بنام خدا

گزارش تمرین سری سوم متلب

محمدرضا سیدنژاد ۹۹۱۰۱۷۵۱

آرشام لولوهری ۹۹۱۰۲۱۵۶

:۱

(در این سوال با توجه به اینکه مقادیر بدست آمده در هر بخش در بخش های بعدی نیز استفاده میشود، در section های کد، فرض بر این است که section ها از Q1.1 تا Q1.8 به ترتیب اجرا میشوند تا داده های مورد نیاز هر section از section های قبل بدست بیاید)

:۱,۱

تابع مذکور در آخرین سکشن نوشته شده است. در اولین سکشن، بردار نت ها و نیز بردار ضرب های آهنگ تعریف شده و به تابع digital_piano داده شده اند. در نهایت هم با audiowrite و فرکانس نمونه برداری ۴۴۱۰۰، بردار y که مربوط به سیگنال آهنگ است در یک فایل wav ریخته میشود.

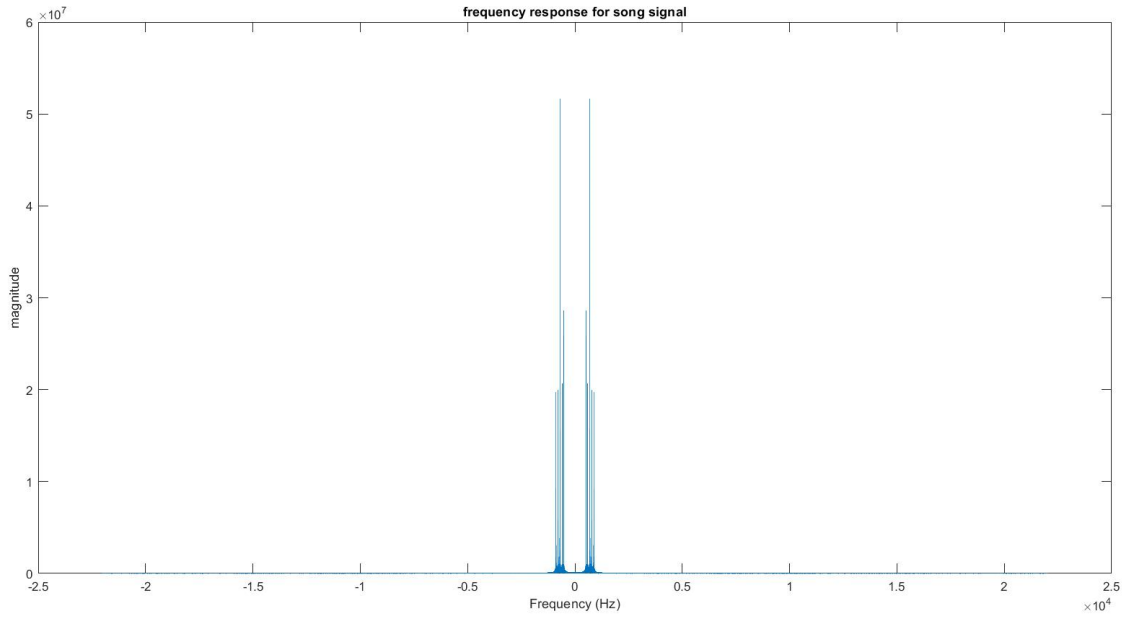
در مدتی که یک کلید پیانو فشرده است(داخل یکی از ضرب ها هستیم)، به تقریب سیگنال سینوسی با دامنه ثابت داریم. پس از رها کردن کلید و قبل از فشردن کلید دوم، سیگنالی که قبلا تولید شده بود، شروع به افت دامنه به صورت نمایی میکند. این مکانیزم در تابع digital_piano پیاده شده است. ابتدا در dur ، مدت زمان نگه داشتن هر کلید به ازای هر ضرب (طول بازه ی هر ضرب) بر حسب ثانیه ریخته شده است. در space، فاصله زمانی بین رها کردن یک کلید و فشردن بعدی را بر حسب ثانیه داریم. دامنه سیگنال ها نیز در M است و بر اساس این سه، کل مدت زمان پخش آهنگ در totalTime

محاسبه شده است(مجموع زمان ضرب ها به اضافه مجموع زمان های space). حال با یک حلقه فور، تک تک نت ها و کلید های ورودی را پیاده میکنیم. فرکانس f0، فرکانس نت اعمالی است که بر اساس نام آن، مشخص میشود. از روی آن فرکانس زاویه ای w0 را میسازیم. signalDur، مدت زمانی است که کلید آن نت فشرده شده است. زمان شروع نت جدید در signalStart ریخته شده، به نحوی که اگر اولین نت باشد، زمان آن صفر است و در غیر این صورت، این زمان، حاصل جمع زمانِ تمام نت هایی است که پیش از این نواخته شده بود. زمان اتمام فشردن کلید نیز در signalFinish آمده است. در انتها، فاصله ی بین رها کردن این کلید و فشردن کلید بعدی رابا جمع کردن signalFinish+space ایجاد کرده و در partFinish میریزیم.

