

به نام خدا

سوال یک)

تبدیل کسینوسی گستته (DCT)، دنباله‌ای محدود از نقاط داده را به صورت مجموع توابع کسینوسی که در فرکانس‌های متفاوت نوسان می‌کنند، نمایش می‌دهد. این تبدیل استفاده گسترده‌ای در علوم و مهندسی دارد مثل فشرده‌سازی صوت و تصویر.

از آنجایی که توابع کسینوسی کمتری برای تقریب زدن یک سیگنال مورد نیاز است (در مقایسه با توابع سینوسی)، استفاده از تابع کسینوس به جای سینوس در فشرده‌سازی ضروری است. همچنین در معادلات دیفرانسیل، توابع کسینوسی دارای شرایط مرزی مشخص‌تری هستند.

تبدیل کسینوسی گستته (DCT)، شباهت بسیاری به تبدیل فوريه گستته (DFT) دارد، با این تفاوت که حاصل تبدیل فقط مقادیر حقیقی دارد (برخلاف تبدیل فوريه که منجر به مقادیر مختلط می‌شود).

تبدیل کسینوسی گستته اصلاح شده (MDCT) یک تبدیل مبتنی بر تبدیل کسینوس گستته است که دارای ویژگی‌های افزون بر ویژگی‌های تبدیل کسینوسی گستته است.

این تبدیل طراحی شده است تا بر روی بلوک‌های متوالی یک مجموعه داده بزرگتر اجرا شود، جایی که بلوک‌های بعدی با هم همپوشانی دارند به طوری که نیمه آخر یک بلوک با نیمه اول بلوک بعدی منطبق است. این همپوشانی، علاوه بر کیفیت تراکم انرژی DCT، MDCT را به ویژه برای کاربردهای فشرده سازی سیگنال جذاب‌تر می‌کند، زیرا به جلوگیری از مصنوعات ناشی از مرزهای بلوک کمک می‌کند. در نتیجه این ویژگی‌ها، MDCT را پرکاربردترین تکنیک فشرده سازی با اتلاف در فشرده سازی داده‌های صوتی کرده است.

اگر تبدیل کسینوسی گستته را بر روی سیگنال X (یک بردار n تایی) اعمال کنیم، خروجی یک بردار n تایی است.

$$y = Ex$$

$$E_{ij} = \sqrt{\frac{2}{n}} \cos \frac{(i + \frac{1}{2})(j + \frac{1}{2})\pi}{n}$$
 یک ماتریس $n \times n$ است.

حال اگر n عددی صحیح، زوج و مثبت باشد، آنگاه MDCT به شکل زیر در می‌آید.

$$M_{ij} = \sqrt{\frac{2}{n}} \cos \frac{(i + \frac{1}{2})(j + \frac{n}{2} + \frac{1}{2})\pi}{n}$$
 یک ماتریس $n \times 2n$ است.

MDCT به عنوان IMDCT معکوس شناخته می شود. از آنجایی که تعداد ورودی ها و خروجی های این تبدیل متفاوت است، در نگاه اول ممکن است به نظر برسد که MDCT نباید معکوس داشته باشد. اما برگشت پذیری کامل با افزودن IMDCT های همپوشانی بلوک های همپوشانی بعدی حاصل می شود که باعث لغو خطاهای بازیابی داده های اصلی می شود.

اگرچه استفاده مستقیم از فرمول MDCT به محاسبات زیاد نیاز دارد، اما می توان همان چیز را تنها با فاکتور گیری بازگشتنی محاسبه کرد، مانند تبدیل فوریه سریع (FFT).

سوال دو)

در ابتدا به تولید pure tone می پردازیم. برای تولید این صوت ابتدا فرکانس نمونه برداری را تعیین کرده و سپس زمان را مقداردهی میکنیم. در آخر با استفاده از تابع \cos به ازای فرکانس های مختلف(مثلا ۱۲۸ و ۲۵۶ و ...) این صوت را به طول یک ثانیه می سازیم و با دستور sound آن را پخش می کنیم.

پس ایجاد صوت، ملزومات خود مثل سایز بلوک، اطلاعات کوانتیزه شدن و طول صوت را تعیین می کنیم. سپس به اندازه ی سایز بلوک کپی از اول و آخر صوت را به آن اضافه کرده تا اولین و آخرین بلوک قابل بازیابی شوند.

بعد از آن به ایجاد ماتریس MDCT می پردازیم. با توجه به توضیحات سوال قبل و اینکه سایز بلوک مثبت، صحیح و زوج است، از ماتریس M استفاده می کنیم. با تشکیل این ماتریس، ماتریس های u_1, u_2, u_3, u_4 که 32×32 هستند و به طبع آن 7×7 ، طبق رابطه $v = M \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ and $v = M \begin{bmatrix} u_2 \\ u_3 \end{bmatrix}$ روبرو تعریف می شوند.

سپس با transpose کردن آن inverse MDCT را نیز تشکیل می دهیم. اکنون نوبت به تعیین درایه های سیگنال بازسازی می رسد. ابتدا در حلقه ی نوشته شده دو بلوک کنار هم متناظر را از سیگنال را جدا کرده و transpose آن را در block ذخیره می کنیم. سپس با ضرب آن بلوک 7×7 تایی در ماتریس MDCT متغیر z را مقداردهی میکنیم و عمل کوانتیزه کردن را انجام می دهیم. سپس رند این متغیر به متغیر q را در z ذخیره کرده و y_{bar} که حاصلضرب این z در q است را محاسبه میکنیم و عکس عمل کوانتیزه کردن را انجام می دهیم. در واقع با ضرب وارون MDCT در 7×7 ماتریس های W_1, W_2, W_3, W_4 طبق رابطه $v = N \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ and $v = N \begin{bmatrix} w_3 \\ w_4 \end{bmatrix}$ را بازسازی کرد.

$$w_2 = \frac{1}{2}(w_2 + w_3)$$

به همین شکل W_3, W_4 را ایجاد کرده و صوت بازسازی شده را ایجاد میکنیم.

MDCT به بلوک های مجاور اعمال می شود تا سیگنال تبدیل شده $(v_1, v_2, \dots, v_{m-1})$ بدست آید. این v_i ها جزء فرکانس ها هستند، بنابراین می توانیم فرکانس های خاصی را حفظ کنیم و بر فرکانس های دیگر تأکید نکنیم.

پس از کوچک کردن محتوای v_i توسط کوانتیزه کردن یا ابزارهای دیگر $(u_1, u_2, \dots, u_{m-1})$ می توان از حالت فشرده خارج کرد. البته با الگوریتم padding که در کد اعمال شده است، می توان u_m را نیز بازسازی کرد.

اکنون با استفاده از دستور pause دو ثانیه تاخیر ایجاد کرده و صوت بازسازی شده را پخش می کنیم. دلیل استفاده از این تاخیر پخش نشدن همزمان دو صدا است.

با شنیدن این دو صوت متوجه می شویم صوت اصلی و بازسازی شده هم از نظر کمی و هم از نظر کیفی شباهت بسیاری به یکدیگر دارند.

با استفاده از دستور audiowrite چند puretone که آزمایش کردیم را ذخیره کردم.

برای محاسبه $RMSE$ طبق رابطه $\sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$ زیر عمل کرده و حاصل را با دستور display نمایش می دهیم.

$$RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

سپس به رسم سیگنال اصلی در حوزه زمان و فرکانس می پردازیم. برای رسم در حوزه زمان ابتدا با تقسیم طول سیگنال به فرکانس نمونه برداری، زمان کل را محاسبه کرده و τ را از صفر تا زمان کل با گام های معکوس فرکانس نمونه برداری مقداردهی می کنیم و در آخر با دستورات figure, plot, subplot خروجی ها را رسم می کنیم.

برای رسم در حوزه فرکانس W را بین $-Fs/2$ تا $Fs/2$ با گام های معکوس زمان کل مقداردهی کرده و با استفاده از دستورات fft, fftshift سیگنال ها را به حوزه فرکانس منتقل کرده و اندازه آنها را رسم میکنیم.

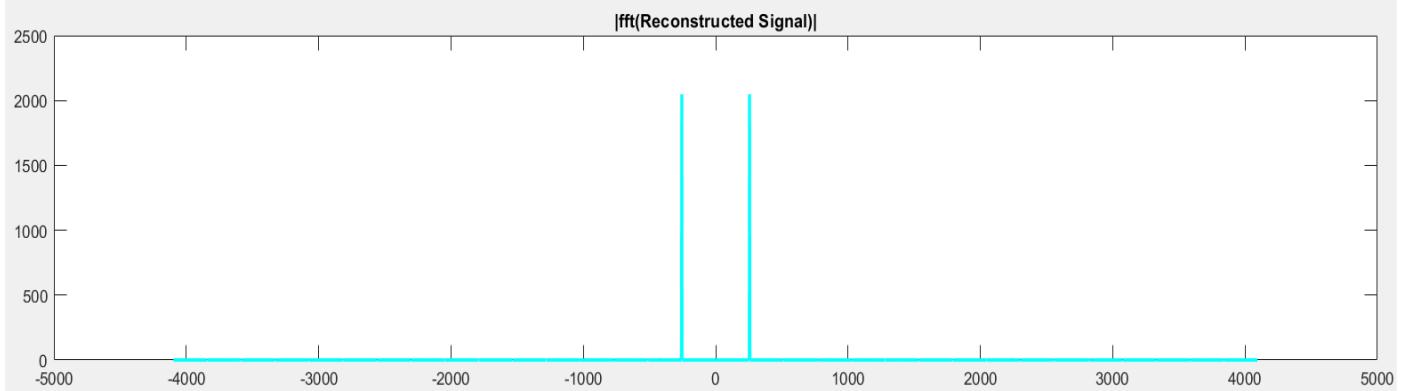
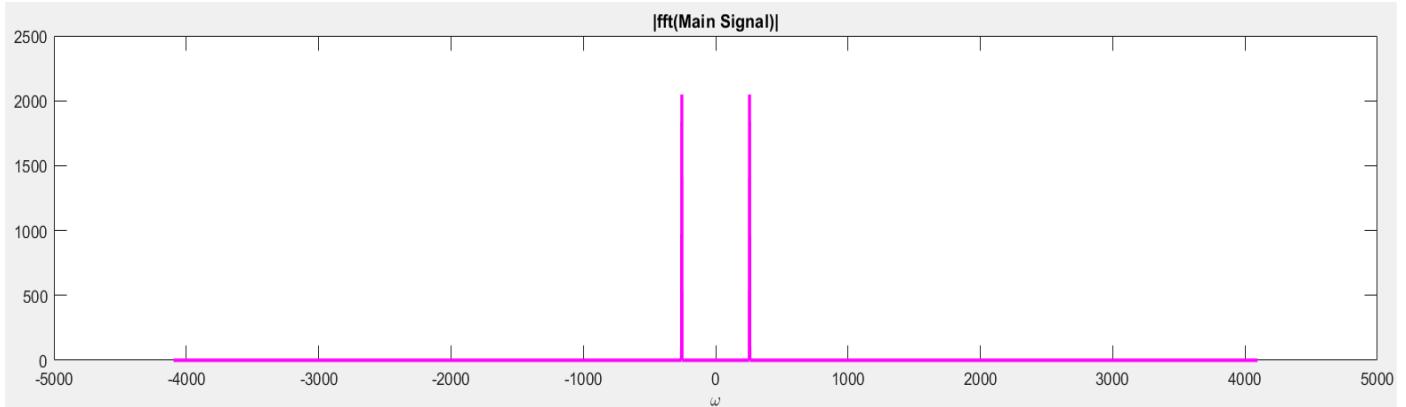
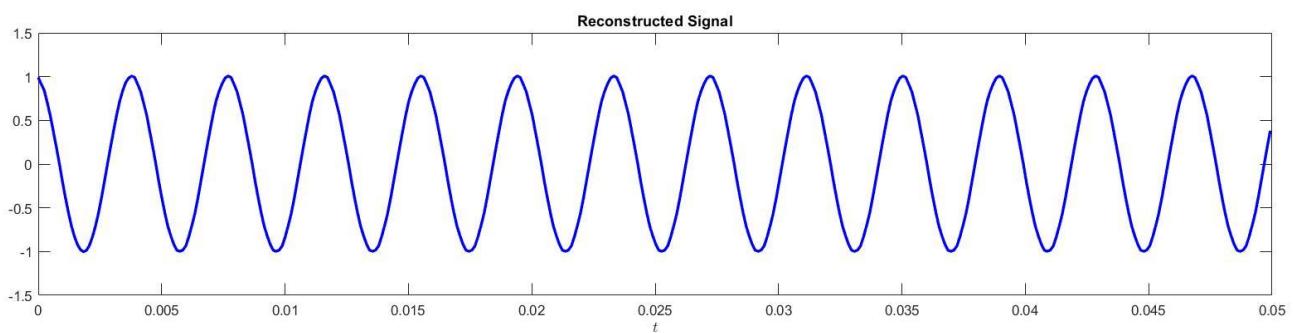
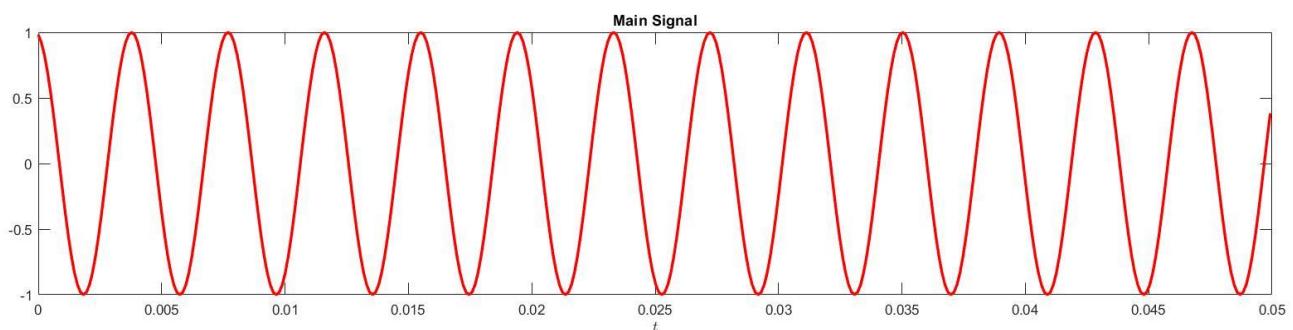
در صفحات بعد نمودار های مربوط به چهار pure tone آزمایش شده را مشاهده می کنید.

ابتدا ۴ آزمایش به ترتیب با فرکانس های ۱۲۸، ۲۵۶، ۵۱۲، و ۱۰۲۴ با تعداد بیت های کوانتیزاسیون ۸ انجام می دهیم. سپس به ترتیب سه آزمایش با فرکانس ۲۵۶ و تعداد بیت های کوانتیزاسیون ۴، ۸، و ۱۲ انجام می دهیم. آزمایش نمودار RMSE بر حسب فرکانس هم با تعداد بیت های کوانتیزاسیون ۸ انجام شده است.

RMSE =

0.0062

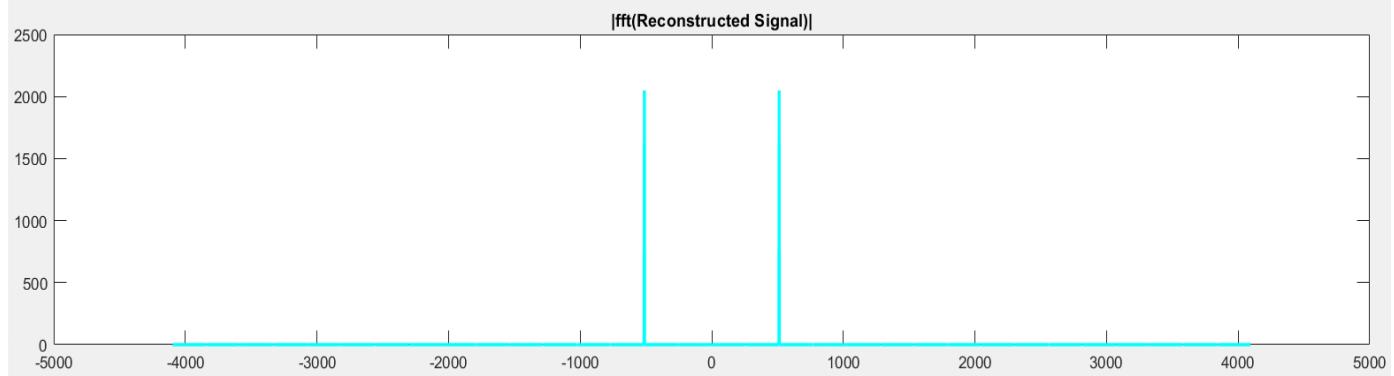
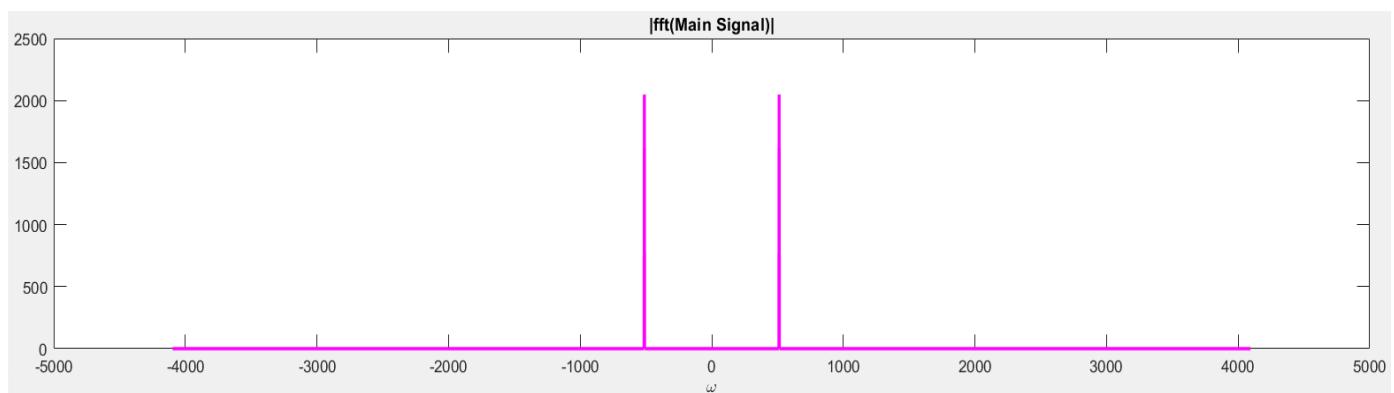
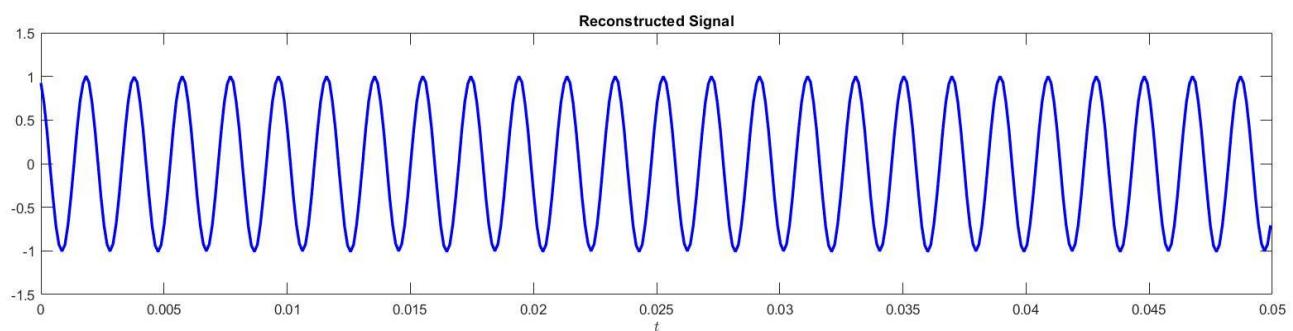
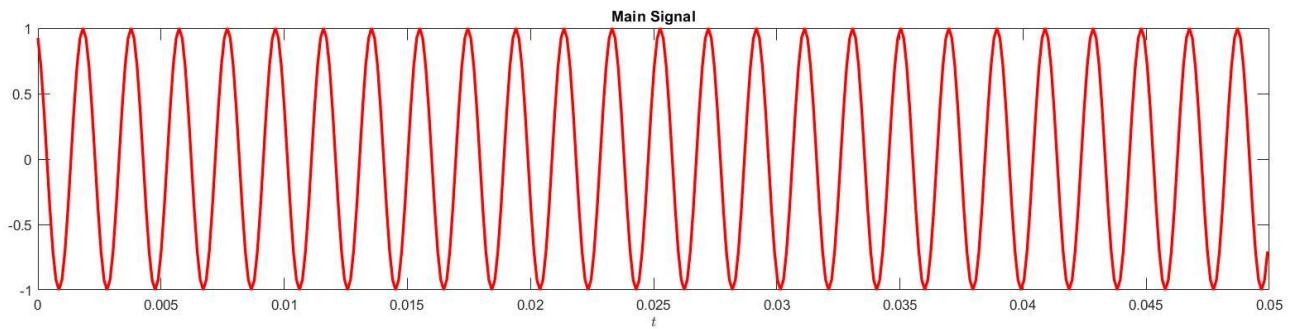
Pure tone 1



