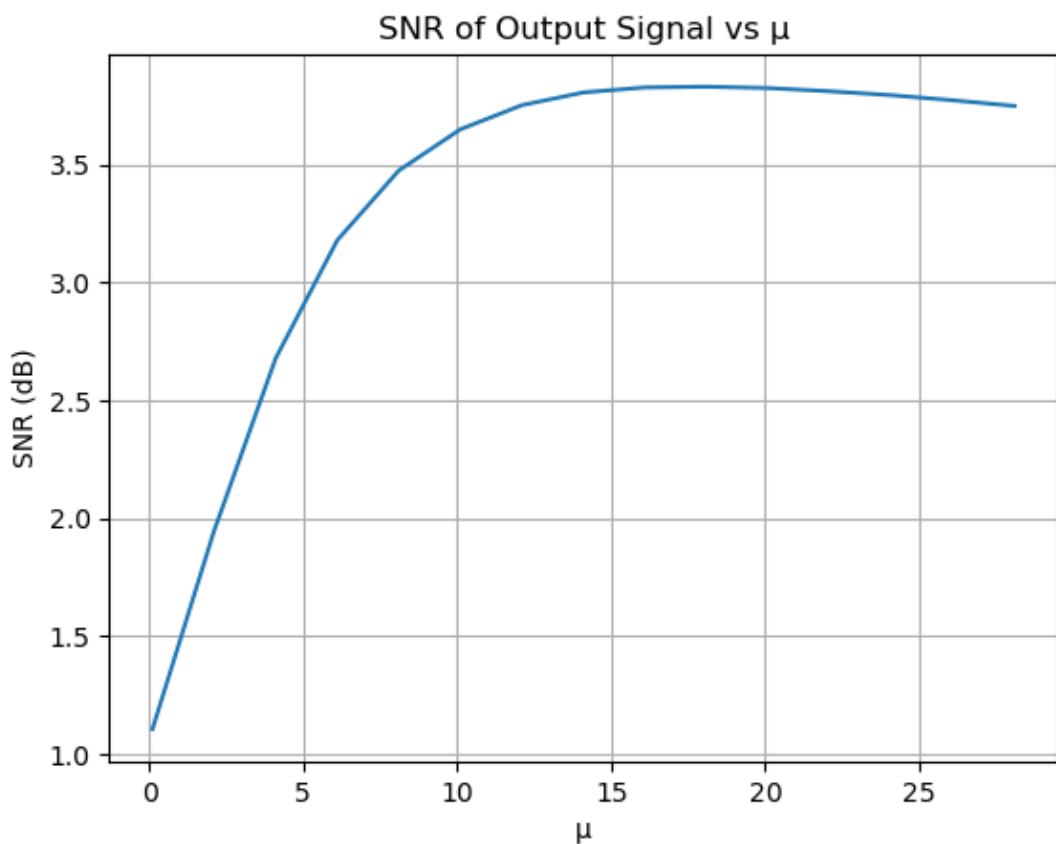
کمتر شود، به علت آنکه اکثر توان و سیگنال ما در بخش نردیک به مرکز قرار دارد و بخش های با دامنه‌ی بزرگتر، بخش کمتر توان سیگنال را تشکیل می‌دهند. همینطور که از نتایج عکس زیر مشاهده می‌کنید، شهود و استدلال‌های ما تماماً درست بوده و به ازای compandig نسبت سیگنال به نویز ما افزایش یافته است.

```
The SNR of the output signal for μ = 10 L = 4 is: 1.8466849813786033 dB
The SNR of the output signal for μ = 10 L = 5 is: 2.186255359717069 dB
The SNR of the output signal for μ = 10 L = 6 is: 3.6418556820476224 dB
The SNR of the output signal for μ = 10 L = 7 is: 3.6808594601021065 dB
The SNR of the output signal for μ = 10 L = 8 is: 5.074098553270056 dB
*****
The SNR of the output signal for μ = 50 L = 4 is: 1.7860386816757343 dB
The SNR of the output signal for μ = 50 L = 5 is: 2.4937016741930766 dB
The SNR of the output signal for μ = 50 L = 6 is: 3.4333987557080876 dB
The SNR of the output signal for μ = 50 L = 7 is: 4.136540159491959 dB
The SNR of the output signal for μ = 50 L = 8 is: 4.831651503556086 dB
*****
The SNR of the output signal for μ = 75 L = 4 is: 1.5962793332987133 dB
The SNR of the output signal for μ = 75 L = 5 is: 2.2969416588834006 dB
The SNR of the output signal for μ = 75 L = 6 is: 3.120662860712275 dB
The SNR of the output signal for μ = 75 L = 7 is: 3.883368265466472 dB
The SNR of the output signal for μ = 75 L = 8 is: 4.548742590904315 dB
*****
The SNR of the output signal for μ = 125 L = 4 is: 1.3774548019672517 dB
The SNR of the output signal for μ = 125 L = 5 is: 2.013748149656604 dB
The SNR of the output signal for μ = 125 L = 6 is: 2.7104017043785182 dB
The SNR of the output signal for μ = 125 L = 7 is: 3.4738071739470184 dB
The SNR of the output signal for μ = 125 L = 8 is: 4.203614183531697 dB
*****
The SNR of the output signal for μ = 255 L = 4 is: 1.1298371263943232 dB
The SNR of the output signal for μ = 255 L = 5 is: 1.6517172970784149 dB
The SNR of the output signal for μ = 255 L = 6 is: 2.2114452946148093 dB
The SNR of the output signal for μ = 255 L = 7 is: 2.89895514909197 dB
The SNR of the output signal for μ = 255 L = 8 is: 3.6315931789155664 dB
*****
```

