به نام خالق دانایی

گزارش پروژه درس سیگنال و سیستم – فاز یک

سروش مومنی (402102538)– محمد مهدی انصاری (402101306)

1- نویز و سیگنال

- نویز های معرفی شده : (تصاویر نویز ها در حوزه گسسته زمان و فرکانس با کمک از نرم افزار متلب تولید شده اند)
- **نویز گوسی :**

طبق توضیحات داده شده می توان نویز گوسی را در حوزه گسسته زمان به صورت زیر مدل سازی کرد :

$Y[n] = x[n] + w[n]$

که در آن $y[n]$ سیگنال مشاهده شده است و $x[n]$ سیگنال اصلی و $w[n]$ سیگنال نویز با توزیع زیر می باشد :

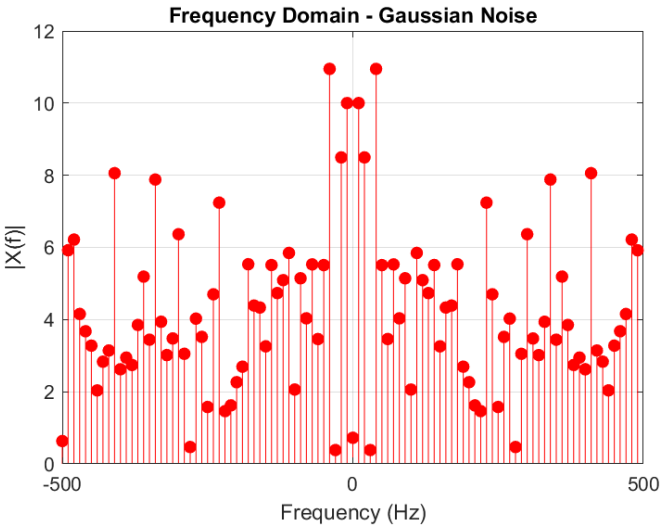
$$w[n] \sim N(0, \sigma_2)$$

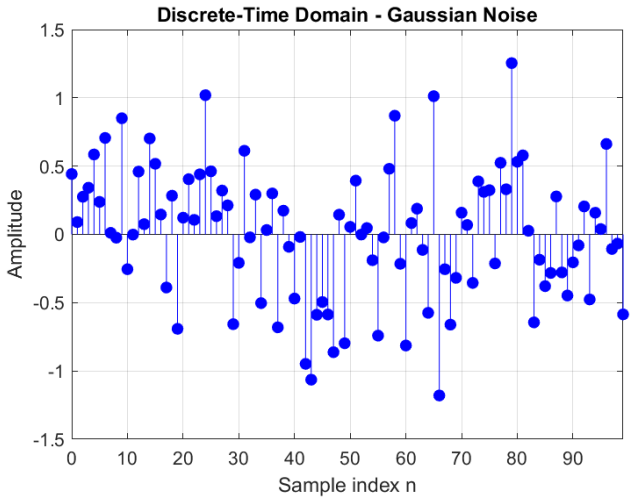
می دانیم که σ^2 نشان دهنده واریانس توزیع و دراصل قدرت نویز می باشد و $w[n]$ دارای میانگین صفر است .

نمونه ها در حوزه گسسته بدون همبستگی هستند یعنی : $E[w[n]w[m]]=0$ for $n \neq m$

اگر این نویز با عرض باند خود (W) و چگالی توان خود (N_0) بیان شود ، واریانس نویز را می توان از رابطه زیر استخراج نمود :