RMSE =

0.0058

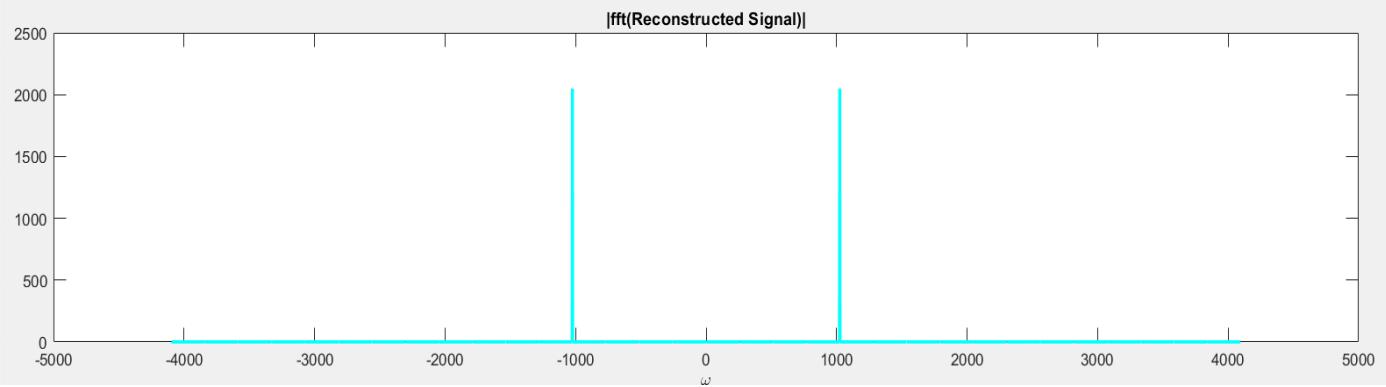
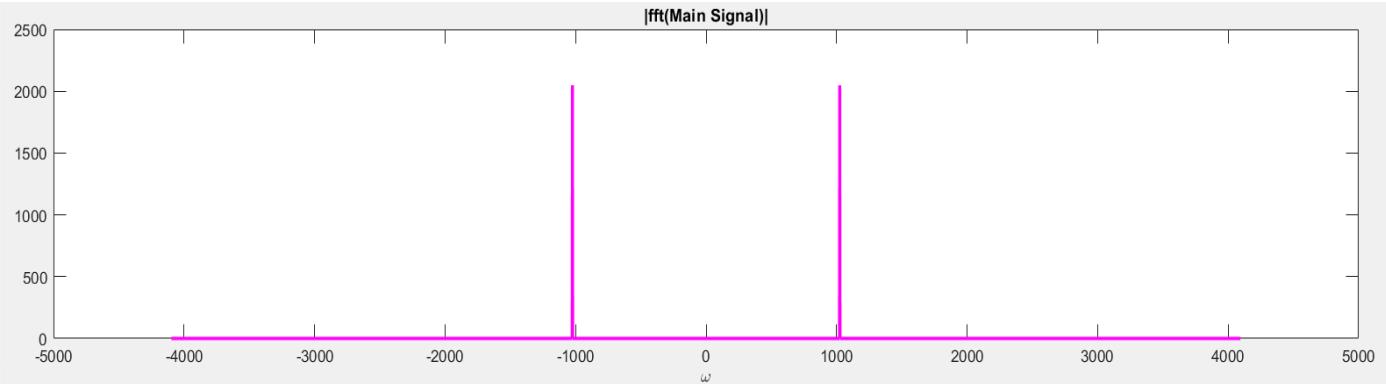
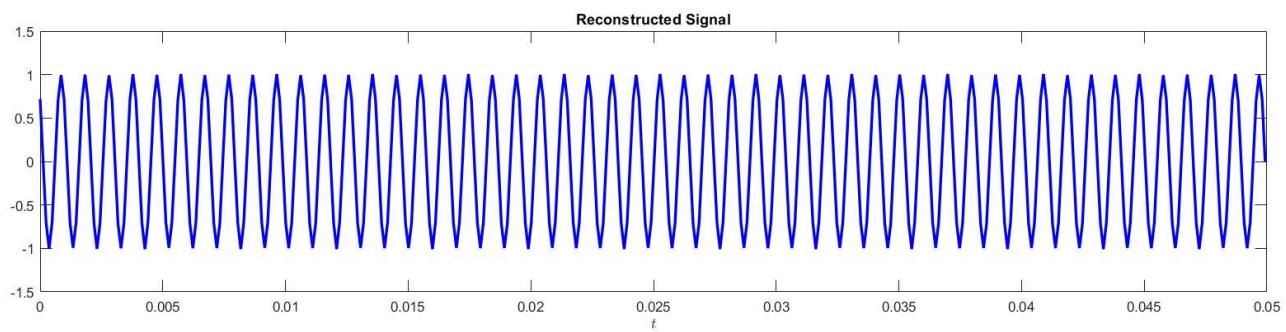
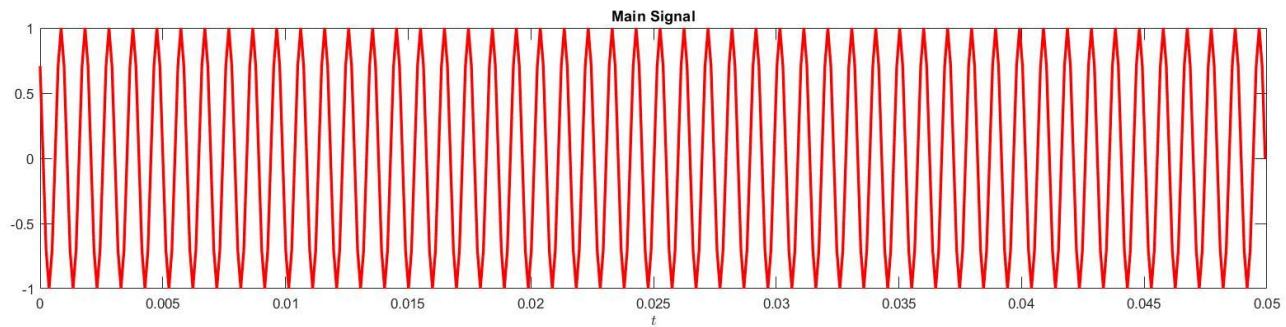
Pure tone ↪



RMSE =

0.0065

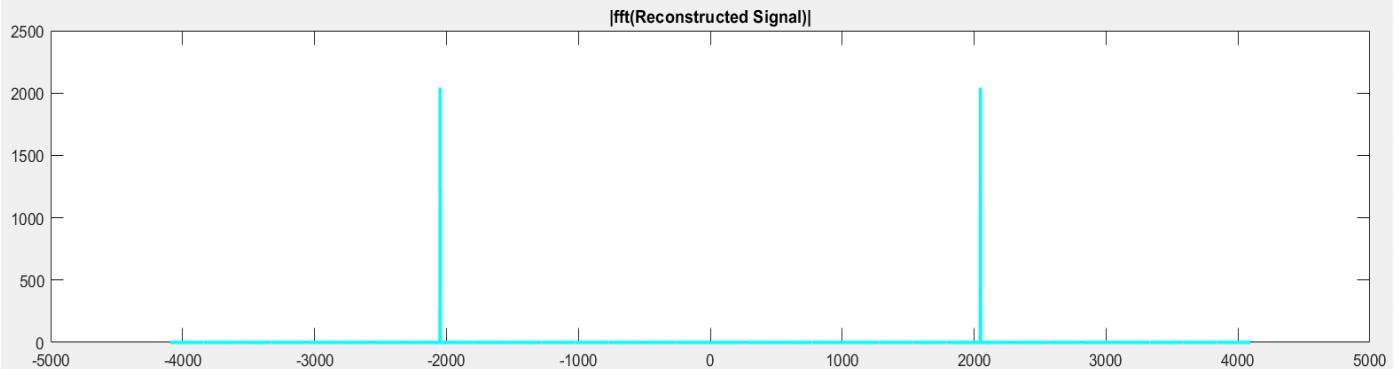
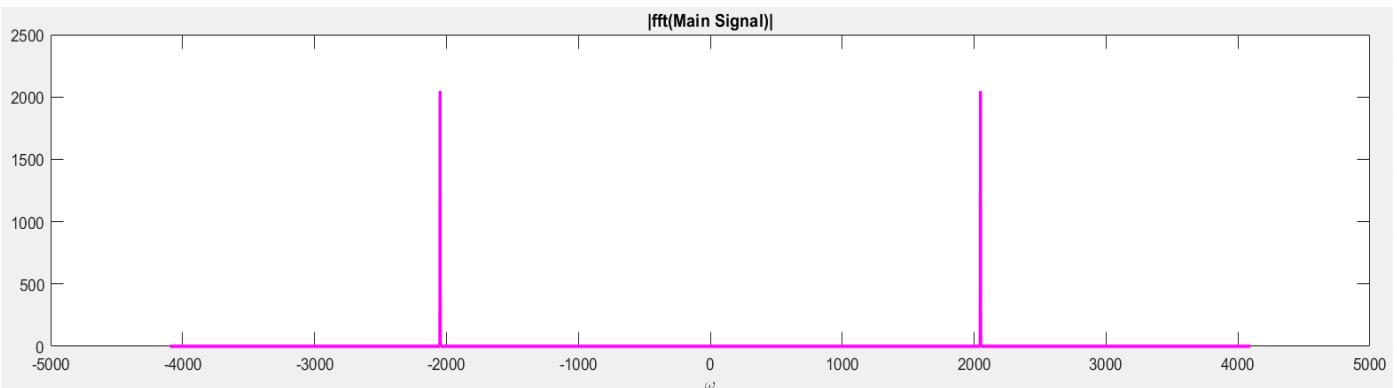
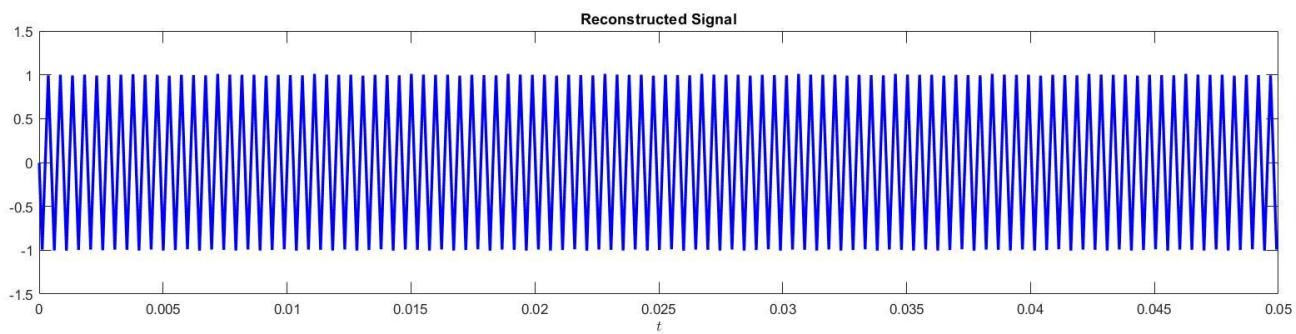
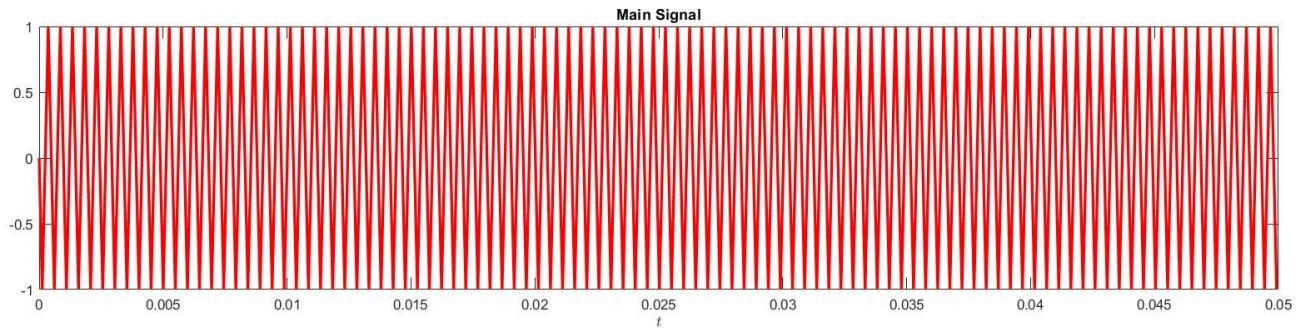
Pure tone ↪