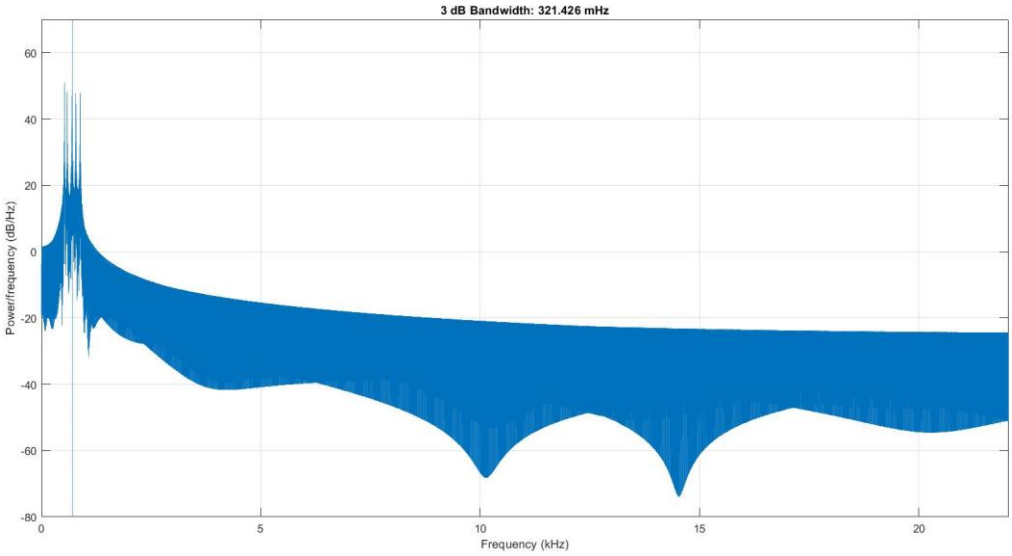
حال هر بار برای ساختن سیگنال های جدید، باید یک سیگنال سینوسی با دامنه M و فرکانس w0 بسازیم که از زمان signalStart شروع و در زمان signalFinish پایان می یابد. این کار را به راحتی با دو تابع پله واحد انجام داده ایم. سپس، پس از رها کردن کلید نیز از زمان signalFinish تا زمان partFinish، همانطور که ذکر شد، یک سیگنال سینوسی با همان فرکانس داریم که دامنه آن به صورت نمایی افت پیدا میکند. این هم با تابع پله قابل ساخت است. همه این موارد در انتهای digital_piano انجام میشوند و تابع کامل میشود.

:۱,۲

تعداد کل نمونه ها که برابر با طول y در سکشن Q1.1 است را L مینامیم. فرکانس نمونه برداری fs را مشابه قسمت قبل قرار میدهیم و سپس برای فوریه گرفتن، از دستور fft استفاده میکنیم. آنچه در اینجا بدست می آید، برای پاسخ فرکانسی در بازه صفر تا fs است که برای متقارن کردن طیف نسبت به مبدا، نیاز به اعمال fftshift روی خروجی هستیم. این بردار مختلط را در fft_y_shift میریزیم. محور افقی که همان فرکانس است را نیز در f تعریف کرده ایم و پس از اعمال شیفت، لازم است این بردار از fs/2– تا fs/2 باشد و نیز طول آن برابر با fft_y_shift باشد که بتوان نمودار را رسم کرد. این کار با linspace انجام شده و سپس حاصل توسط plot رسم شده است:

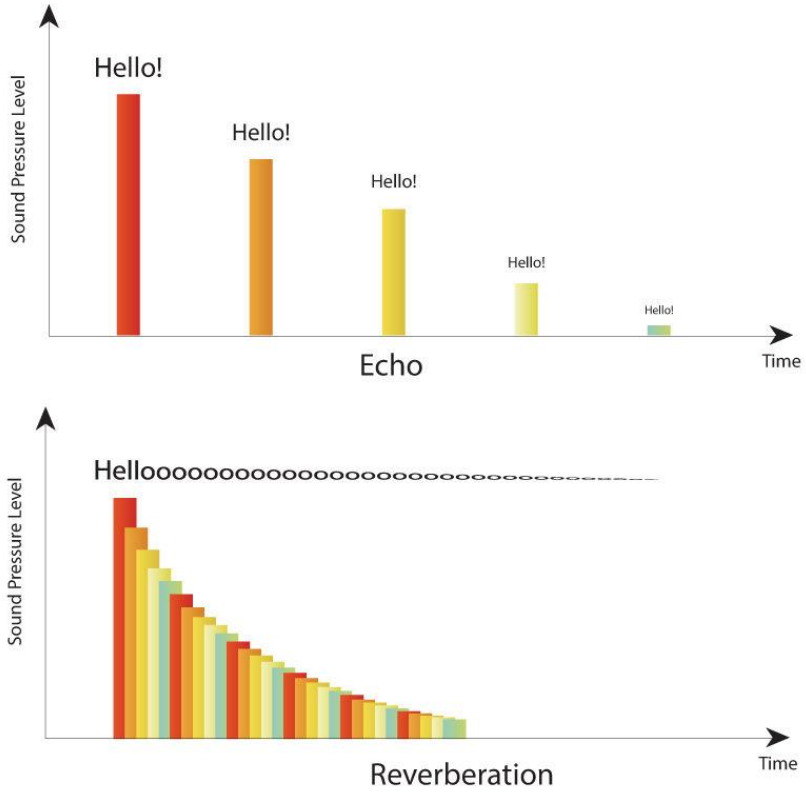


به صورت تقریبی میتوان پهنای باند را نیز حساب کرد. از روی نقاط شکل، ماکزیمم دامنه در حدود 5.16*10^(7) است که برای پهنای باند باید این دامنه بر رادیکال ۲ تقسیم شود، یعنی فرکانس هایی با دامنه 3.65*10^(7). از نقاط روی شکل، فرکانس قطع پایین و بالا به ترتیب 698.088 و 697.941 هرتز است که نشان میدهد پهنای باند خیلی کم و حدود 0.147 هرتز داریم. اما بطور دقیقتر هم میتوان با دستور powerbw محاسبه کرد که در ادامه همین سکشن انجام شده است که به حدود ۳۲۱ میلی هرتز رسیده است:



:۱,۳

تصویر زیر به خوبی تفاوت این دو را نشان میدهد:

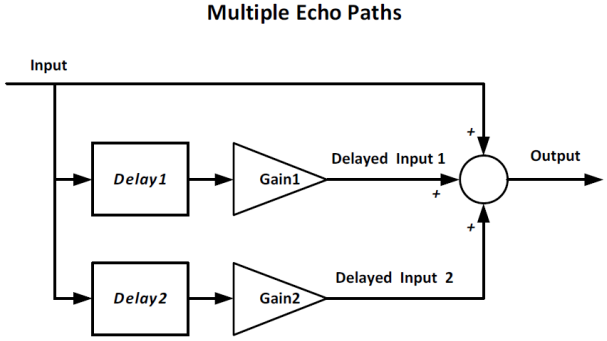


در واقع Echo بدین معناست که یک صدای مشخص، در فواصل زمانی معین، بطور کامل پخش شود و هربار، دامنه سیگنال شنیده شده ضعیف تر از قبل شود. وقتی یک پالس صوتی، دو یا چند بار به گوش انسان برسد، Echo رخ میدهد. البته باید توجه داشت برای تشخیص اکو توسط مغز، حداقل فاصله لازم بین دو پالس شنیده شده، باید ۵۰ میلی ثانیه باشد. این اتفاق معمولا بر اثر بازتاب سیگنال در اثر برخورد به موانع رخ میدهد.