$$\sigma^2 = N_0 W/2$$





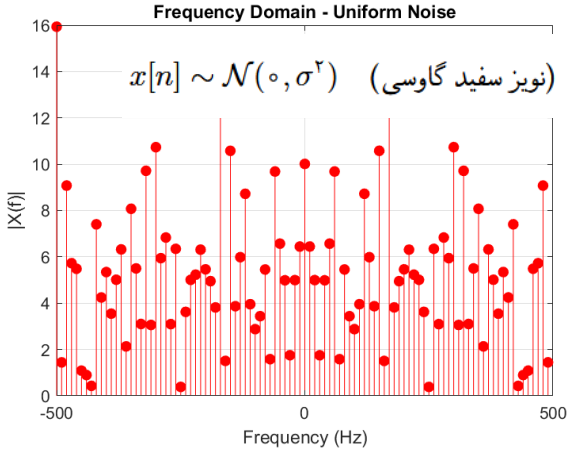
- **نویز یکنواخت :**

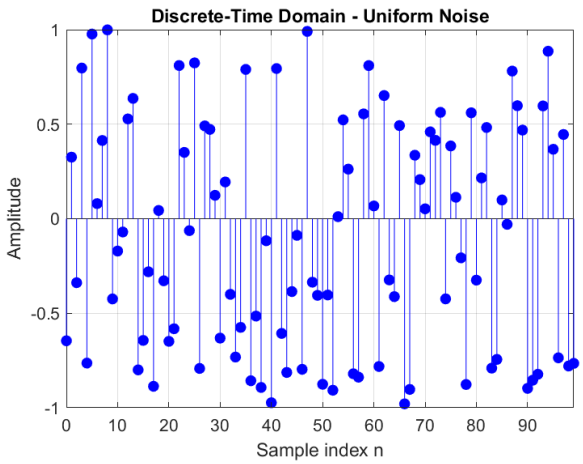
هر نمونه نویز n[k] به صورت مستقل ، از توضیح یکنواخت در بازه [-A,A] انتخاب می شود و به صورت رو به رو فرمول بندی می شود .

$$n[n] \sim U(-A,A)$$

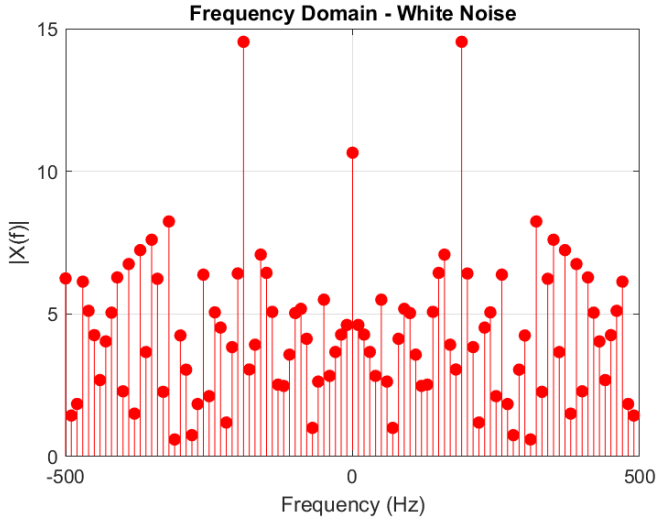
- **نویز سفید گوسی :**

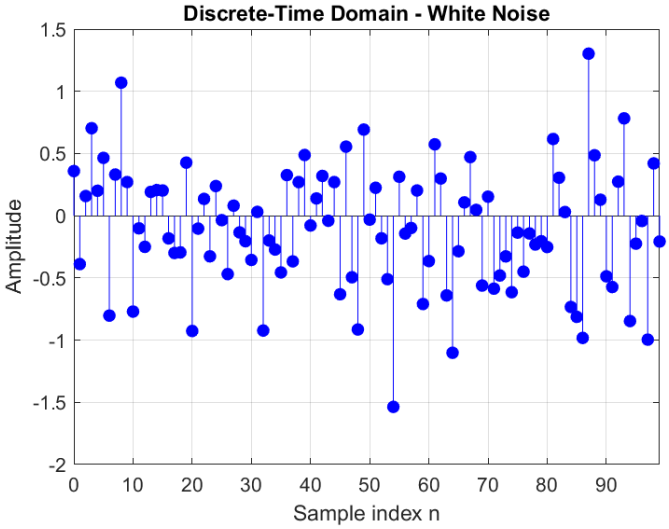
همانند نویز سفید فرمول بندی می شود و معمولا با





واریانس یک است ، همه فرکانس هارا شامل می شود و دارای چگالی توان ثابت (واحد) می باشد :