RMSE =

0.0058

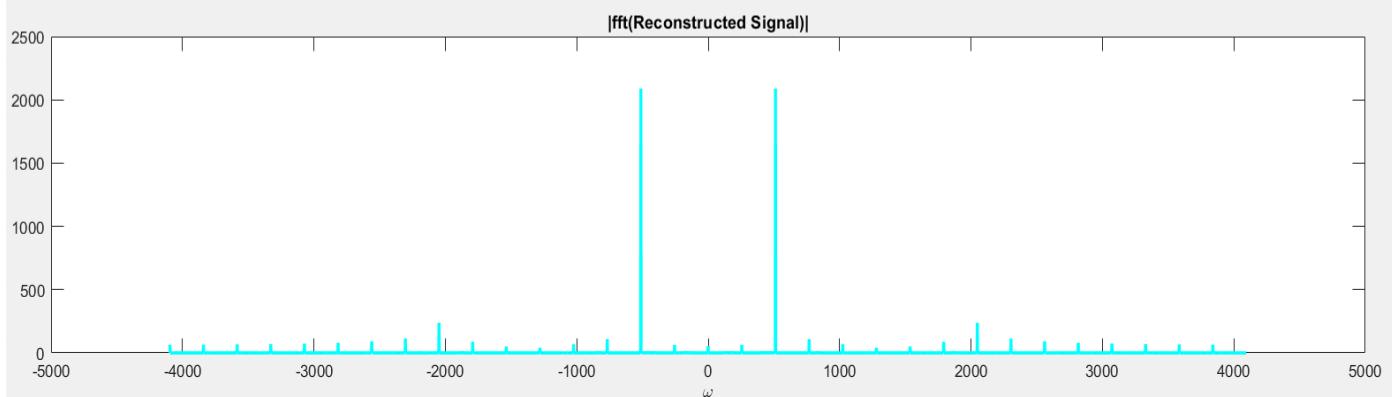
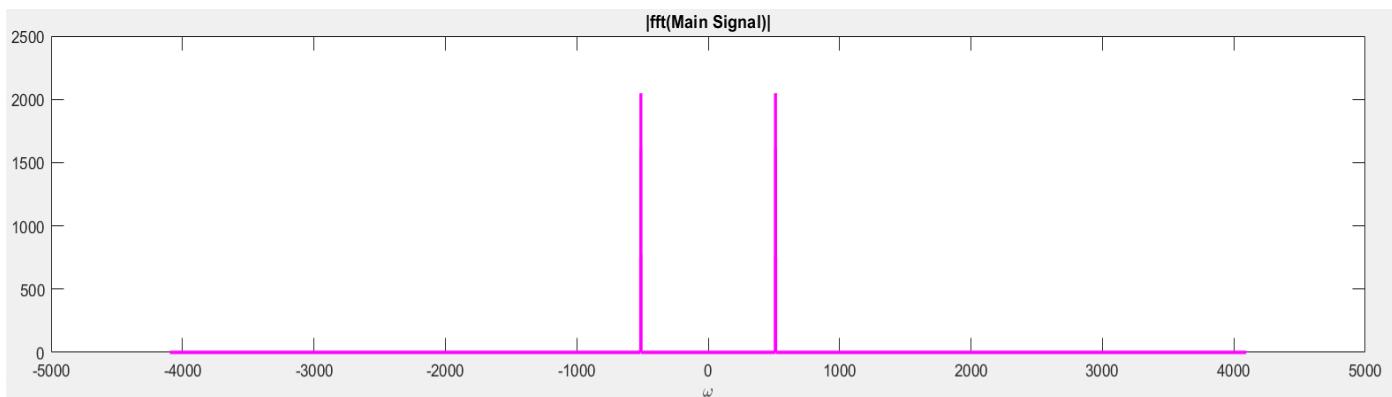
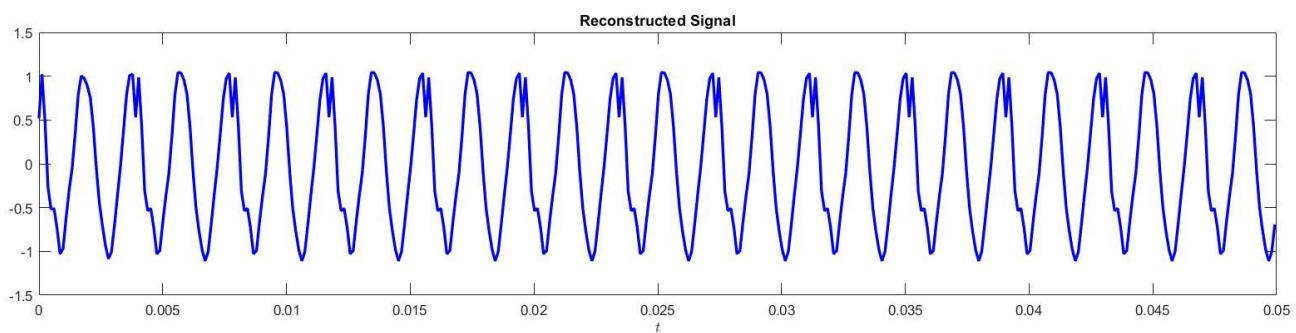
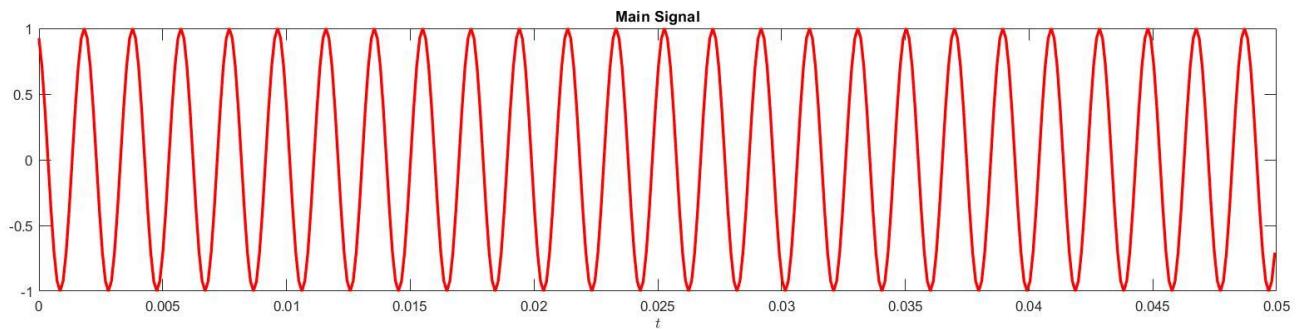
Pure tone ξ



RMSE =

$$b = \xi$$

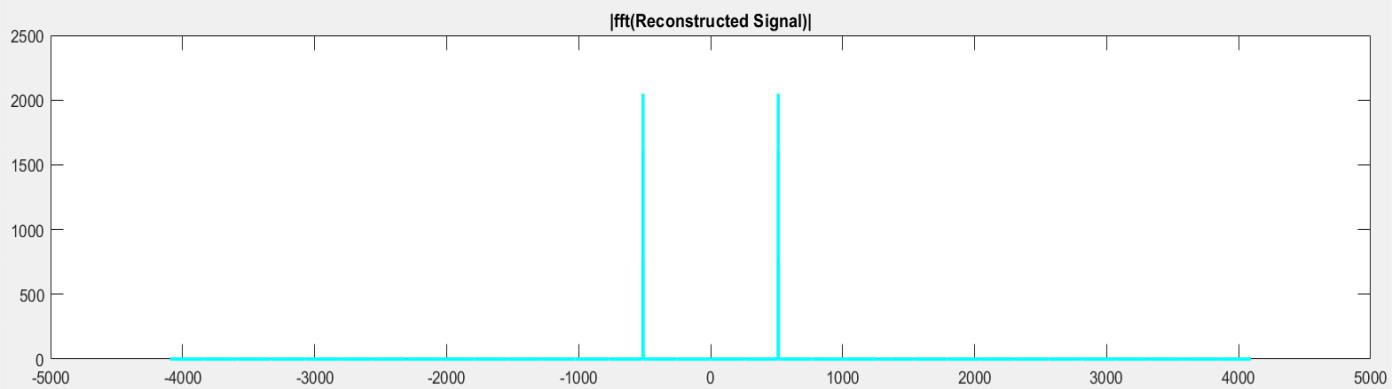
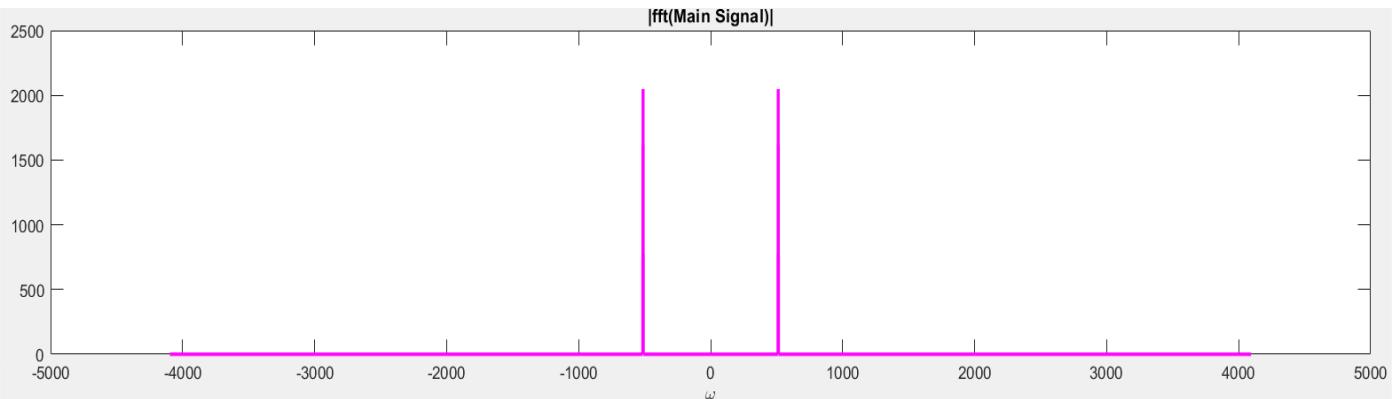
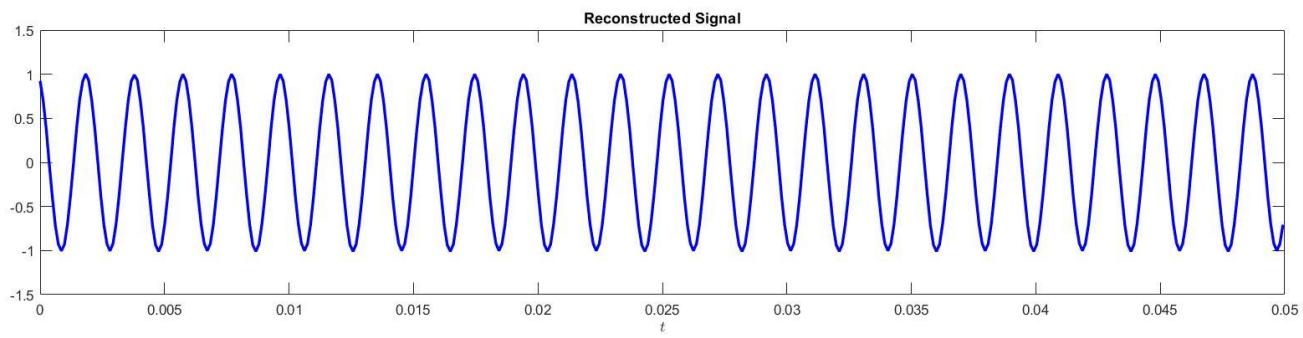
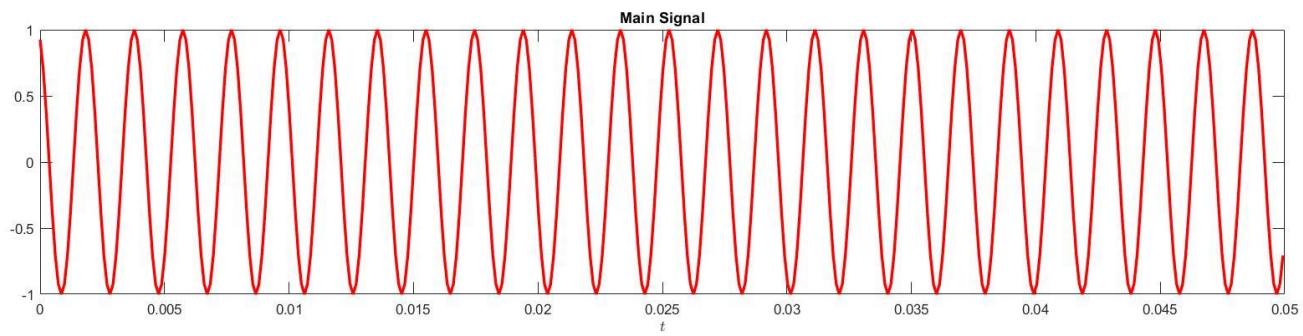
0.1311



RMSE =

0.0058

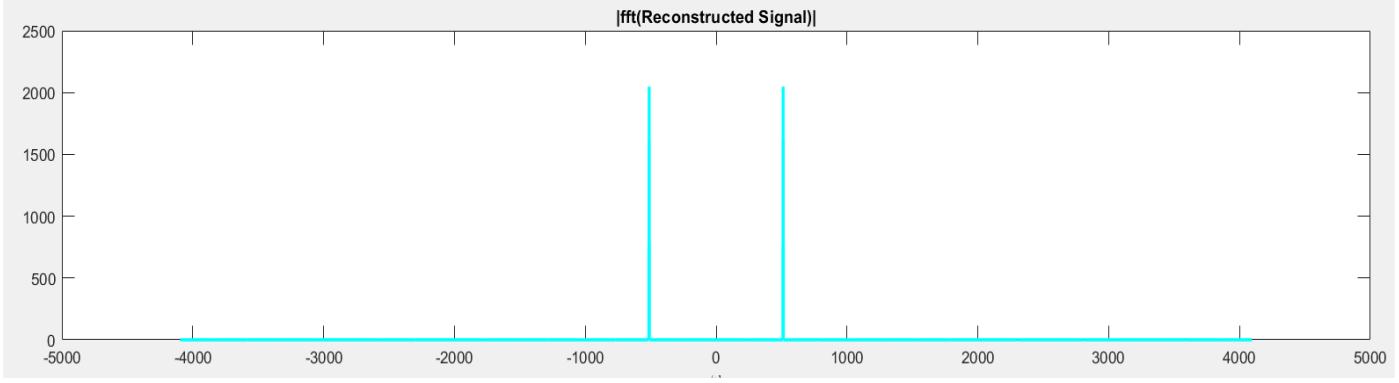
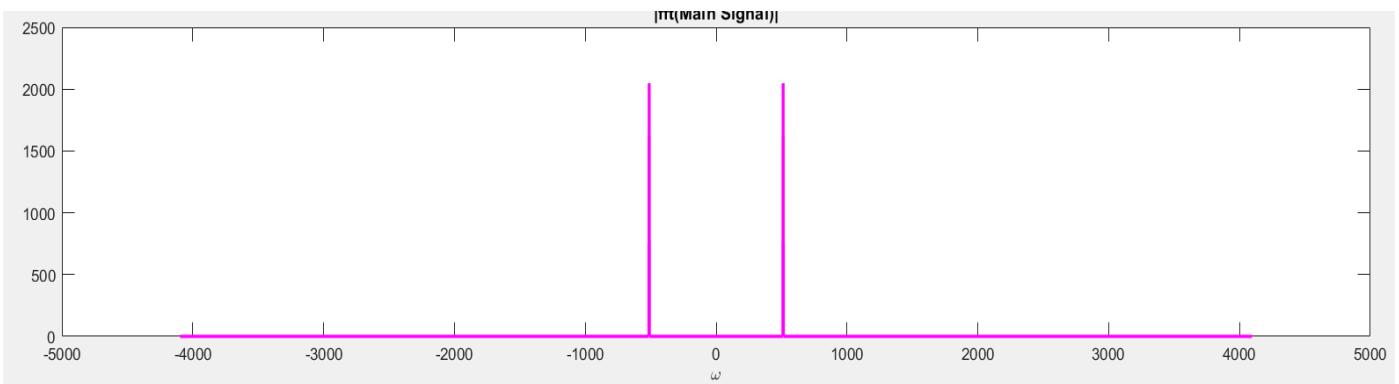
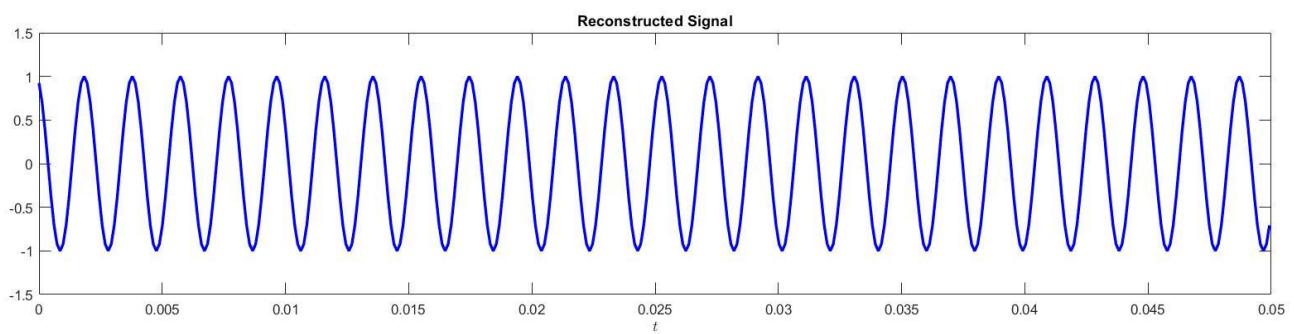
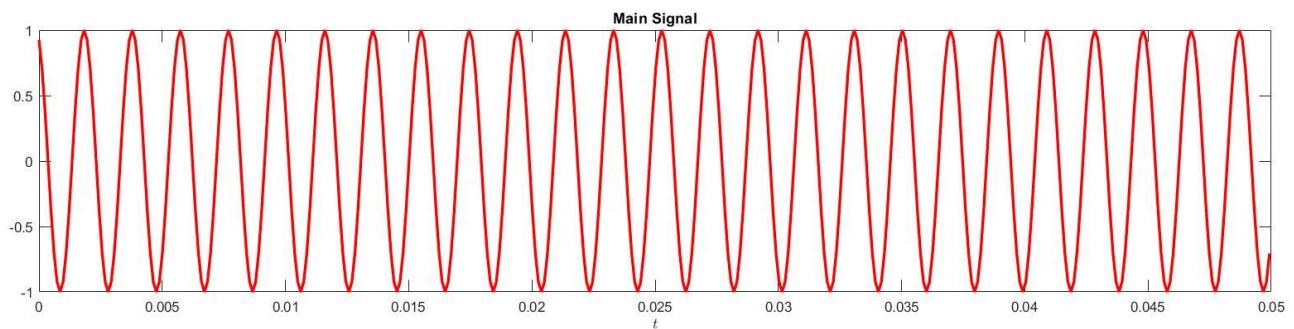
$$\mathbf{b} = \Lambda$$



RMSE =

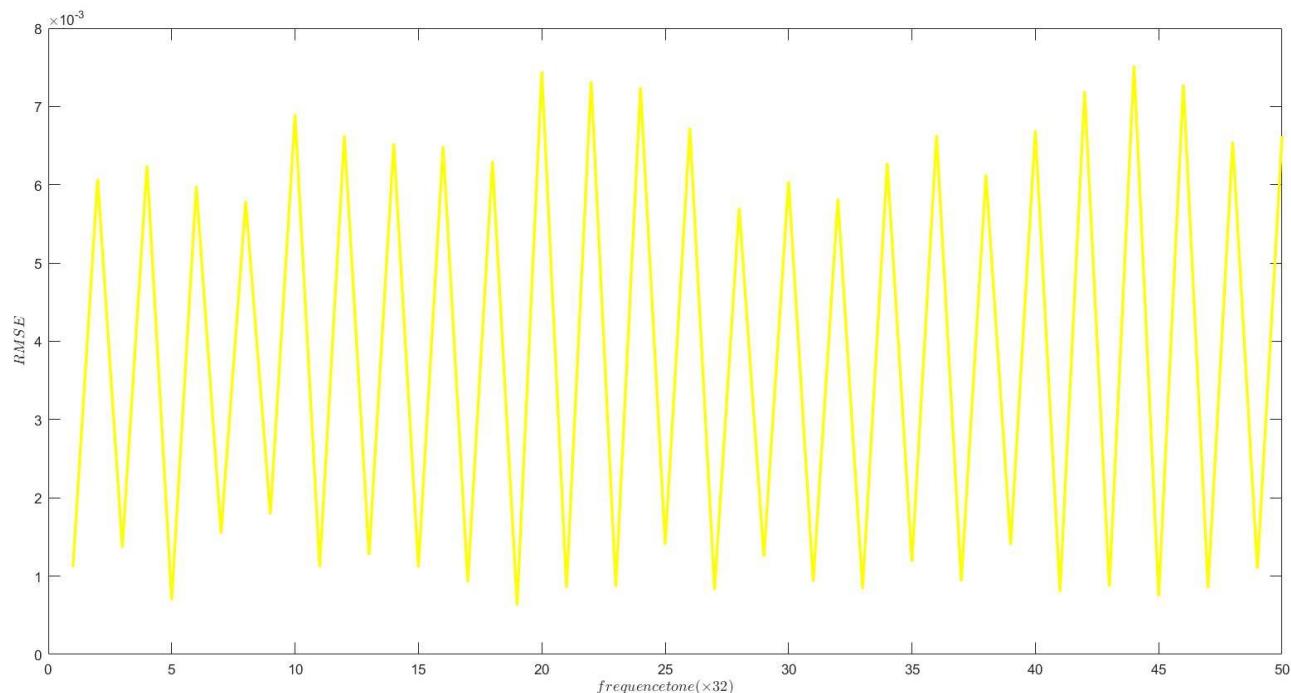
$$b = 12$$

$3.9423e-04$



همانطور که مشاهده می کنید با افزایش تعداد بیت های کوانتیزاسیون، خط کاهش یافته و شباهت سیگنال بازسازی شده به سیگنال اصلی افزایش می یابد و همین امر باعث کاهش RMSE می شود.