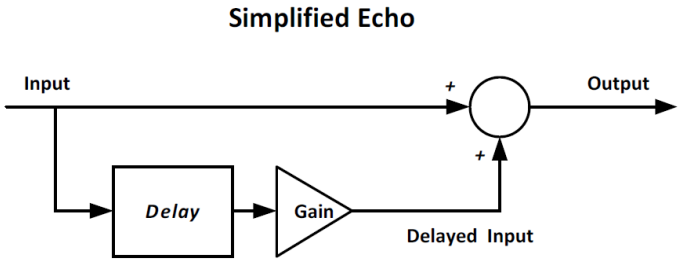
اما reverberation به معنای کشیده شدن صداست تا زمانی که دامنه آن آنقدر کم شود تا قابل شنیدن نباشد. این اتفاق، پس از قطع شدن صدا توسط منبع آن رخ میدهد. وقتی تعداد اصوات بازتاب شده بسیار زیاد باشد، مغز اصوات را به صورت پیوسته میشنود و این اتفاق می افتد.

:۱,۴

با توجه به توضیح بخش قبل، در echo ما علاوه بر ورودی همان لحظه، ورودی تضعیف شده ی لحظات قبل را نیز در خروجی دریافت میکنیم. این کار میتواند، بدلیل بازتاب صوت از موانع مختلف، از چند طرف رخ بدهد، مانند شکل زیر:



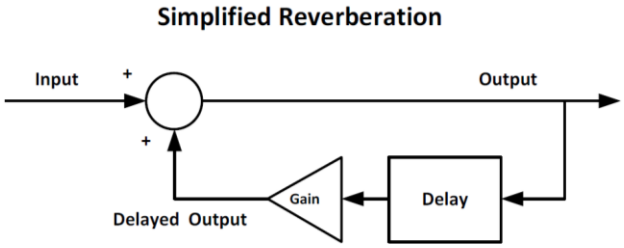
اما در حالت ساده تر میتوان فقط یک بازتاب را برای سیستم اکو تعریف کرد:



و در نتیجه معادله تفاضلی را میتوان به صورت زیر نوشت:

$$y[n] = g * x[n-1] + x[n] \ , \qquad g = Gain$$

در reverberation ، ما علاوه بر صدای ورودی، تضعیف شده ی خروجی های قبلی خود را نیز میشنویم. در واقع این بار، بجای تکرار شدن ورودی، خروجی است که پیوسته تکرار و البته تضعیف میشود تا ما صدای کشدار و پیوسته ای را بشنویم. پس دیاگرام به صورت زیر است:



و برای معادله تفاضلی داریم:

$$y[n] = x[n] + g * y[n-1] \ , g = Gain$$

:۱,۵

تابع با نام echo در سکشن آخر(مربوط به توابع) برای این بخش است. ورودی هایش اسم فایل صوتی (که طبیعتا باید در محل اجرای کد باشد) و تعداد مراحل انجام عمل اکو است و خروجی هایش، بردار سیگنال اکو شده و فرکانس نمونه برداری فایل صوتی ذکر شده است.

ابتدا با audioread سیگنال و فرکانس فایل را میخوانیم و در y,fs میریزیم (در ادامه y را به بردار سطری تبدیل میکنیم). بهره ای که در دیاگرام بالا بود را با gain تنظیم میکنیم. delayTime هم مشخص میکند که اکوی هر جزئ سیگنال، بعد از چه مدتی شنیده شود (توجه داریم در عمل و واقعیت، صدای اکو، دقیقا در لحظه بعدی صدای اصلی شنیده نمیشود، بلکه یک فاصله زمانی محسوسی با هم دارند). extendedSamplesSize هم با توجه به فرکانس نمونه برداری و مدت زمان فاصله بین صدا و اکوی صدا، تعیین میکند که چند sample جدید باید به سیگنال ما افزوده شود. مثلا اگر فرکانس ۴۴۱۰۰ هرتز داریم و فاصله صدا و اکویش ۰,۲ ثانیه است، تعداد نمونه های اضافه شده به اندازه حاصلضرب این دو عدد است. در این تابع ما قابلیت انجام چند مرحله ای اکو را هم داریم، یعنی سیگنالی که اکو شده، خود دوباره با همان گین اکو شود. تعداد مراحل اکو شدن در stages ریخته شده و به دلیل همین چند مرحله ای بودن، تعداد نمونه ها باید به اندازه stages* extendedSamplesSize زیاد شود. این کار در y_extended انجام شده است. برای ایجاد سیگنال نهایی y_eff ، یک y_shifted داخل حلقه فور تعریف کرده ایم. در هر مرحله ی i ام، این بردار، سیگنال ورودی را به

اندازه extendedSamplesSize *i شیفت میدهد و آن را در بهره gain^i ضرب میکند تا تضعیف مرحله i ام انجام شود. بردار نهایی، y_eff خواهد بود.