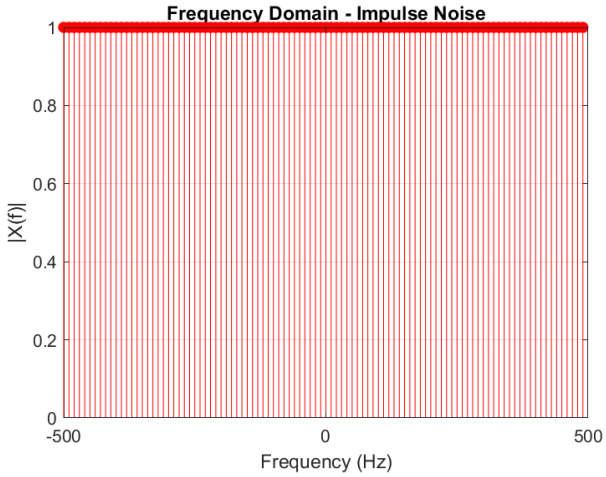


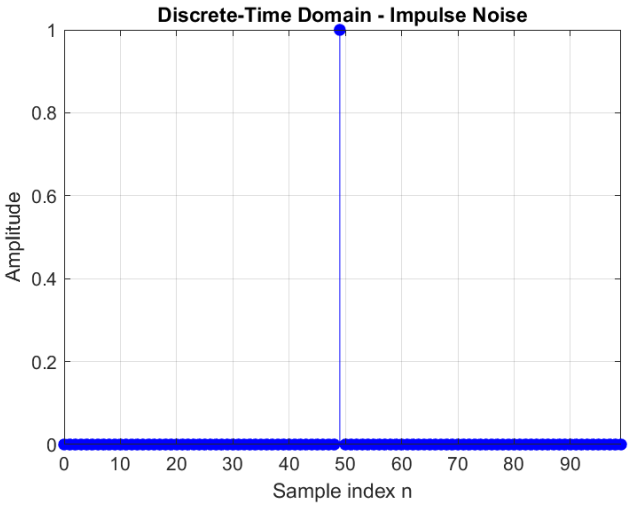


- **نویز ضربه :**

به صورت یک ضربه واحد در زمان n=0 تعریف می شود :

$$n[n] = \delta[n] = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{otherwise} \end{cases}$$





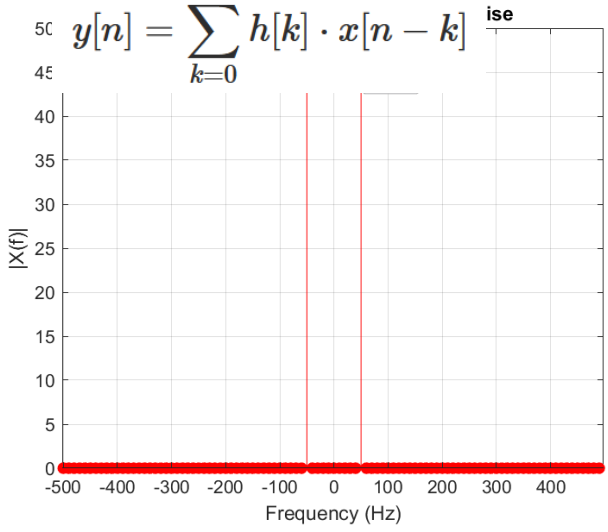
- **نویز پاور لاین :**

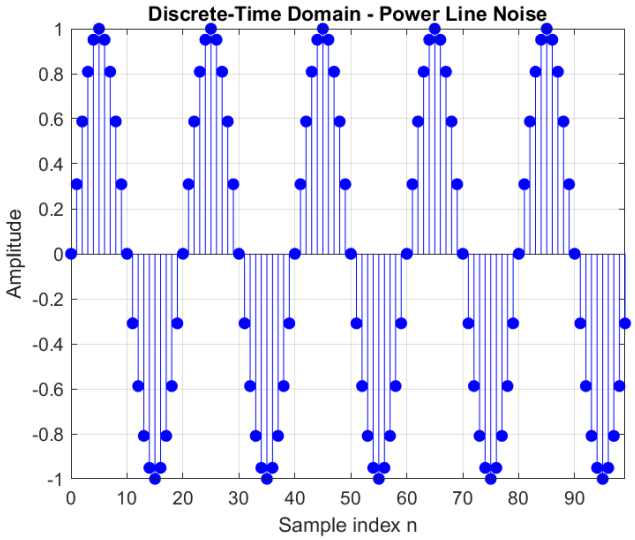
$$n[n] = A \cdot \sin(2\pi f_d n + \phi)$$