در سکشن بعدی با استفاده از یک حلقه ۲۰ تایی مضارب ۱۲۸ را فرکانس خود قرار داده و با محاسبه $RMSE$ آنها طبق رابطه ای که بیان شد به رسم نمودار آن می پردازیم.

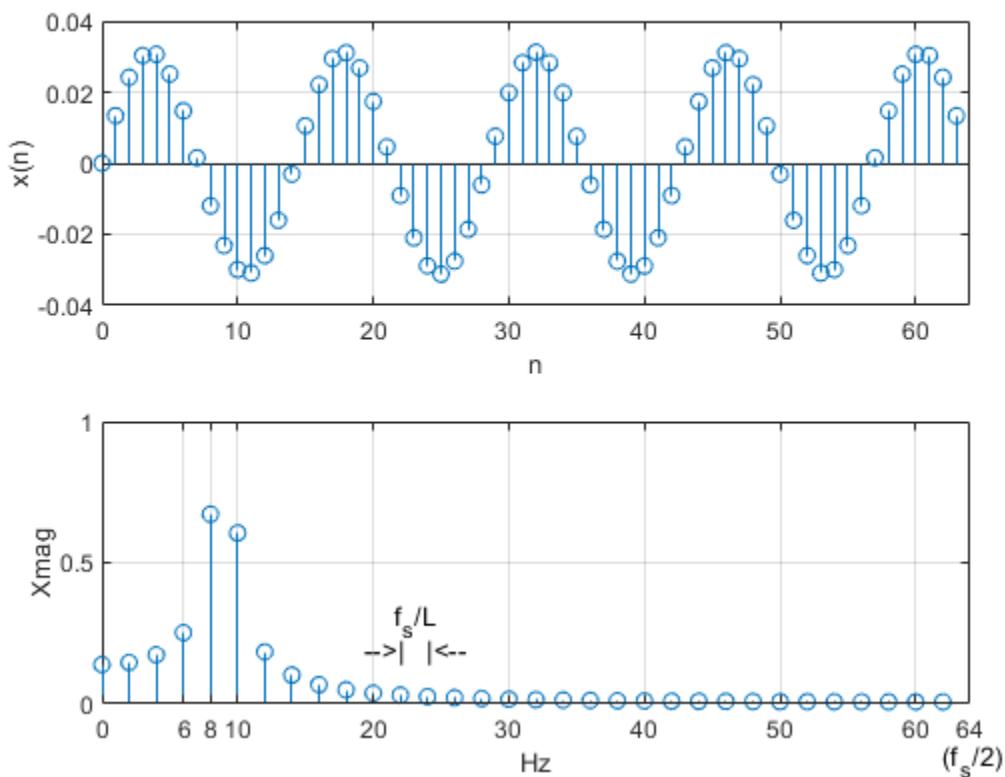


سوال سه)

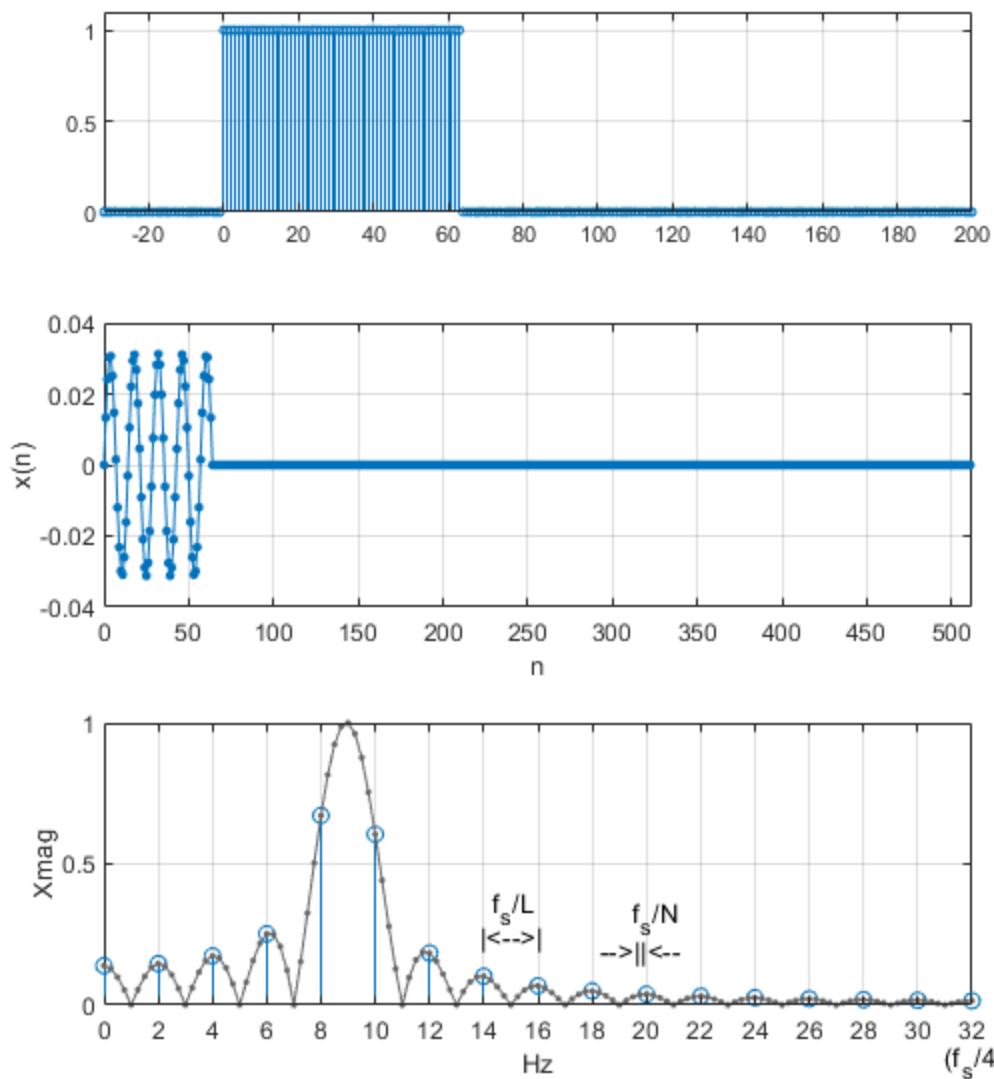
Window function ها توابعی ریاضی هستند که در خارج بازه ای از محور، مقدار صفر دارند، معمولاً با توزیع نرمالی حول مرکز بازه خود هستند و به طور معمول در مرکز مقدار ماقزیم دارند. بر این اساس، هنگامی که تابعی دیگر را در این توابع ضرب می کنیم، حاصل در خارج بازه ای که window function مقدار صفر دارد، برابر صفر است. یعنی در واقع اطلاعات سیگنال را در همان بازه ای که مقدار دارد حفظ می کند.

هنگام استفاده از تبدیل ها روی یک سیگنال، به نوعی نشت طیفی رخ می دهد و طیف دقیقی از فرکانس ها را در خروجی به ما نمی دهد. با استفاده از window function تا حد بسیار خوبی نشت طیفی را به شکلی که می خواهیم مدیریت کنیم و کاهش دهیم.

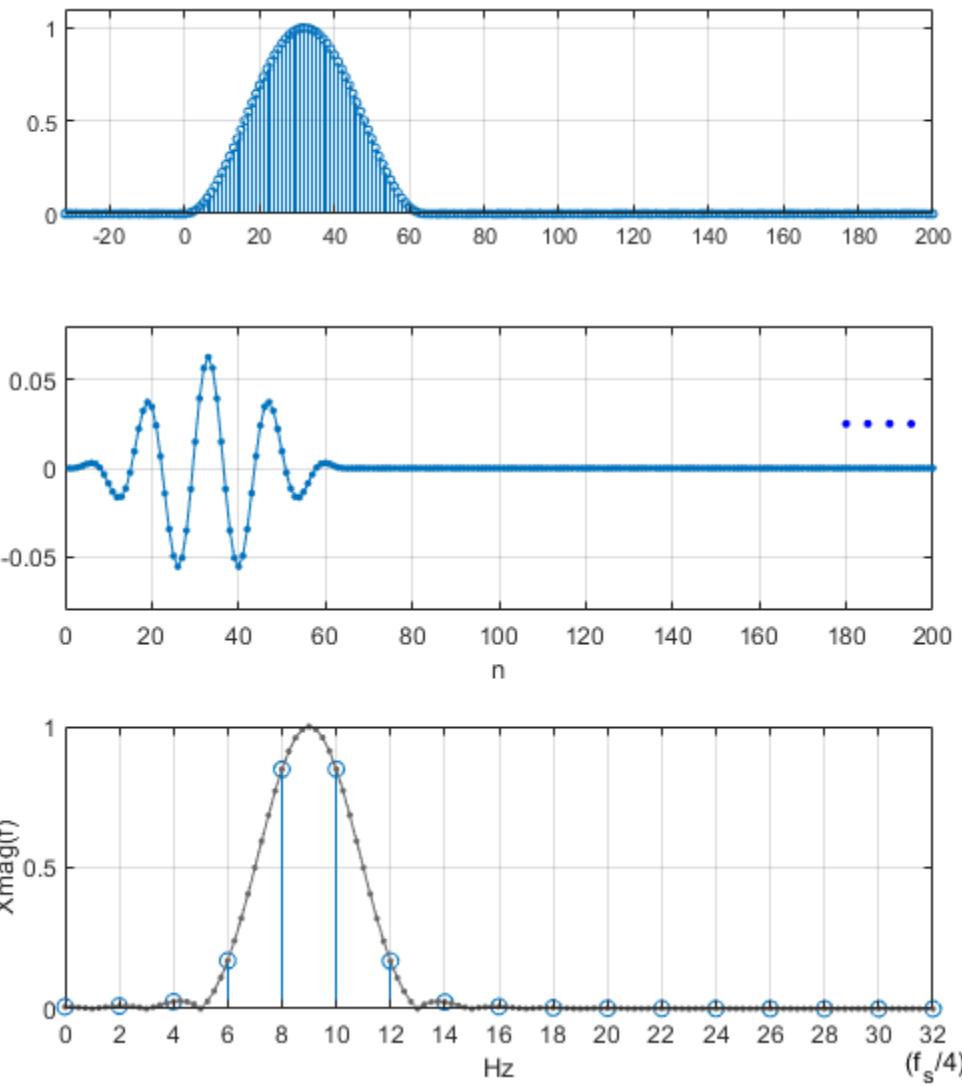
برای مثال تبدیل فوریه این سیگنال سینوسی با فرکانس ۹ هرتز، به شکل زیر در می آید. در اینجا نشت طیفی رخ داده است و مشخص است که در فرکانس هایی غیر از فرکانس ۹ هرتز هم مقدار دارد.



حال برای حل این مشکل می توان از توابع پنجره استفاده کرد. یک تابع پنجره ابتدایی و نه چندان مناسب استفاده از تابع مستطیلی است. در شکل زیر نتیجه استفاده از پنجره مستطیلی را مشاهده می کنیم. اما مشاهده می شود که هم چنان به نشت طیفی روبرو هستیم و این مشکل حل نشده است.



یک تابع پنجره که مورد استفاده قرار می‌گیرد به شکل زیر است. این تابع پنجره بر خلاف تابع مستطیلی به یکباره مقدار صفر به خود نمی‌گیرد. اگرچه این تابع باعث تضعیف بخشی از سیگنال می‌شود اما مشاهده می‌شود که اثر بهتری به نسبت تابع مستطیلی دارد. با مقایسه نتیجه نسبت به حالت قبلی مشاهده می‌شود که فرکانس‌های فرعی مقدار بسیار کمتری پیدا کرده‌اند. هر چند که در نزدیکی فرکانس ۹ هرتز کمی دچار گستردگی شده است.



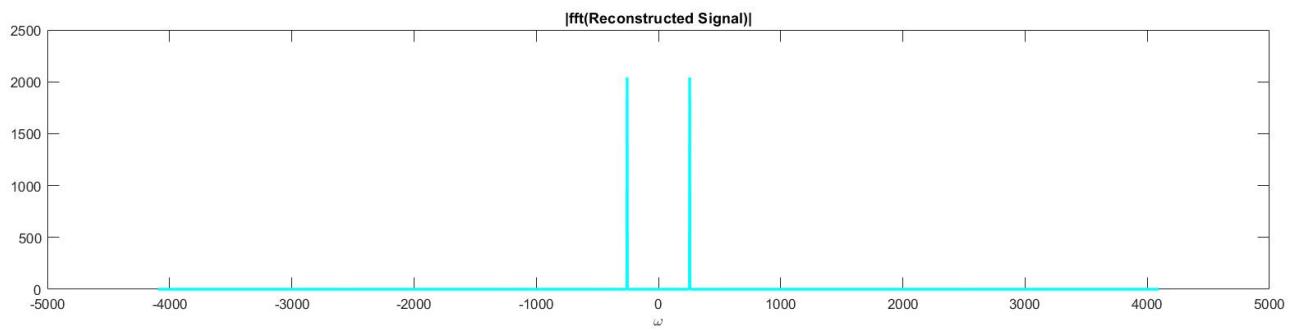
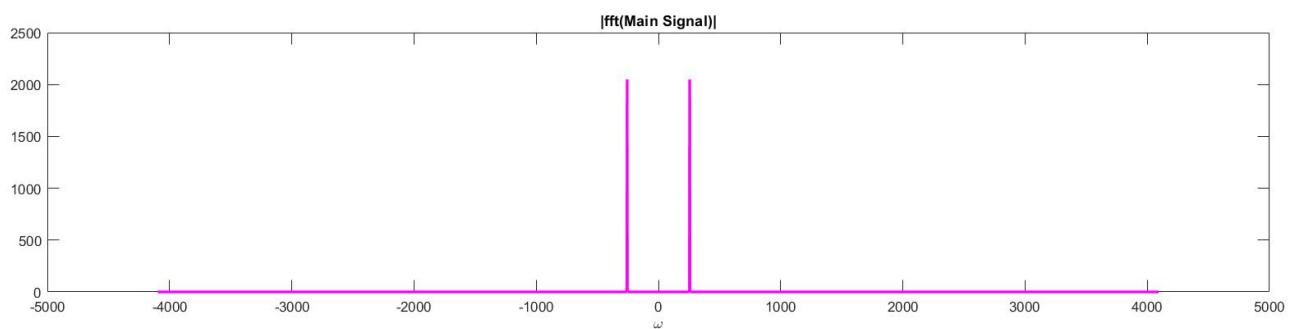
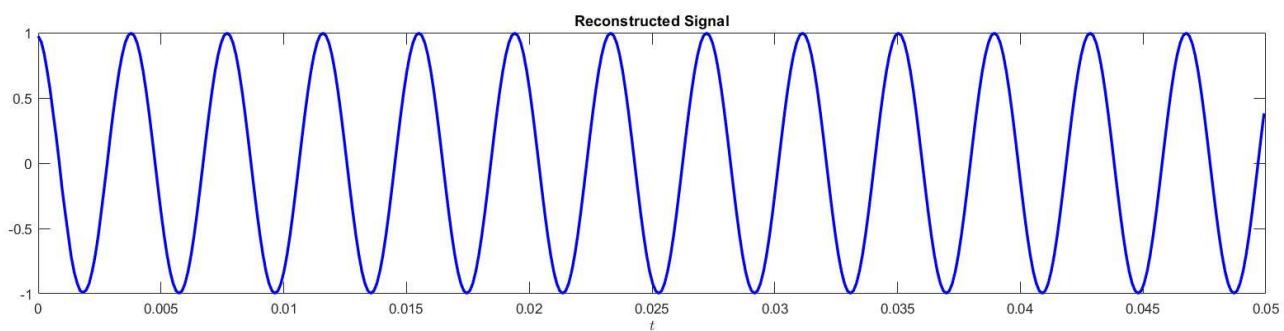
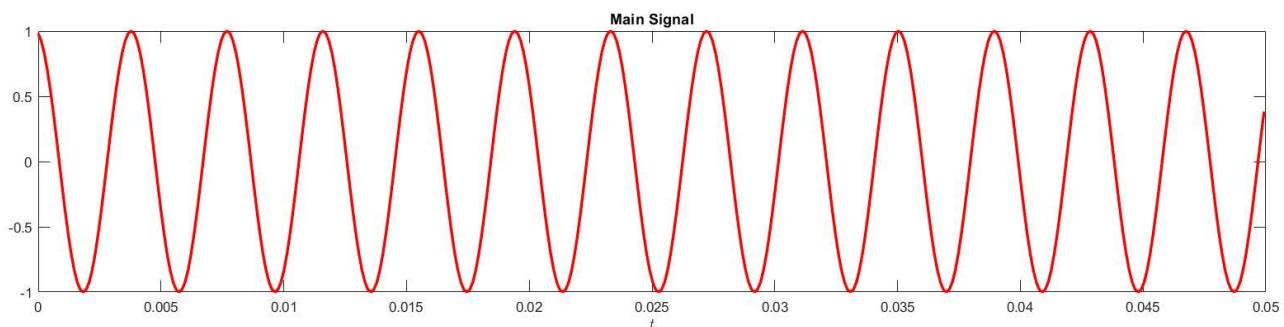
حال کاری را که در بخش ۲ انجام دادیم، در این بخش با اضافه کردن window function در متلب انجام می دهیم. یک بار قبل از انجام تبدیل و یک بار بعد از انجام تبدیل وارون از تابع h استفاده می کنیم و نتایج را در ادامه در گزارش می آوریم.