:۱,۶

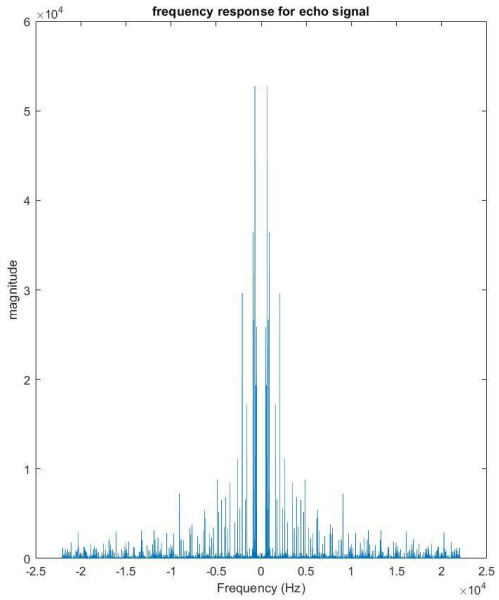
این تابع هم در بخش توابع با نام reverb نوشته شده است. اکثر کد مشابه تابع echo است فقط توجه داریم در این حالت ، اثر کشیدگی صدا خیلی زود در خروجی حس میشود. در واقع فاصله بین صدای اصلی و صدای تضعیف شده در مرحله بعد، باید انقدر کم باشد که شنونده احساس وجودیک پیوستگی در کاهش صدا بکند. به همین دلیل delayTime در این حالت کمتر از اکو است. به طرز مشابه قبل، تعداد sample اضافه شده در هر مرحله را در extendedSamplesSize میریزیم و y را در y_extended به اندازه نمونه های اضافه شده ی کل، extend میکنیم. این بار در حلقه فور، y_eff_shifted، برداری است که هربار در آخرین خروجی تا آن لحظه، یک gain ضرب میکند و خود بردار را نیز به اندازه یک مرحله، یعنی به تعداد extendedSamplesSize به جلو شیفت میدهد. خروجی جدید نیز در y_eff ، برابر با حاصل جمع این بردار با بردار ورودی است که این، کاملا مطابق با معادله تفاضلی و دیاگرام تعریف شده در بالا برای reverberation است. توجه داریم در این حالت، بدلیل کوتاه مدت بودن هر مرحله، باید تعداد مراحل زیادتر باشند (مثلا در حدود ۷۰ تا).

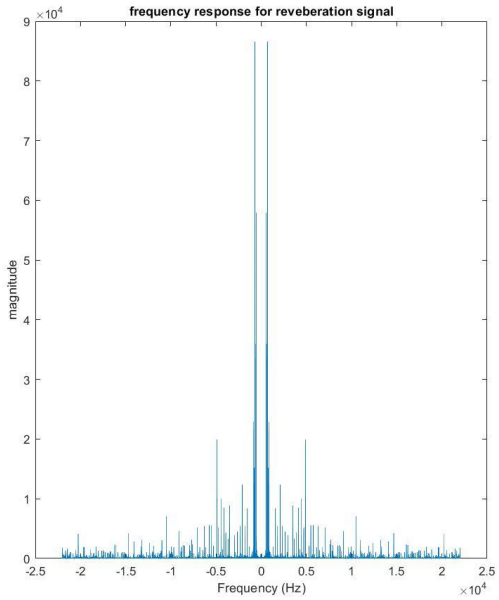
:۱,۷

در سکشن Q1.7، کد هردو نوشته شده است. در y_echo ، اثر echo و در y_reverb هم اثر reverberation لحاظ شده است و سپس فایل های مربوطه ساخته شده اند. برای نمونه، برای اکو ۳ مرحله و برای reverberation ۷۰ مرحله انجام داده ایم. در فایل اکو میبینیم که تکرار و اکو شدن صدا، به خصوص پس از قطع شدن صدای اصلی، کاملا حس میشود. اما در reverberation بجای تکرار صدا، صرفا صدا کشیده شده و با دامنه ای میرا به سمت صفر میرود (نویز های احتمالی به وجود آمده در صدا بخاطر جمع شدن سیگنال تضعیف شده ی قبلی با ورودی جدید هستند که طبیعتا اندکی نظم صدا را مختل میکنند اما اثر reverberation به خصوص در انتهای فایل صوتی، کاملا محسوس است).

:۱,۸

نحوه رسم پاسخ فرکانسی در بخش دوم سوال کاملا تئضیح داده شد. همان فرایند را در سکشن Q1.8 برای echo_y,reverb_y که در سکشن قبل بدست آمدند، پیاده میکنیم. حاصل به صورت زیر است:



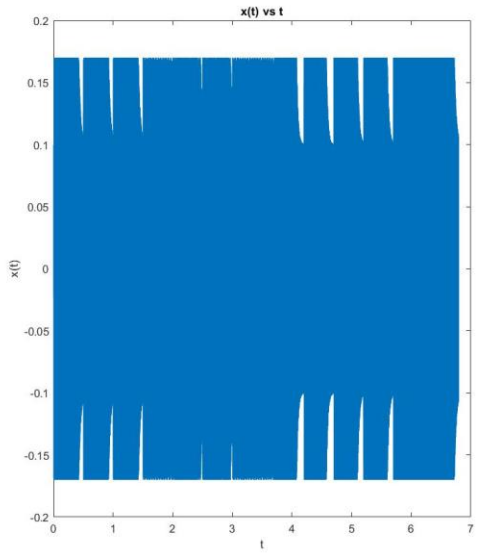


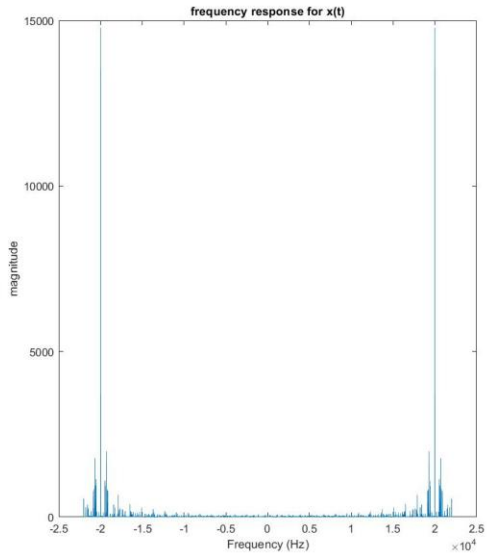
میبینیم که در مقایسه با آنچه در بخش دوم بدست آمده بود، بازه فرکانسی پخش تری داریم و فرکانس های بیشتری در این دو حالت، در تولید سیگنال نقش دارند، بطوریکه در نمودار، مقادیر 20kHz نیز دیده میشود. در واقع اعمال ورودی(خروجی) های زمان های قبل توسط feedback یا feedforward، موجب شده که از دامنه ی پاسخ فرکانسی در هر فرکانس کاسته شده و در عوض، فرکانس های بیشتری در سیگنال ایفای نقش کنند.