## SQNR سیگنال خروجی به ازای لول‌ها و $\mu$ ‌های مختلف

۱۰-۵

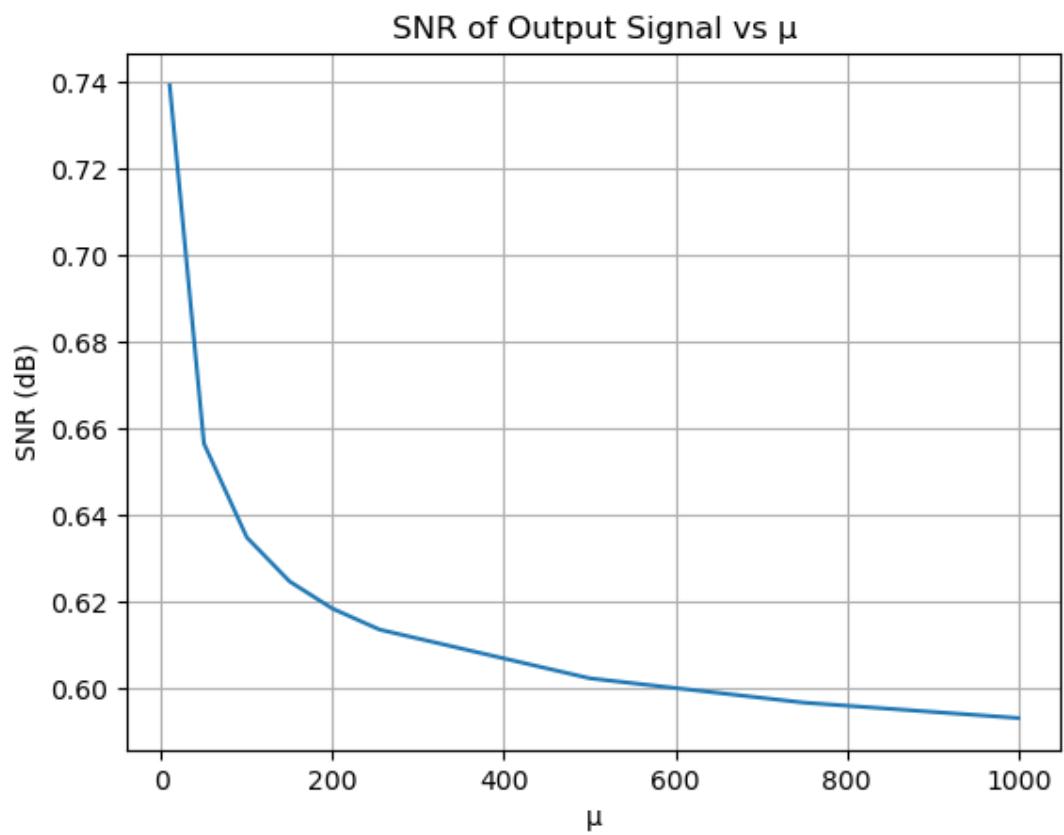
از آنجایی که با افزایش  $\mu$ ، اختلاف میان سطوح پایین تر و بالاتر سیگنال کاهش می‌یابد، و همانطور که در بخش قبل گفتیم، این کار باعث کمتر شدن نویز و در نتیجه بیشتر شدن SNR می‌شود. و همینطور ما در شبیه سازی نیز میتوانیم به این امر پی ببریم که شهود و استدلالی که داریم درست است و با نتایج شبیه سازی تطابق دارد.



SNR سیگنال خروجی به ازای تعداد لول ۶ و  $\mu$  های مختلف

۱۱-۵

در این بخش دقیقاً بر خلاف بخش قبل، ابتدا اختلاف میان سطوح پایین و بالا افزایش می‌یابد و عملاً به جای کاهش اختلاف که باعث کاهش نویز می‌شد، این افزایش اختلاف باعث افزایش نویز می‌شود و به ازای بزرگتر شدن  $\mu$  ها این اختلاف بزرگتر می‌شود و در نتیجه باعث کاهش SNR می‌شود. و همینطور ما در شبیه سازی نیز میتوانیم به این امر پی ببریم که شهود و استدلالی که داریم درست است و با نتایج شبیه سازی تطابق دارد.



SNR سیگنال خروجی به ازای تعداد لول ۶ و  $\mu$  های مختلف وقتی جای compressor و expandor را عوض می کنیم.

## References