ابتدا با چهار فرکانس متفاوت به ترتیب با فرکانس ۱۲۸، ۲۵۶، ۵۱۲، ۱۰۲۴ و تعداد بیت های ۸ این کار را انجام می دهیم. سپس برای فرکانس ۲۵۶، با تعداد بیت های ۸، ۱۲ این کار انجام شده و نتایج آن را مشاهده می کنیم. آزمایش RMSE بر حسب فرکانس را هم با تعداد بیت های کوانتیزاسیون ۸ انجام می دهیم.

RMSE =

0.0022

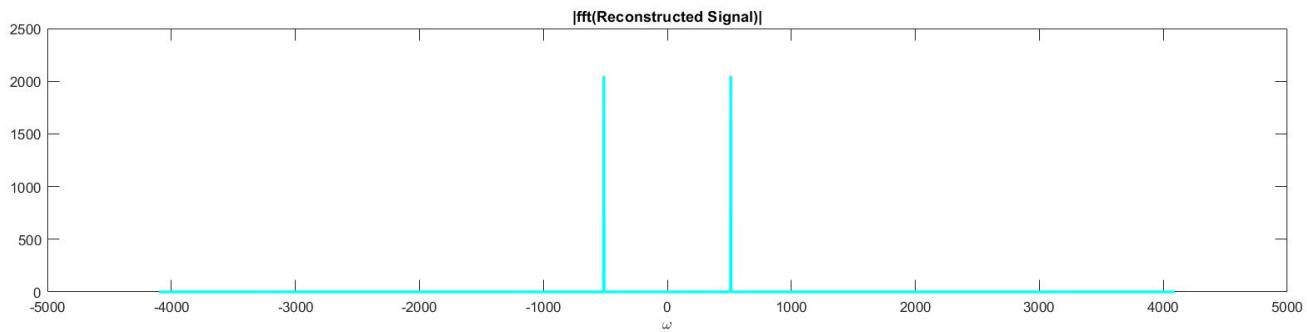
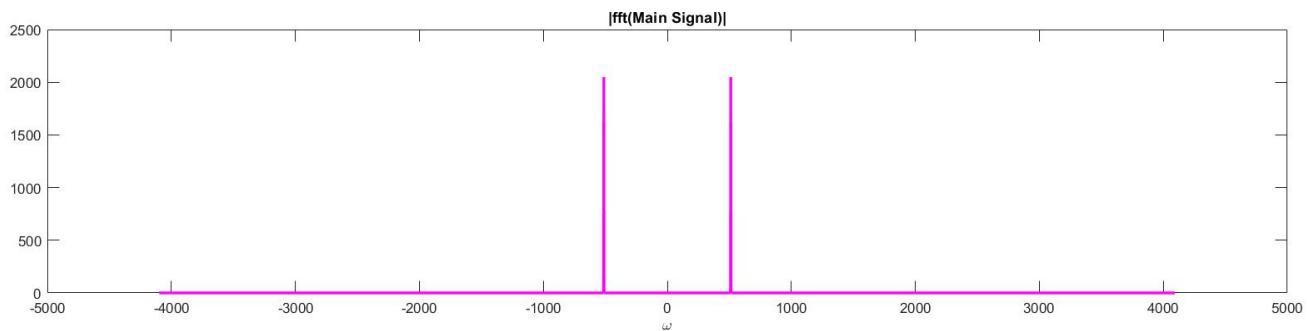
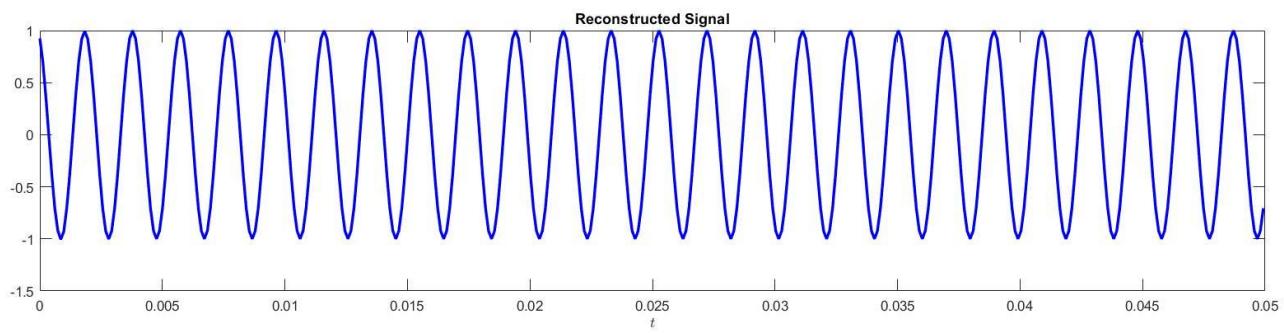
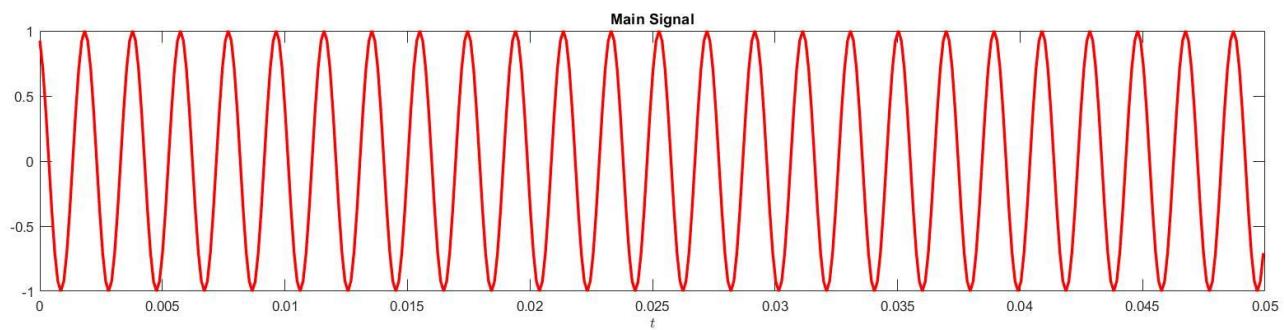
Pure tone 1



RMSE =

Pure tone γ

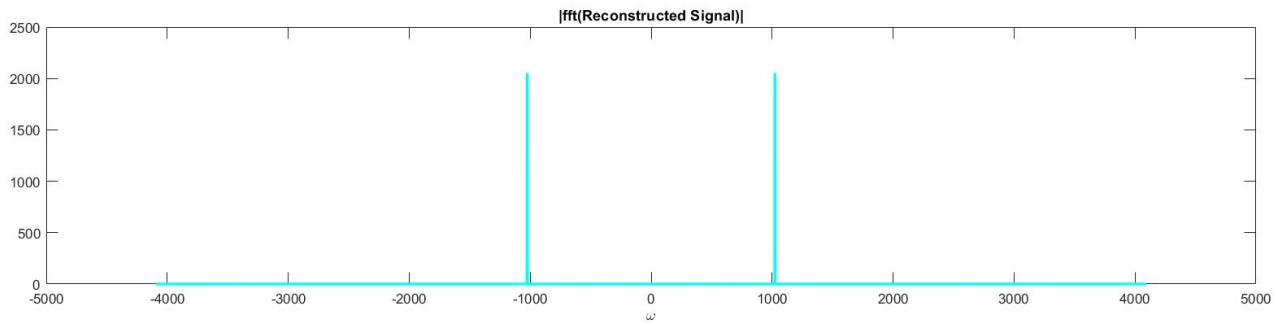
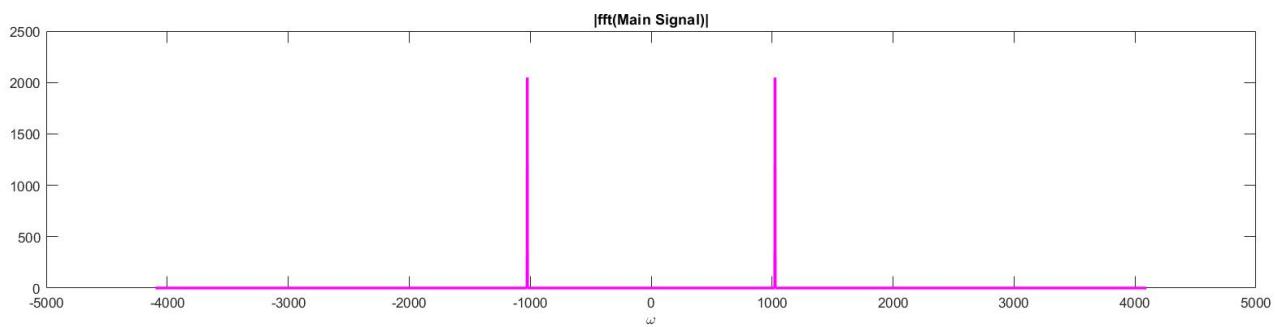
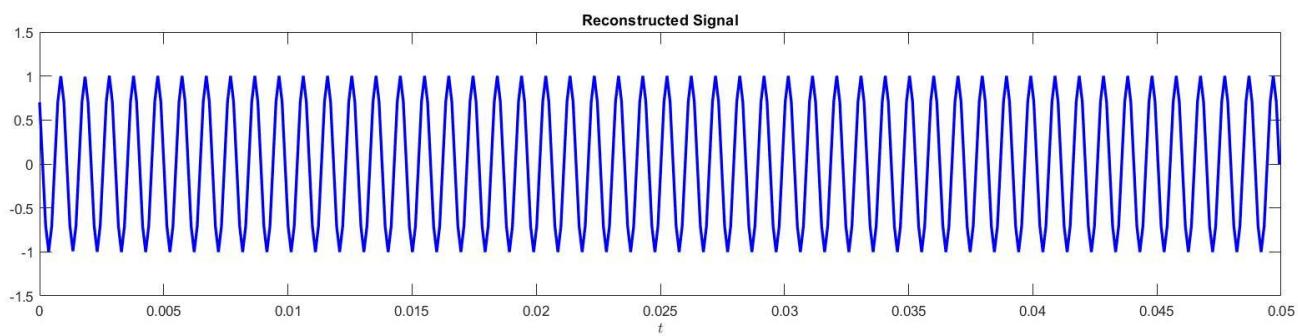
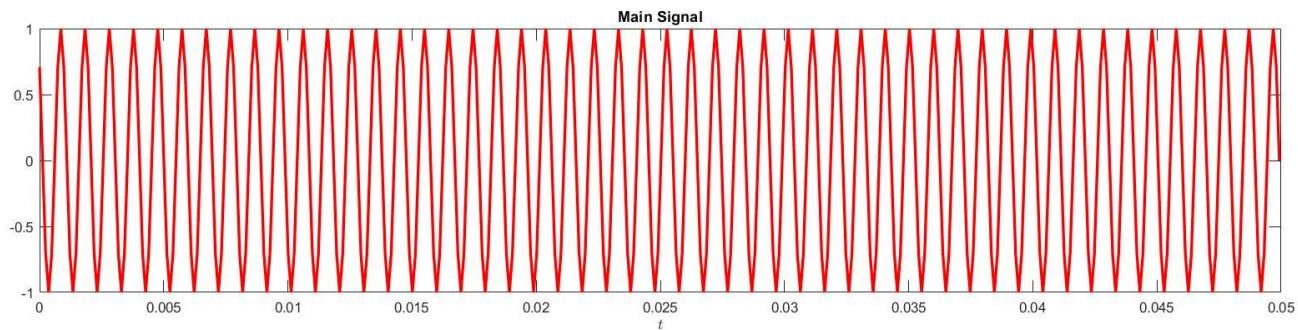
$9.1929e-04$



RMSE =

0.0017

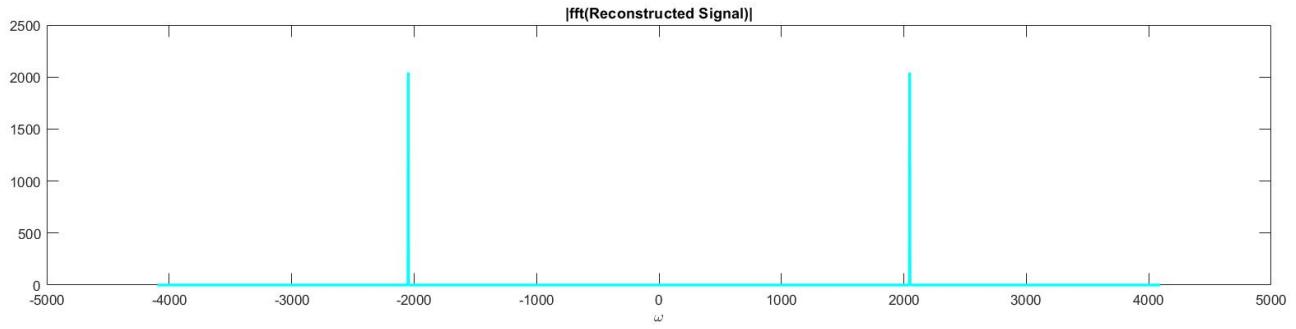
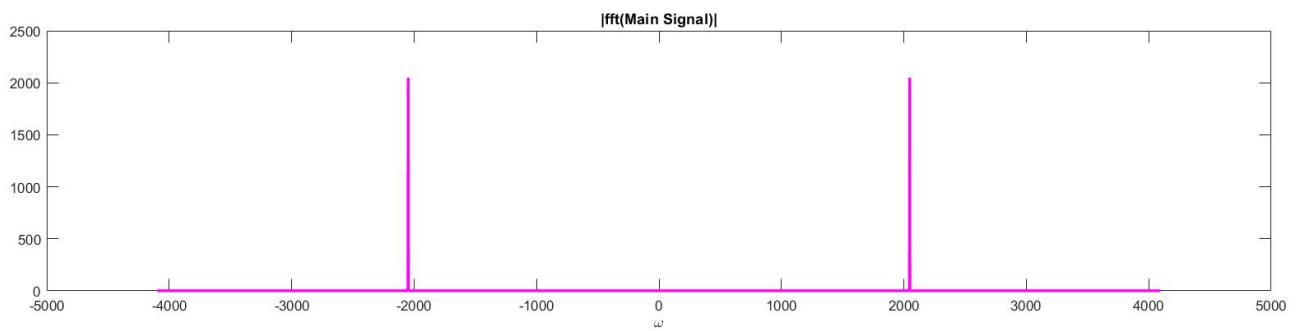
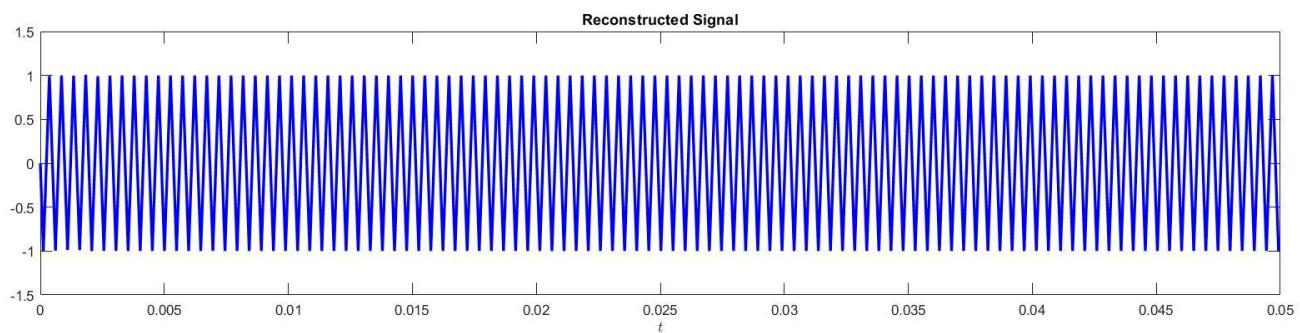
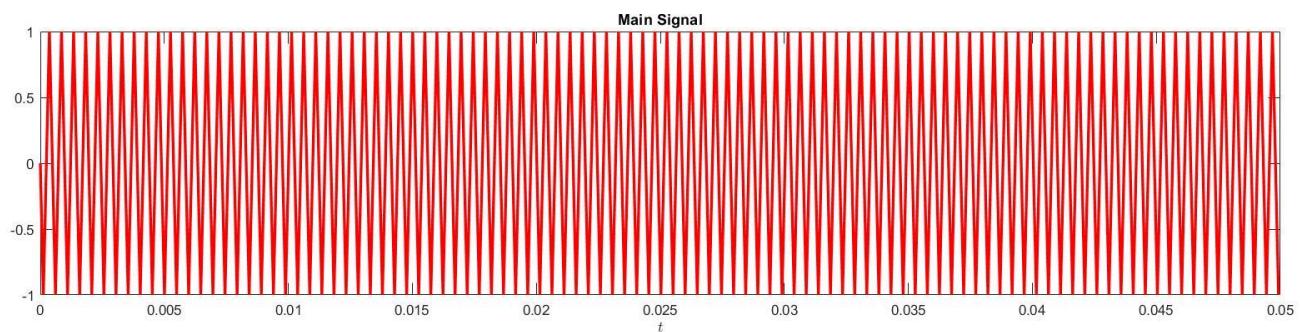
Pure tone ↪