۲:

۱:

برای بازیابی سیگنال تولید شده در بخش قبل، فایل wav ساخته شده در سوال قبل در محل فایل متلب این سوال هم کپی شده است تا اطلاعات از روی آن خوانده شود. سیگنال خوانده شده را در m و فرکانس نمونه برداری شده را fs مینامیم. با توجه به تعداد نمونه ها و فرکانس نمونه برداری، t یا همان بازه زمانی نمایش سیگنال را میتوان مشخص کرد و سپس سیگنال x را بر حسب t رسم کرد. پاسخ فرکانسی نیز دقیقا به روشی مشابه با آنچه در بخش دوم سوال اول بود، بدست می آید. نتایج به صورت زیر است:

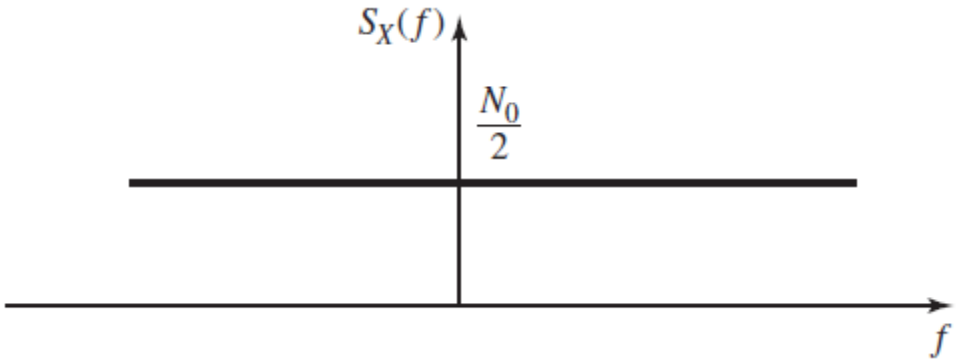




توجه داریم در سیگنال زمان، جاهایی که دامنه کم میشود و فاصله هایی داریم، همان فاصله بین رها کردن یک کلید و زدن کلید بعدی است که در آن دامنه به صورت نمایی کم میشد.

:۲,۲

نویز سفید به فرایند هایی گفته میشود که در آنها، طیف توان برای تمام فرکانس ها ثابت است. به بیان دیگر، تمام مولفه های فرکانسی با مقدار توان یکسانی در نویز سفید وجود دارند:

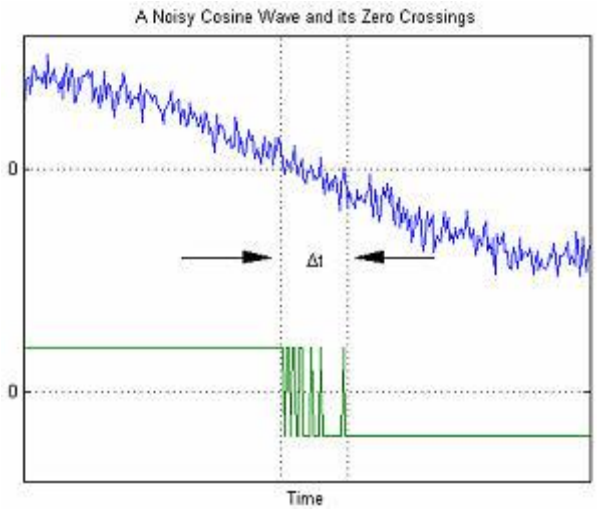


بنابراین طبق رابطه زیر، واضح است که مقدار توان فرایند سفید بینهایت است:

$$P_X = \int_{-\infty}^{\infty} S_X(f)df = \int_{-\infty}^{\infty} Cdf = \infty$$

نویز سفید گوسی جمع شونده، بدلیل بالا سفید نامیده میشود. جمع شونده بودن بدلیل جمع شدن آن با سایر نویز ها و گوسی بودن آن نیز بدلیل توزیع گوسی این نویز در طول زمان میباشد. مطابق توضیحات بالا، اثر این نویز در

سیگنال، جمع شدن یک مقدار ثابت در طیف فرکانسی است. این نویز برای مدل کردن نویز های ناشی از RJ(random jitter) کاربرد دارد. در شکل زیر یک سیگنال کسینوسی نویزی و یک نویز سفید گوسی را میبینیم:



وقتی در حوالی بازه Δt ، پس از جمع شدن با نویز سفید، در واحد زمان، تعداد نقاط بالا رونده با پایین رونده برابر است و رابطه زیر را داریم:

$$\frac{\text{positive zero crossings}}{\text{second}} = \frac{\text{negative zero crossings}}{\text{second}} = f_0 \sqrt{\frac{\text{SNR} + 1 + \frac{B^2}{12 f_0^2}}{\text{SNR} + 1}},$$