RMSE =

0.0014

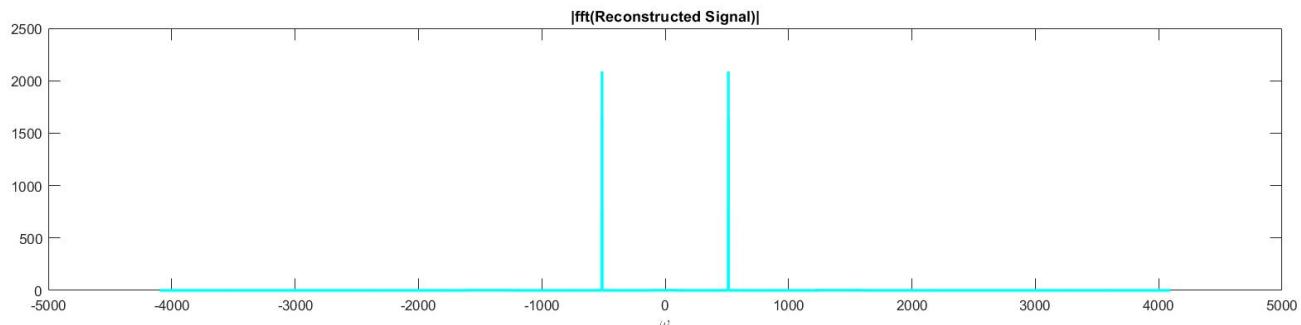
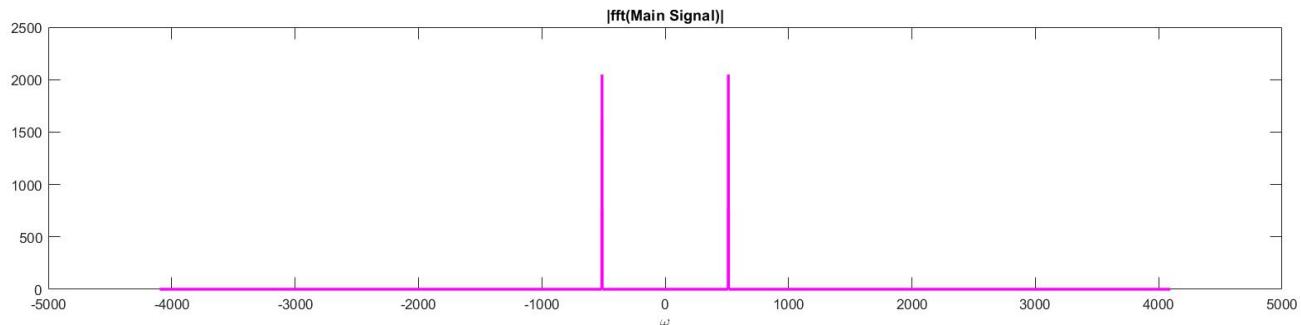
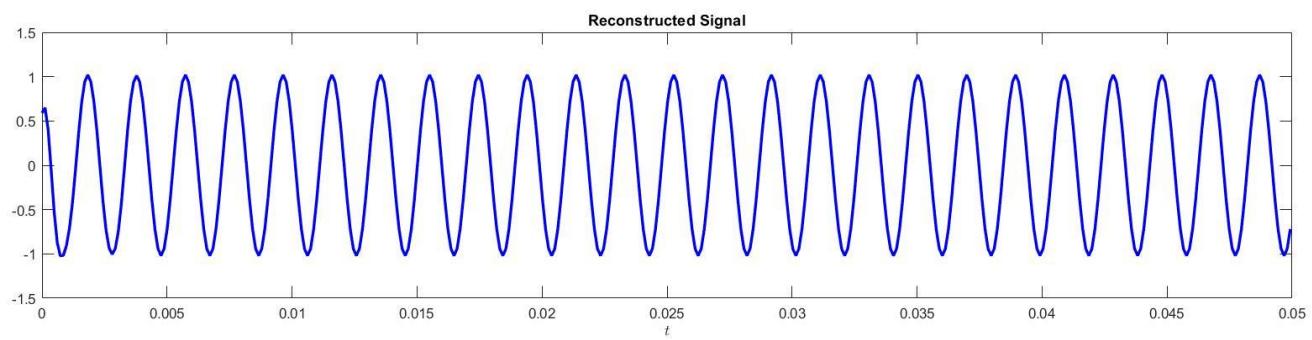
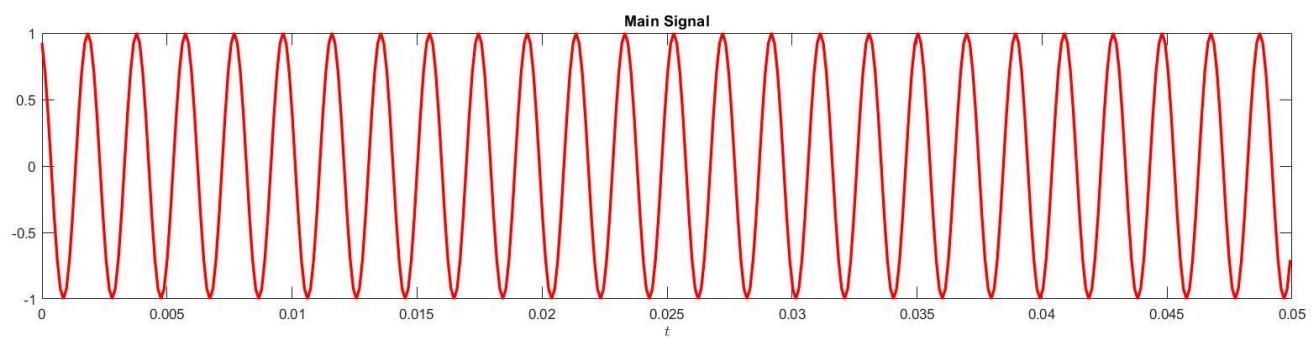
Pure tone ξ



RMSE =

0.0172

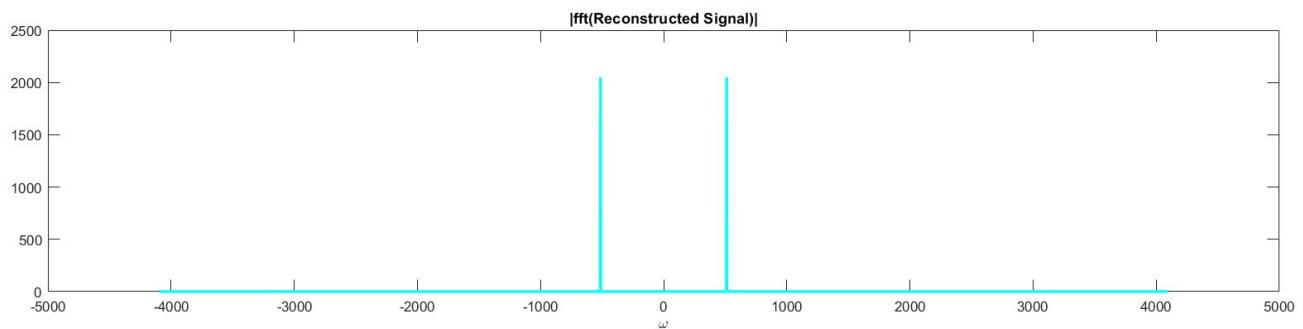
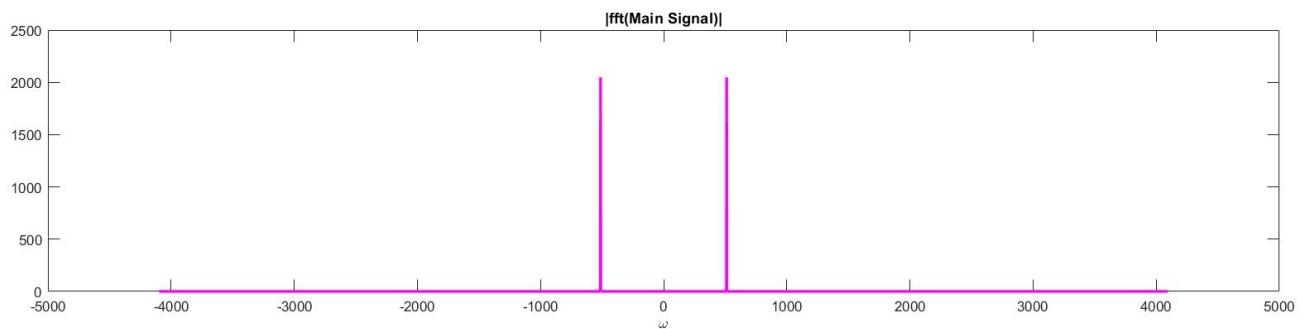
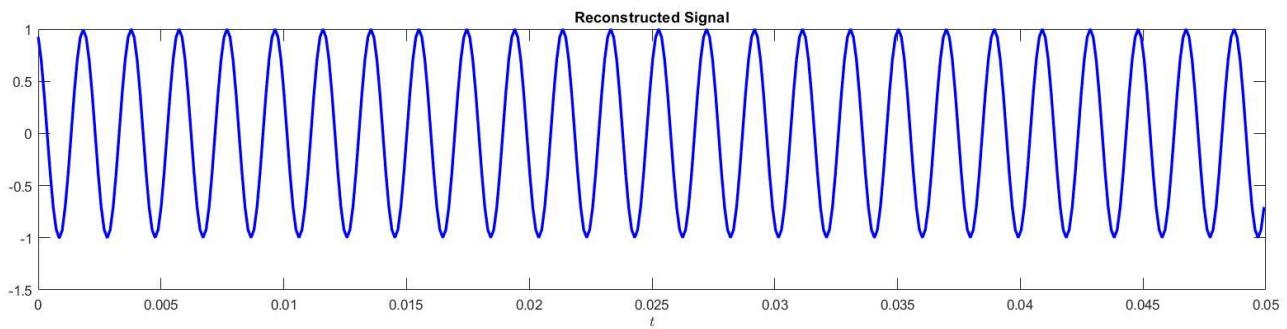
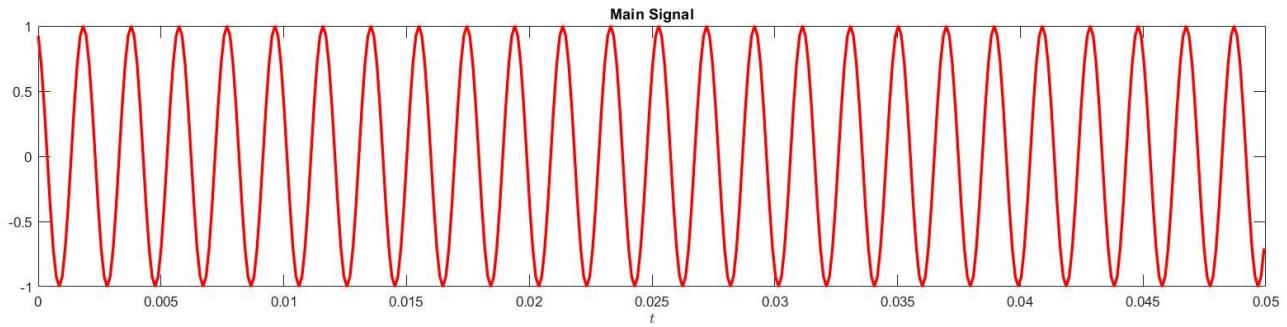
$b = \xi$



RMSE =

9.1929e-04

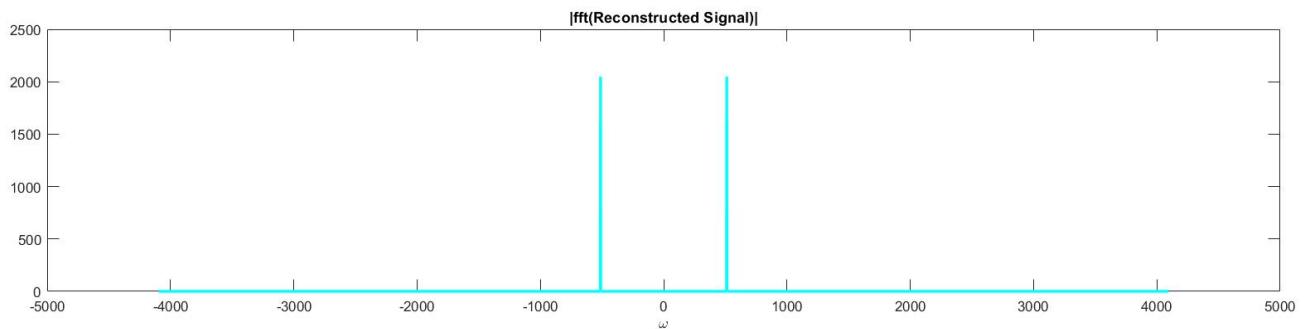
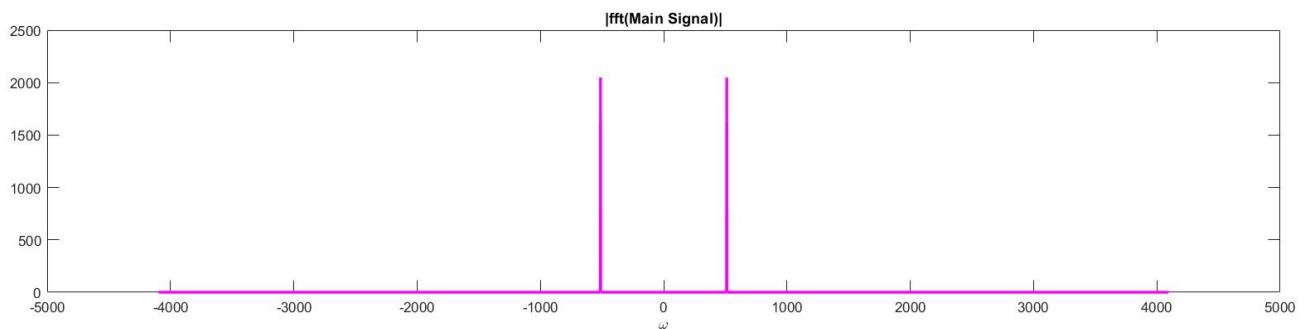
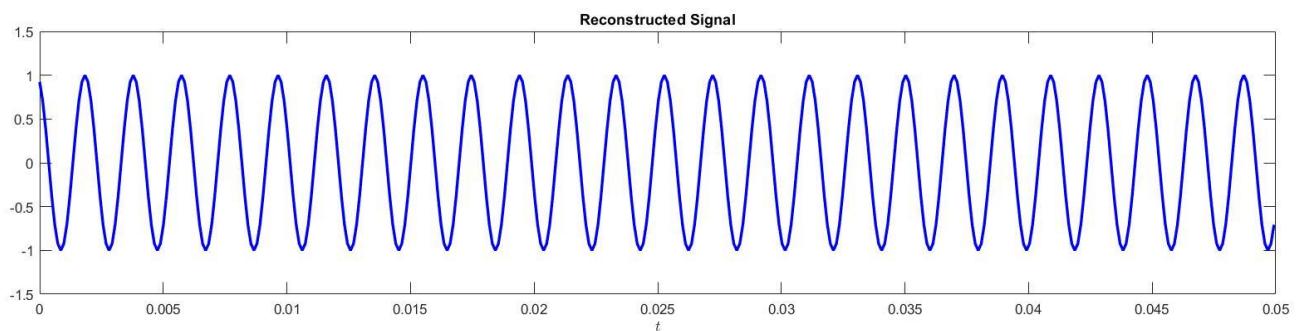
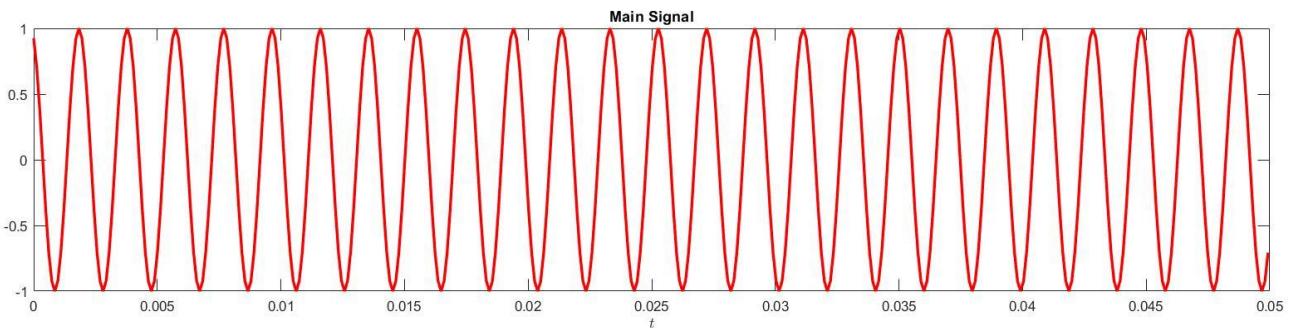
$$\mathbf{b} = \Lambda$$



RMSE =

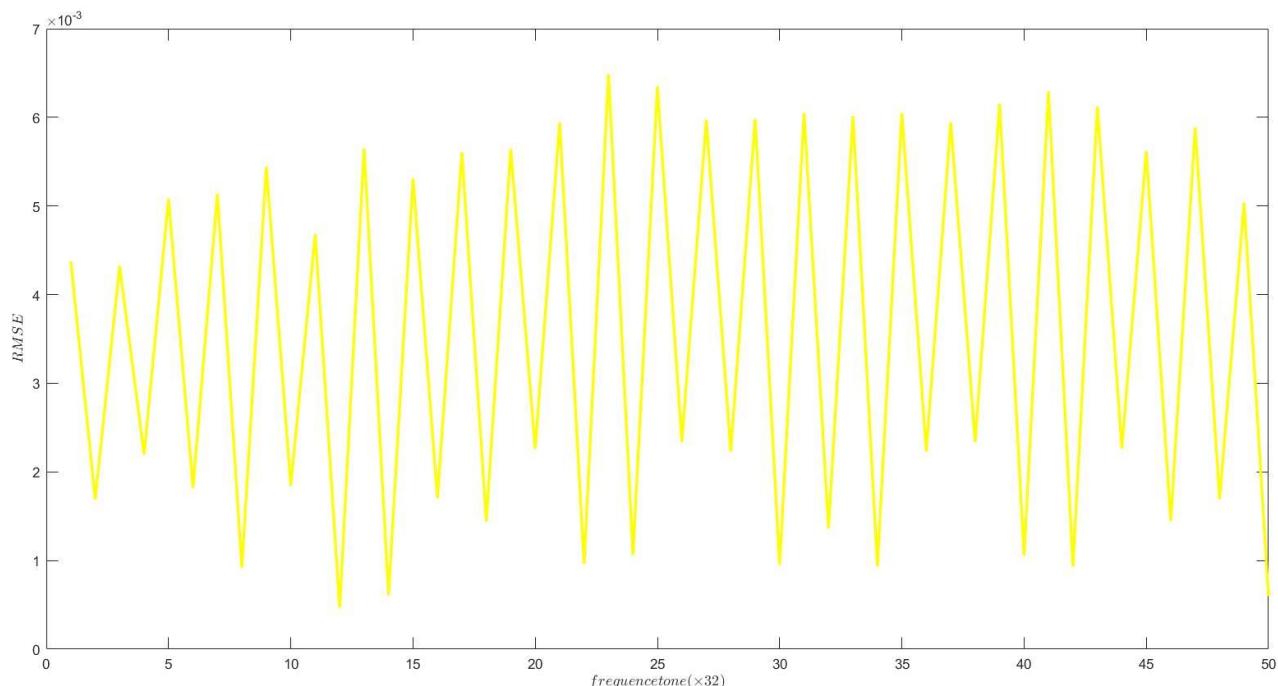
$$b = 12$$

1.2416e-04



به نسبت بخش ۲ که بدون window function بود، مشخص است که سیگنال بازسازی شده شباهت بیشتری به سیگنال اصلی داشته و آن هم در همه بخش ها نسبت به متناظر آن در بخش ۲ کاهش محسوسی یافته است. هم چنین با افزایش تعداد بیت های کوانتیزاسیون هم مثل بخش قبل خطا کاهش یافته و شباهت سیگنال بازسازی شده به سیگنال اصلی بیشتر شده و RMSE هم کاهش یافته است.

در بخش بعدی هم مشابه بخش ۲ بر اساس رابطه گفته شده به محاسبه نمودار خواسته شده می پردازیم.



مشخص است که به نسبت بخش ۲ که بدون استفاده از window function بود، مقدار RMSE به میزان محسوسی در فرکانس های مشابه کاهش یافته است و سیگنال بازسازی شده شبیه تر به سیگنال اصلی است و میزان خطا کاهش یافته است.

سوال چهار)

فرمت WAV فرمتی است که قابلیت ذخیره کردن داده های مربوط به موسیقی و صدا در آن پیش بینی شده است. ویژگی اصلی این است که یک فرمت بدون فشرده سازی و Uncompressed است. به همین علت کیفیت اصلی را حفظ می کند اما حجم بسیار زیادی دارد. به طوری که یک دقیقه موسیقی با این فرمت حدود ۱۰ مگابایت حجم دارد.

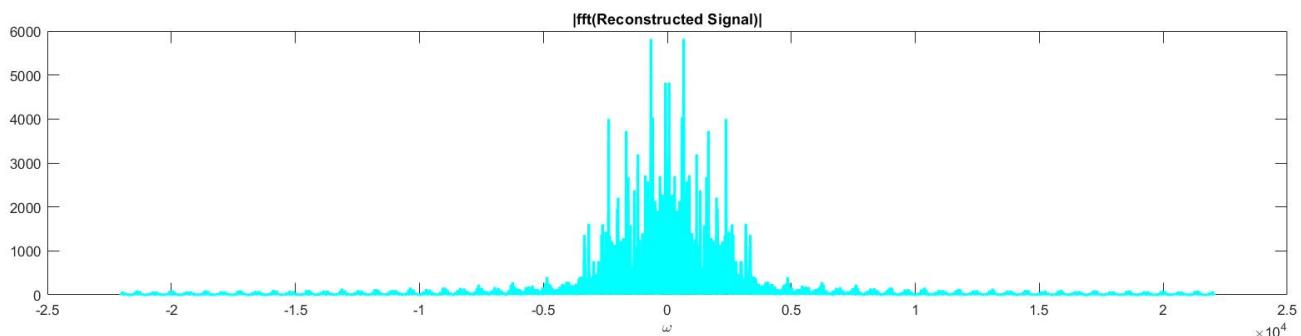
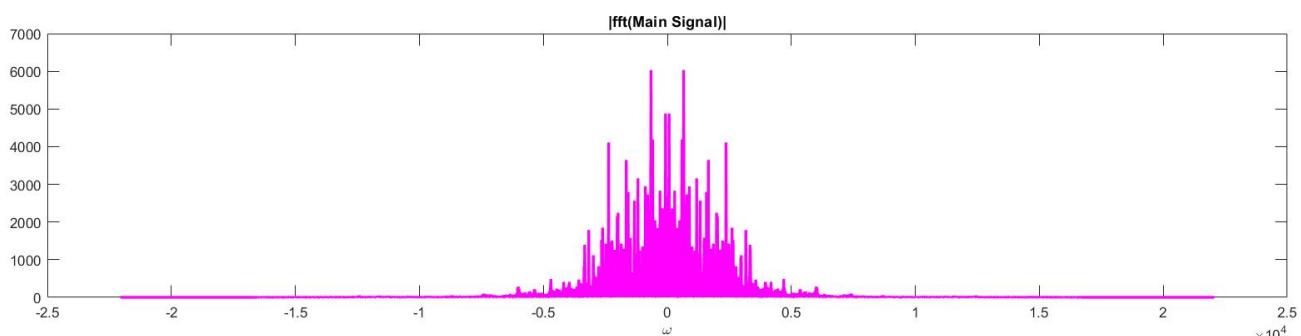
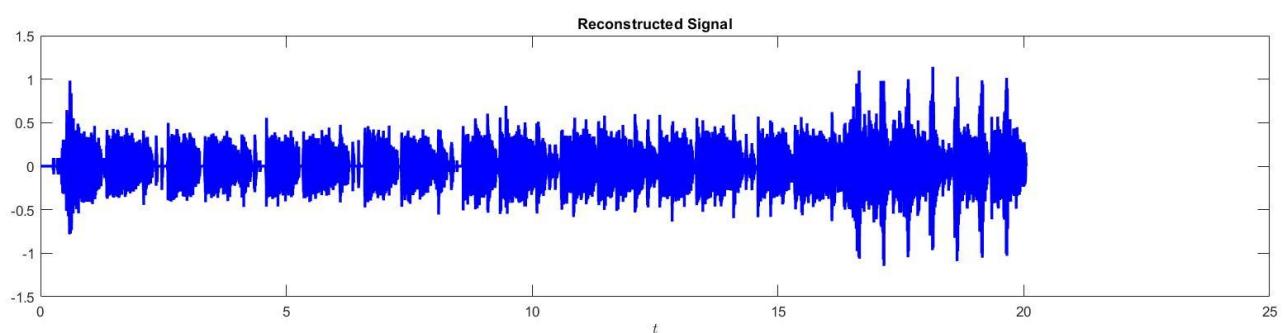
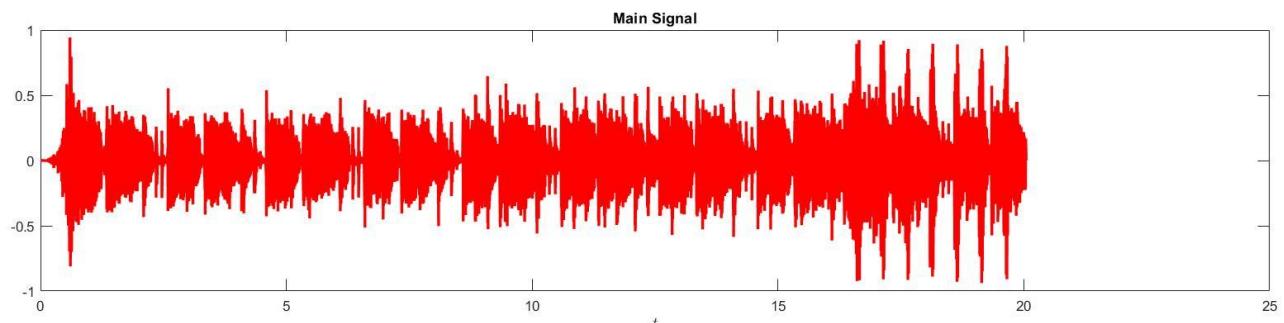
هم چنین حداکثر حجم یک فایل WAV معادل ۴ گیگابایت است و به همین علت مدت زمان ضبط موسیقی با این فرمت نسبت به فرمت های فشرده کمتر است.

حال خواسته های سوال را انجام می دهیم. از کد های مربوط به بخش قبل استفاده می کنیم با این تفاوت که این بار به جای یک فایل صوتی را خوانده و روی آن پردازش انجام می دهیم. هم چنین این کار را ۶ بار انجام می دهیم. برای تعداد بیت های ۴، ۸، ۱۲ و برای هر یک با استفاده از Window function و بدون استفاده از آن. نتایج در صفحات بعدی آمده است.

RMSE =

$b = \xi$, without using window function

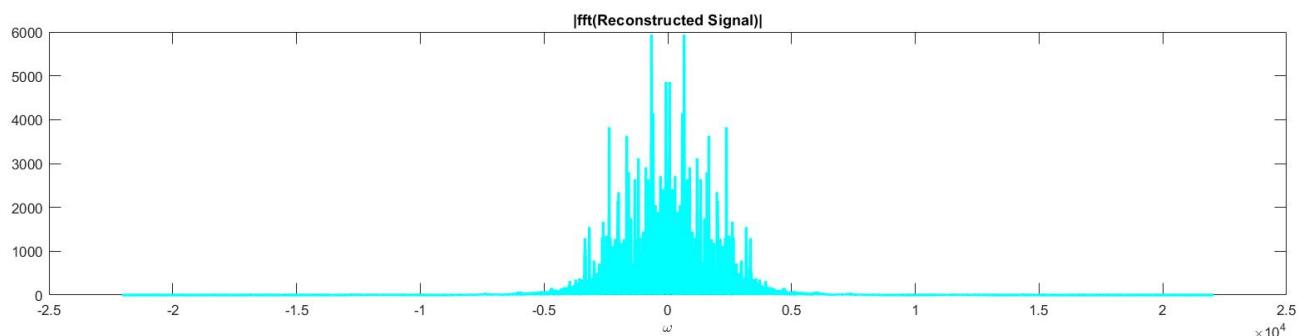
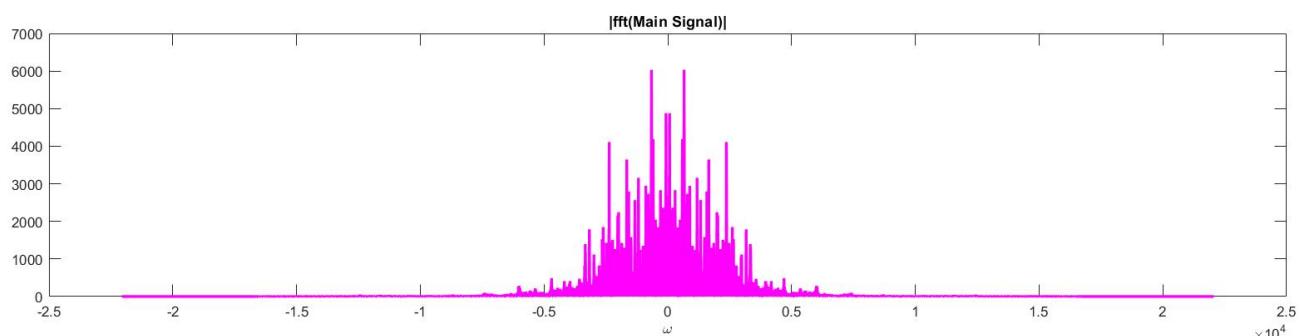
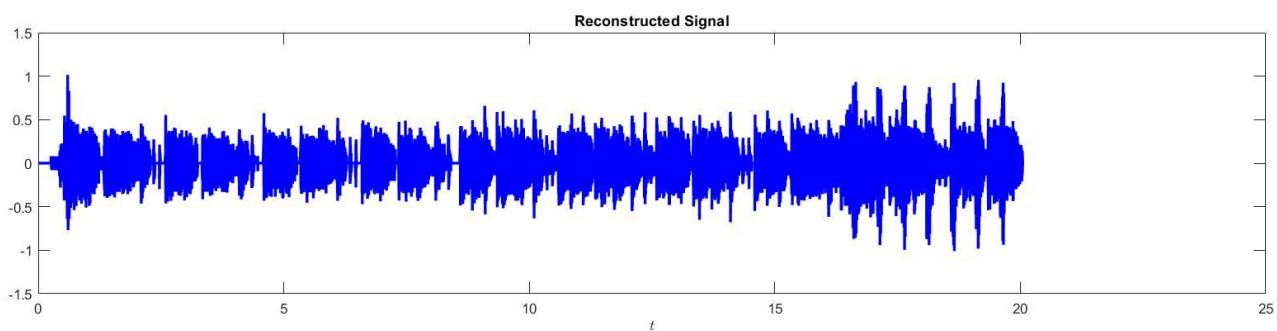
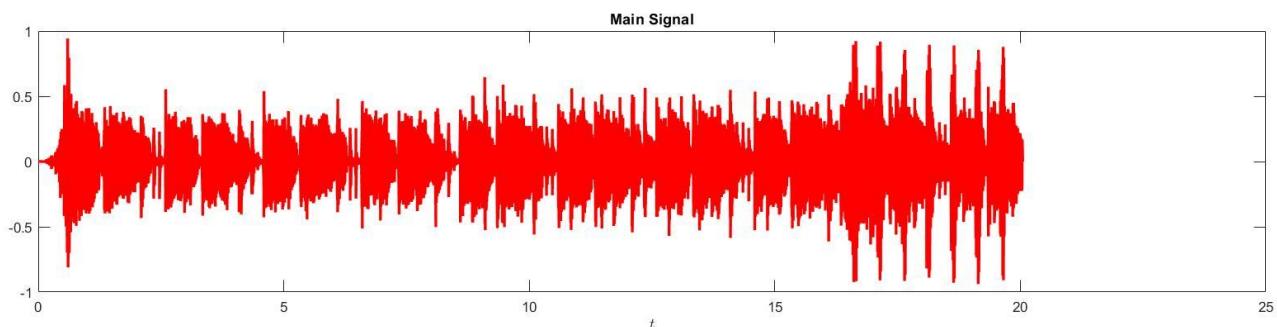
0.0465



RMSE =

0.0431

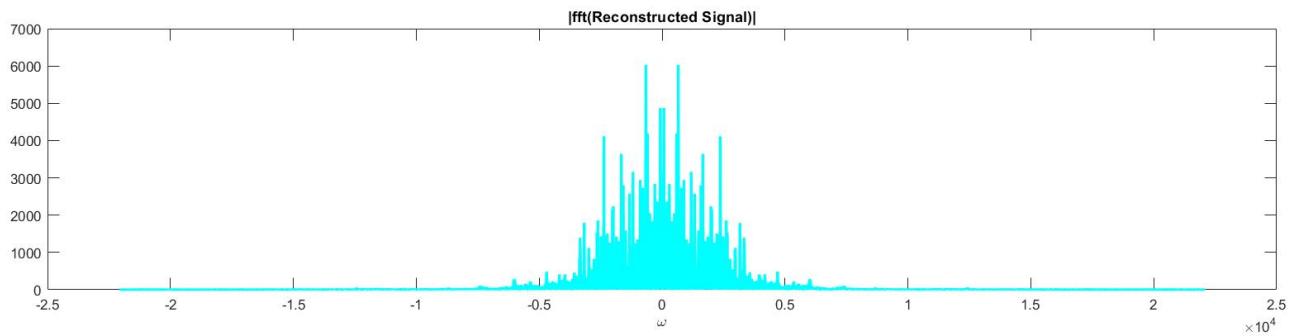
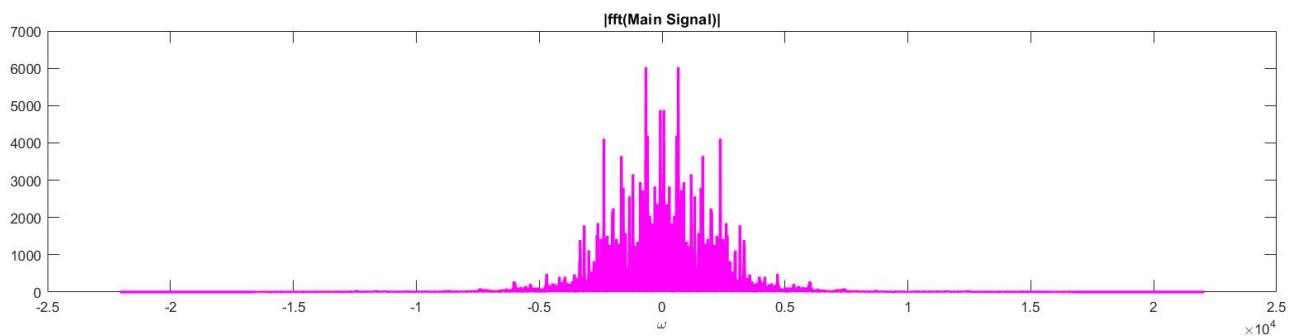
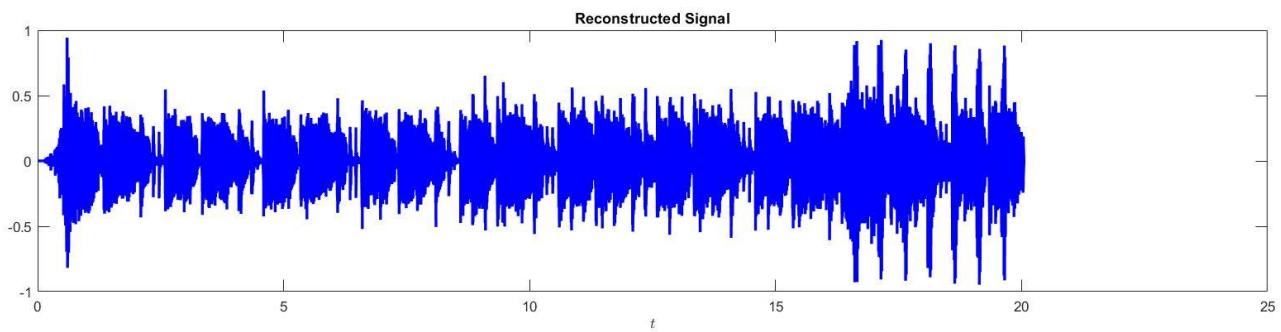
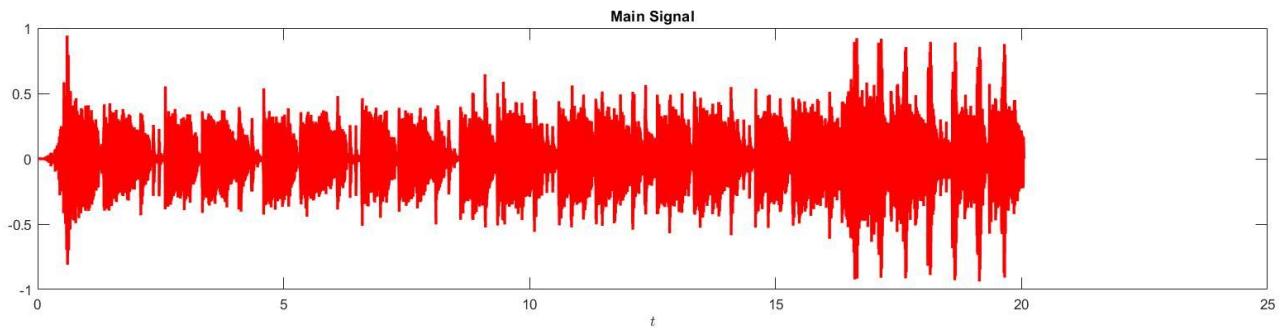
$b = \xi$, using window function