where

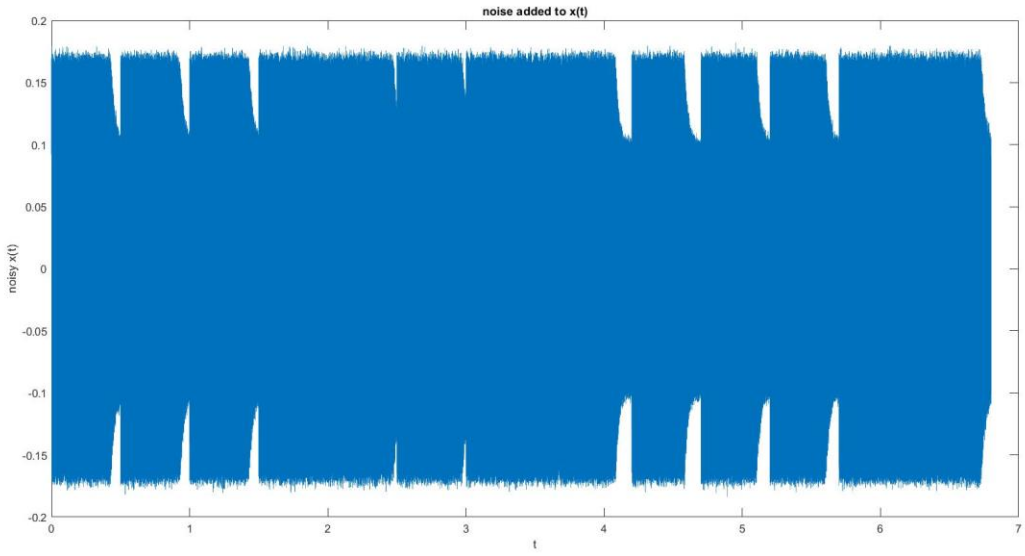
f_0 = the center frequency of the filter,

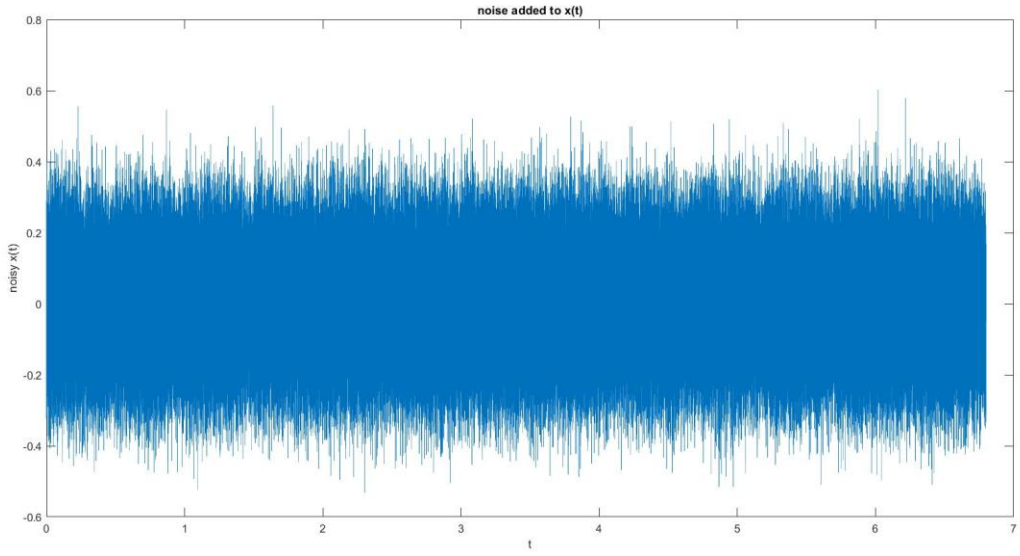
B = the filter bandwidth,

SNR = the signal-to-noise power ratio in linear terms.

:۱,۳

این کار در سکشن Q2.3 انجام شده است. در بالا هم گفته شده که SNR در واقع نسبت توان سیگنال، به توان نویز به ازای ترم های خطی است. با این تعریف طبیعتا هرچه این پارامتر در دستور awgn کمتر باشد، نویز به نسبت سیگنال ما سهم بیشتری داشته و سیگنال را دچار تغییرات شدیدتری میکند. اما در SNR های بالاتر، تاثیر نویز روی سیگنال چندان قابل مشاهده نیست. در رابطه نوشته شده در بخش قبل هم میبینیم که اگر SNR زیاد باشد، تعداد نقاط بالا رونده و پایین رونده در واحد زمان، با هم برابرند و به فرکانس اصلی سیگنال میل میکنند(f0). در زیر تاثیر SNR روی سیگنال را، یکبار با مقدار ۵۰ و یکبار با مقدار ۲۰ میبینیم که به وضوح، مقدار ۲۰ سیگنال را دچار تغییرات اساسی میکند:

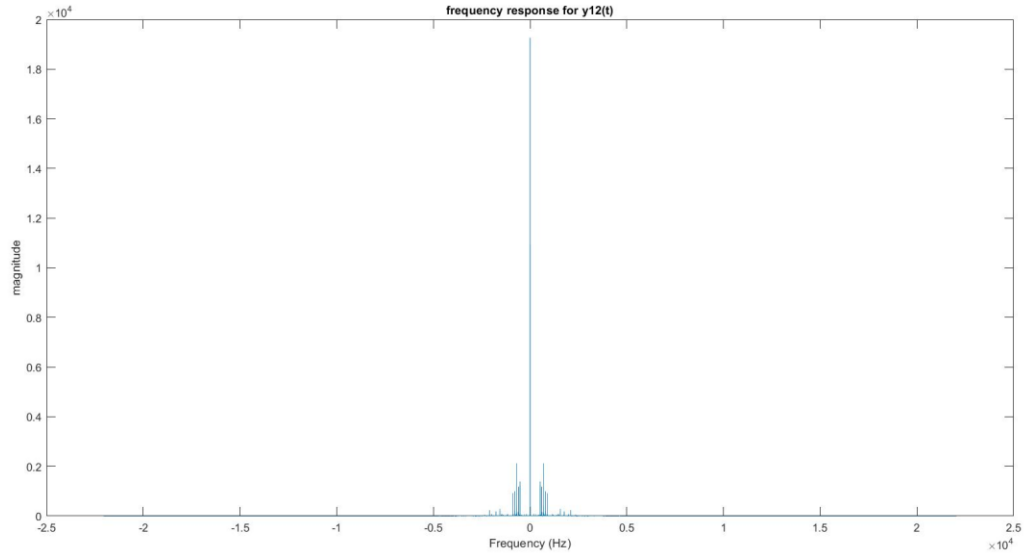




:۱,۴

مطابق دیاگرام سوال، ابتدا سیگنال نویزی را مجددا ساخته و در مرحله اول، اندازه آن را میگیریم. سپس مجددا با powerbw پهنای باند نرمالیزه شده را اندازه گرفته و در wpass ریخته ایم. حال با lowpass و پهنای باند wpass، میتوان y2 را ساخت. فیلتر پایین گذر موجب حذف مقدار خوبی از نویز های فرکانس بالا میشود و تا حد خوبی به سیگنال اولیه میرسیم. اما اگر مقدار SNR از حدی کمتر شود، حذف نویز های شدید بسیار سخت خواهد بود و دیگر صدای اصلی قابل تشخیص نیست. در فایل های با نام ۱ و ۲ و ۳، این کار را به ازای SNR برابر با ۴۰، ۲۰ و ۱۵ انجام داده ایم. مقادیر قدر مطلق

گرفته شده در yi1 ها و مقادیر نهایی در yi2 ها هستند. به ازای ۱۵ دسیبل به پایین، تقریبا سیگنال اصلی حس نمیشود و در حدود ۶۰ دسیبل هم مشکلی در شنیدن صدای اصلی نداریم. سیگنال به ازای ۴۰ دسیبل را در حوزه فرکانس رسم میکنیم:



گزارش سوال ۱_۳

در ابتدا با استفاده از دستور clear ,clcضمن پاک کردن اطلاعات قبلی command window را پاک می کنیم . سپس با استفاده از دستور imortdata داده ها را می خوانیم و در dataذخیره می کنیم . با توجه به فرکانس نمونه برداری طول سیگنال 5 ثانیه ای 1250 سمپل است پس به این تعداد از اول و آخر سیگنال کلی حذف می کنیم تا استراحت ابتدا و انتهای آن حذف شود و در withoutRestData ذخیره می کنیم .

بخش دوم :

داده هارا با استفاده از تابع zscore نرمالیزه می کنیم . عملکرد این تابع به این صورت است که عملیات نرمالیزه کردن را برروی ستون های ماتریس ورودی انجام می دهد . این عملیت در حالت تک ورودی که مطلوب ماست به صورتی انجام می پذیرد که میانگین ستون(بردار) مورد بررسی صفر شود (البته این صفر شدن در عمل به شکل دقیق اتفاق نمی افتد و در نهایت میانگین هر ستون در حدود صفر است .) همینطور واریانس هر ستون بر روی یک تنظیم می شود . همینطور لازم به ذکر است که از نظر بزرگی ترتیب داده ها به هیچ عنوان عوض نمی شود . با استفاده از ورودی های دیگر این تابع می توان بعذی نرمالیزه شدن بر آن اجرا می شود و مشخصات دیگر تبدیل را عوض کرد که هدف این سوال نیست . برای ارضا خواسته سوال لازم است ZSCORE را به شکل مستقیم بر روی دیتا خود پیاده کنیم چرا که دیتا ۱۹ ستون دارد که هر یک نماینده یک کانال است . خروجی را در normalizeddata ذخیره می کنیم سپس دو سیگنال restData و taskData تعریف می کنیم که قرار است مقدار نهایی آن ها مقدار میانگین استراحت و آزمایش باشد . از آنجا که بازه های استراحت و آزمایش کم است بدون زدن حلقه for به شکل مستقیم میانگین می گیریم .

بخش۳ :

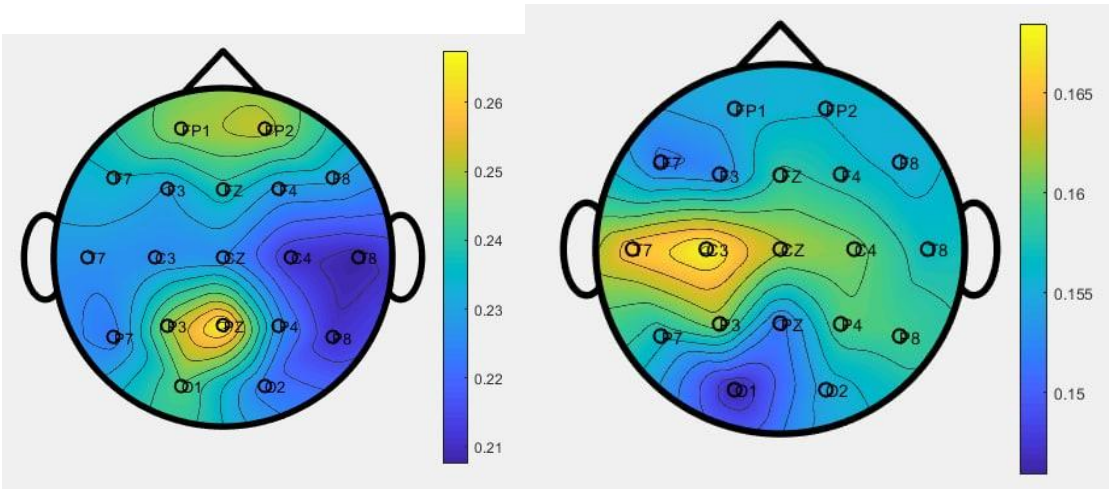
ابتدا با دستور lengthطول سیگنال های استراحت و آزمایش را به شکل دقیق به دست می آوریم . سپس بردار های sigma restو sigma taskتعریف می کنیم . درایه های این دو قرار است توان دو درایه های meanRest و meanTask باشد . برای این کار از ضرب *. استفاده می کنیم و هر بردار را با این ضرب در خودش ضرب می کنیم این کار باعث می شود درایه های هم اندیس در هم ضرب شوند بنابراین توان دو به این شکل ساخته می شود .حال کافی است جمع ستونی را بر روی این بردار ها اعمال کنیم تا sigmaخواسته شده به دست اید برای این کار از تابع sum استفاده می کنیم و به شکل sum(...,1) آن را قرار می دهیم تا جمع ستونی انجام شود . در نهایت حاصل ر بر طول سیگنال ها تقسیم می کنیم و بدین صورت Prest,Ptask به دست می آید .