RMSE =

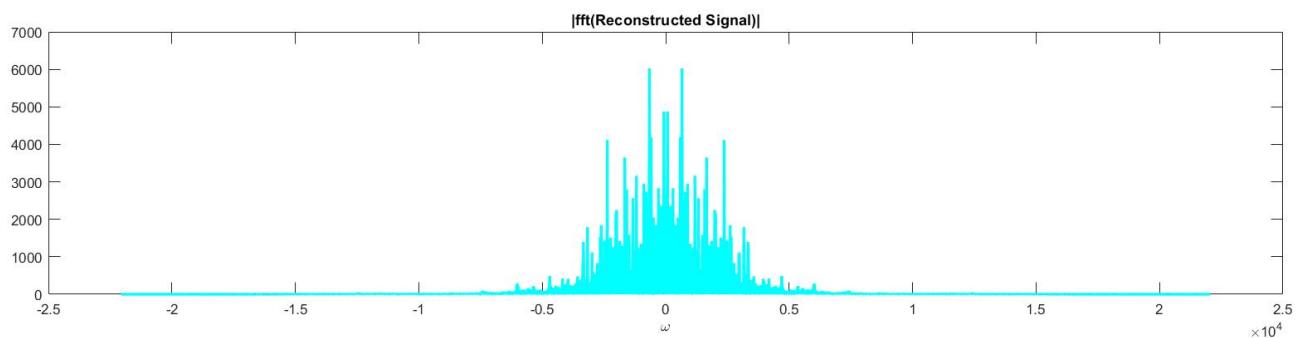
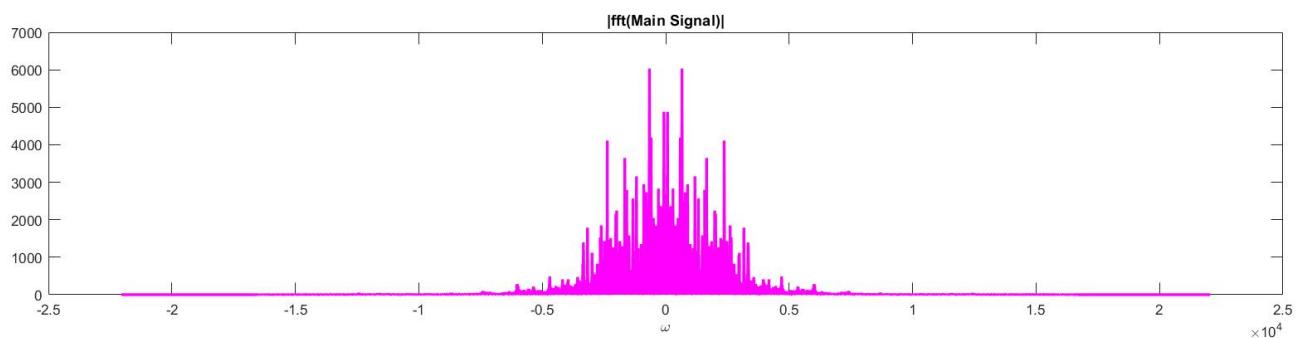
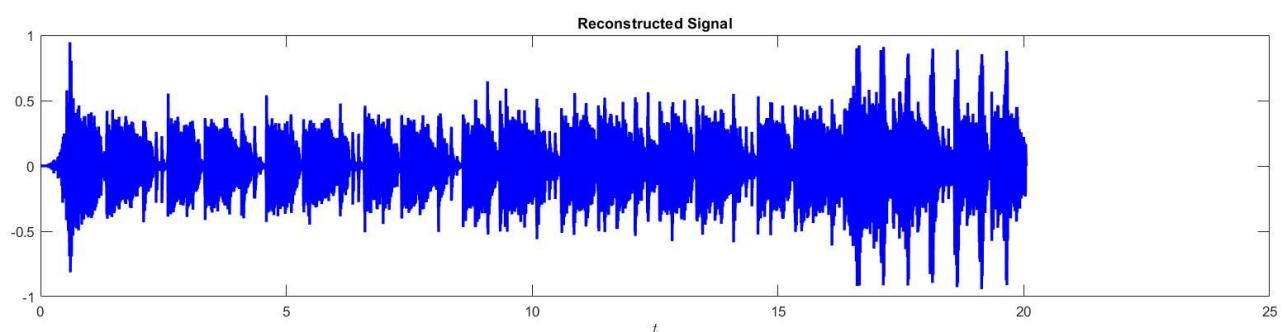
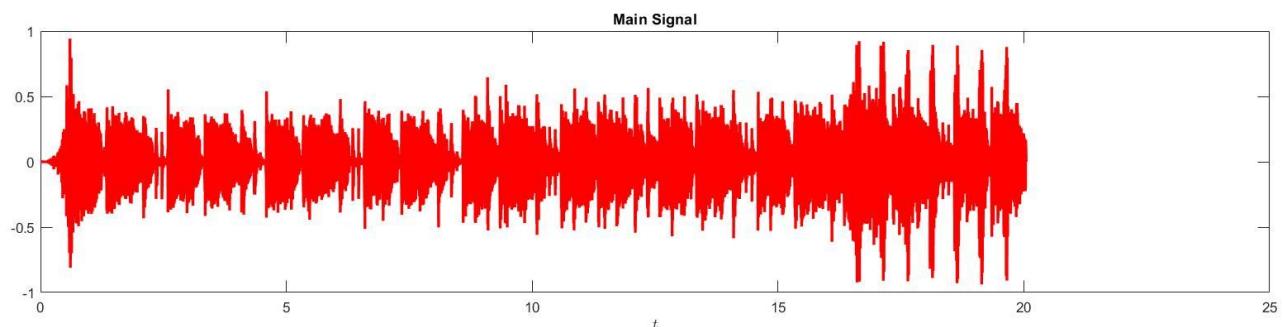
0.0060

$b = \lambda$, without using window function



RMSE =
0.0038

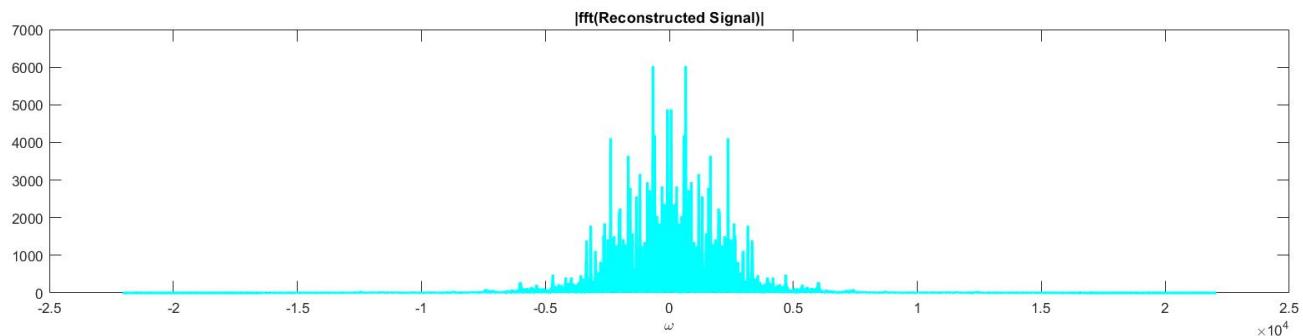
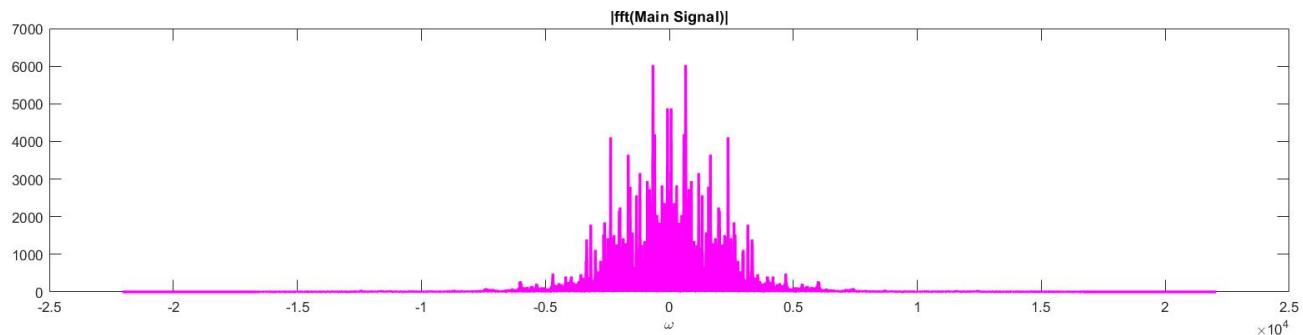
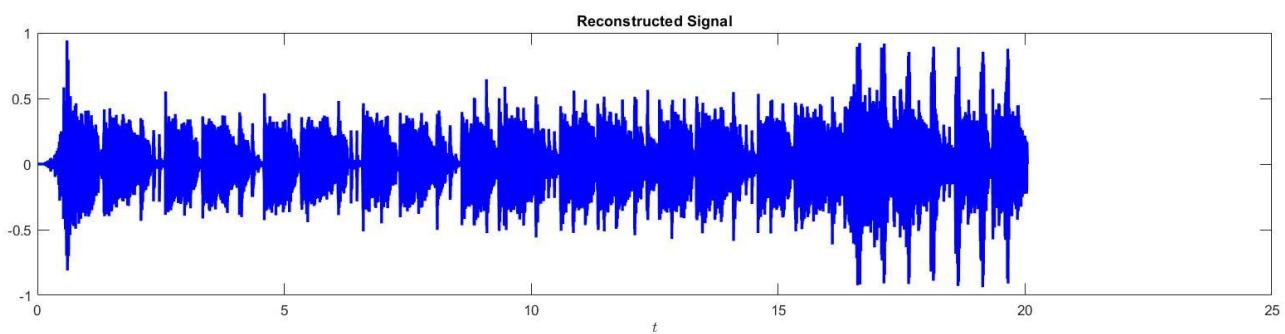
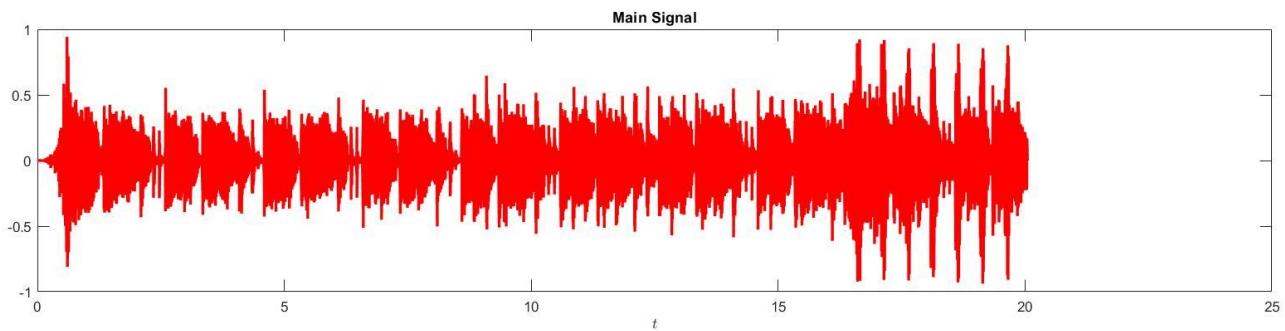
$b = \lambda$, using window function



RMSE =

3.9698e-04

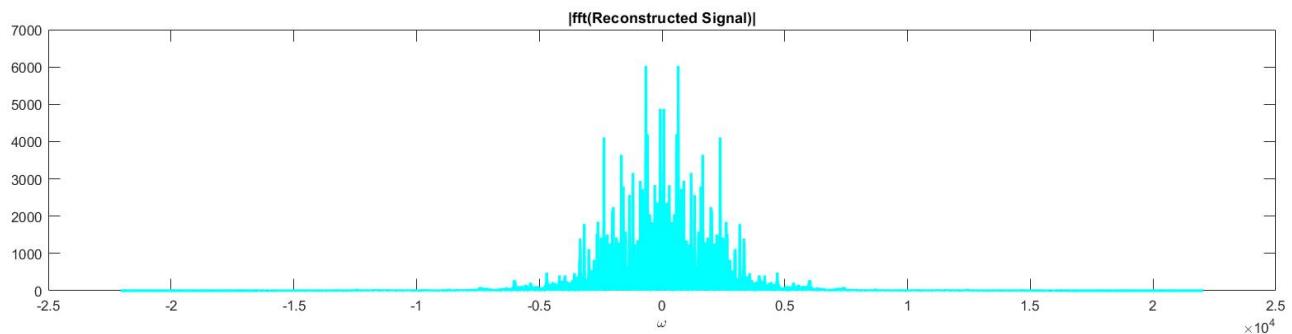
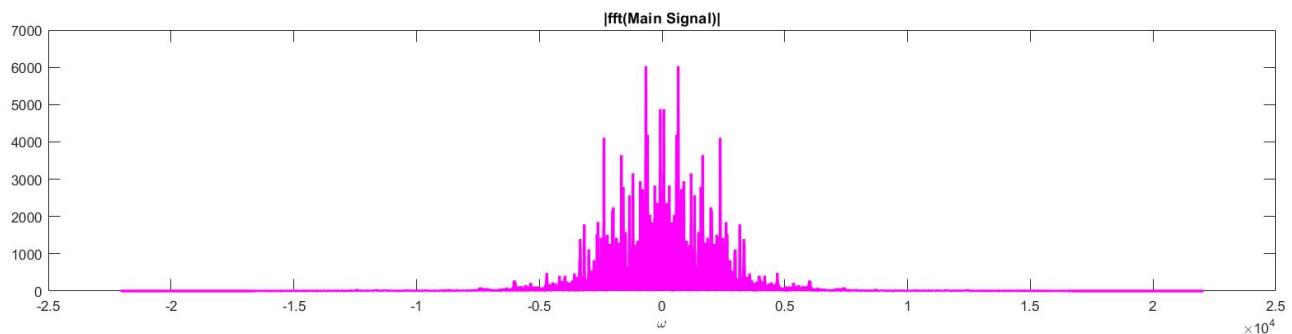
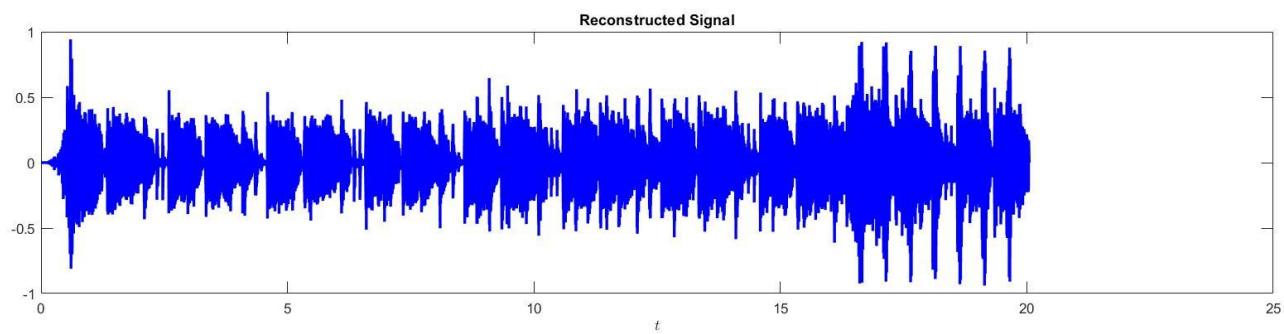
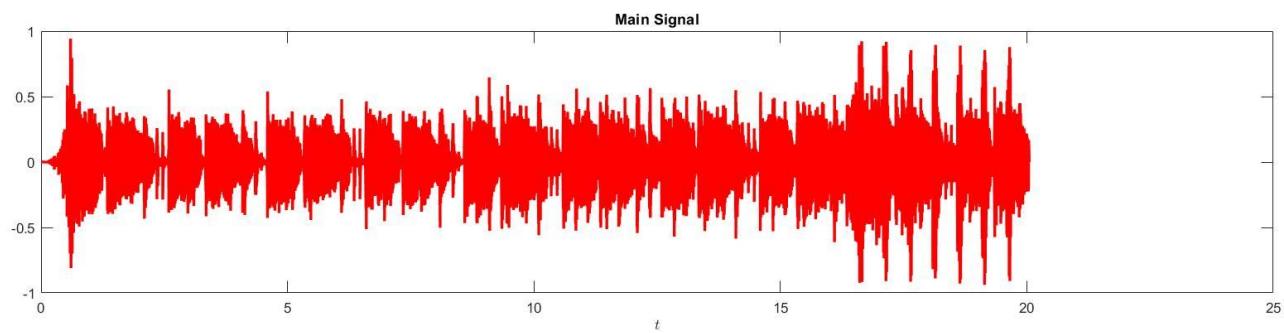
b = 12, without using window function



RMSE =

3.1648e-04

b = 12, using window function



مشخص است که با افزایش تعداد بیت های کوانتیزاسیون خطای کاهش یافته و شباهت سیگنال بازسازی شده به سیگنال اصلی بیشتر شده و RMSE هم کاهش می یابد. هم چنین با استفاده از window function هم همین اتفاق افتاده و نسبت به حالتی که از آن استفاده نشده است، RMSE کوچکتر است.

این اتفاق با شنیدن صوت تولید شده هم به وضوح مشخص است. هر چه تعداد بیت های کوانتیزاسیون افزایش بیابد و هم چنین از window function استفاده شود کیفیت صوت تولیدی یا همان سیگنال بازسازی شده هم بهتر می شود.

در جدول پایین هم این نتایج مشخص است.

RMSE	without using window function	using window function
b = 4	۰۰۴۶۵	۰۰۴۳۱
b = 8	۰۰۰۶۰	۰۰۰۳۸
b = 12	۳.۹۶۹۸e-۴	۳.۱۶۴۸e-۴

مراجع استفاده شده:

ویکی پدیا

فرادرس

كتاب numerical-analysis

۹۹۱۰۲۳۹۴

۹۹۱۰۲۵۰۷

سید محسن نصیری

امیرحسین یاری