بخش۴:

ابتدا channel_title را همانطور که در صورت سوال آمده تعریف می کنیم برای اینکه این تعریف ورودی مناسبی برای تابع باشد باید به چند نکته توجه کنیم .1 . چنل هایی که دو حرفی هستند باید حروف بزرگ داشته باشند . 2 . هیچ نوع فاصله ای در STRING ها نباید وجود داشته باشد . در نهایت مقدار به وجود آمده از نوع CELL است بنابراین در هنگام ورودی دادن می توان از cellstr استفاده کنیم تا ورودی مناسب داده شود .

بخش۵:

مشاهده می شود که در بخش استراحت نواحی مربوط به چشم و سمت عقبی مغز فعالیت بیشتری دارند . ذر حالی که در آزمایش دوم نواحی مربوط به گوش سمت چپ فعال تر ظاهر می شوند . همینطور مشاهده می شود که میانگین توان سیگنال های مغز در حالت استراحت از حالت آزمایش بیشتر است (شاید بتوان اینطور برداشت کرد که فعالیت آزمایش که یک فعالیت شنیداری بوده انرژی کم تری از بخش استراحت که می تواند فعالیت بینایی باشد دارد . یا شابه علت ناقص بودن الکترود ها اطلاعات دقیق نیست).



گزارش سوال ۲_۳

بخش۱:

مشابه قسمت قبل از تابع importData برای خواندن دیتای مربوطه که در فایل قرر دارد استفاده می کنیم .

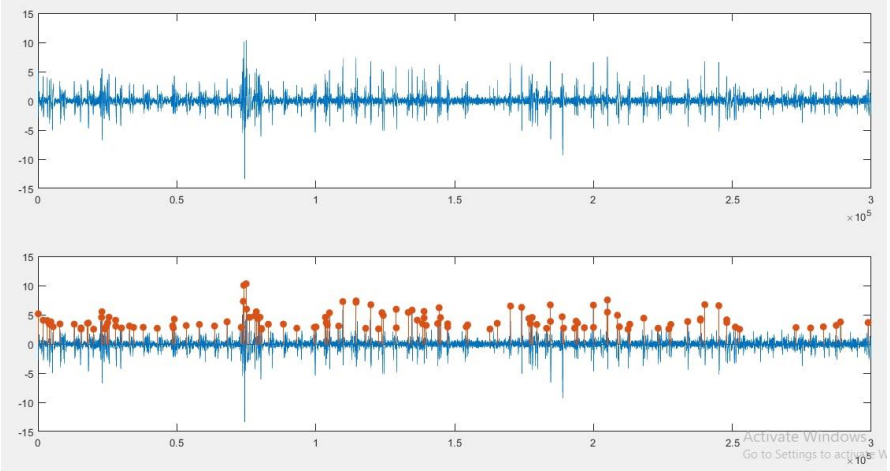
بخش۲:

ابتدا سایز دیتا را به دست می آوریم . سپس تعداد گام های ۵۰۰ تایی که باید برای تحلیل سیگنال لازم داریم را به دست می آوریم . این کار را با تقسیم کردن سایز بر ۵۰۰ انجام می دهیم . لازم به ذکر است که این روش برای زمانی است که ما بخش پذیر بودن تعداد را برا ۵۰۰ می دانیم . اگر اینطور نباشد باید یک دور تقسیم انجام دهیم و سپس مقدار باقی مانده را بر ۵۰۰ بیابیم و اگر این باقی مانده غیر ۰ بود یک پنجره از خانه فعلی تا خانه آخر تعریف کنیم . در نهایت با یک forبه تعداد گام های به دست آمده داده را پیمایش می کنیم . در داخل حلقه با استفاده از max جایی که در این درایه ۵۰۰ تایی max اتفاق افتاده را به همراه مقدار آن به دست می آوریم ذر یک حلقه ifبررسی می کنیم که آیا مقدار max از مقداری که باید برای تشخیص پلک زدن از آن بیشتر باشد بیشتر است یا نه . حال لازم است به تعریف دو داده xblinkو yblink توجه کنیم . xblink در ابتدا به صورت یک داده یک در یک صفر تعریف می شود تا بتوانیم از size آن به عنوان یک داده در داخل for استفاده کنیم . از آنجا که این داده به ازای پلک زدن اول از نظر سایز اطلاعات غلط به ما می دهد در یک if خانه اول را چک می کنیم و اگر صفر بود سایز را صفر در نظر می گیریم . (غیر ممکن است مکان صفر داشته باشیم .)درنهایت درایه ای که باید در آن مقادیر قرار بگیرد با استفاده از جمع زدن سایز با یک به دست می آید y پلک برابر با ماکس و x آن برابر با محل وقوع ماکس در سیگنال پرمایش شده ۵۰۰ تایی به علاوه مقداری است که تا به حال پردازش شده یعنی 500*(i-1) . تعداد پلک هم با سایز گرفتن از xblink به دست می آید .

بخش۳:

با استفاده از subplot , figure را به دو بخش در راستای عمودی تقسیم می کنیم . بخش اول نمایش خود دیتا با استفاده از plot است بخش بعدی شامل خود دیتا به علاوه stem xblink,yblink است که با استفاده از دستور hold on این دو را روی هم می اندازیم . همینطور stem را بر روی filled تنظیم می کنیم تا نمودار مشابه با خواسته سوال شود .

نکته : بخش های این سوال با script جدا شده است بنابراین لازم است به ترتیب اجرا شود .


15
10
5
0
-5
-10
-15
0
0.5
1
1.5
2
2.5
3
×10^5
15
10
5
0
-5
-10
-15
0
0.5
1
1.5
2
2.5
3
×10^5
Activate Windows
Go to Settings to activate W