به صورت رو به رو در حوزه گسسته زمان فرموله می شود :
در این فرمول fd فرکانس نرمالیز شده است که به صورت fd = f/fs می باشد که در اینجا fs فرکانس نمونه برداری است . بدیهی است که ϕ فاز اولیه است که به صورت تصادفی تعیین می شود : تصویر پایین به ازای نویز برق شهری 50 هرتز است :

- **فیلتر ها :**

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$





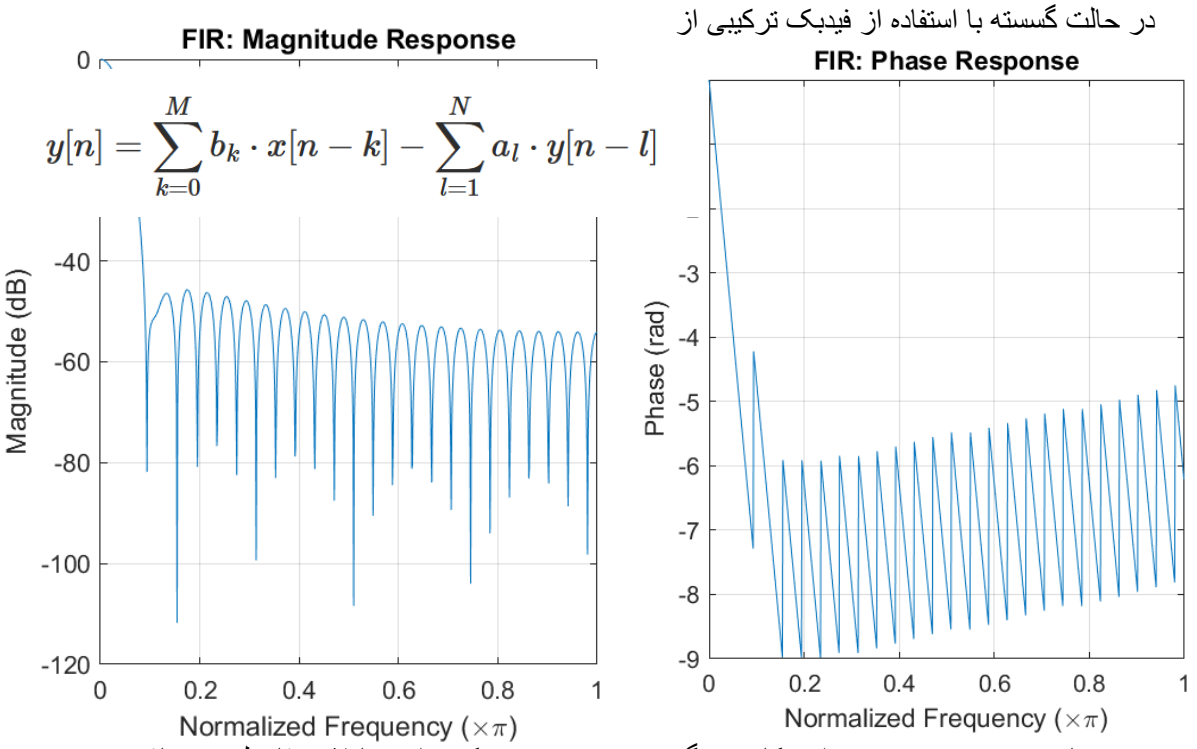
- **پاسخ ضربه محدود : فیلتر FIR**

در این فیلتر از تعداد محدودی از نمونه های ورودی گذشته برای محاسبه خروجی استفاده می کند . در اینجا h[k] پاسخ ضربه فیلتر است و x[k] ورودی فیلتر می باشد و N مرتبه فیلتر می باشد .

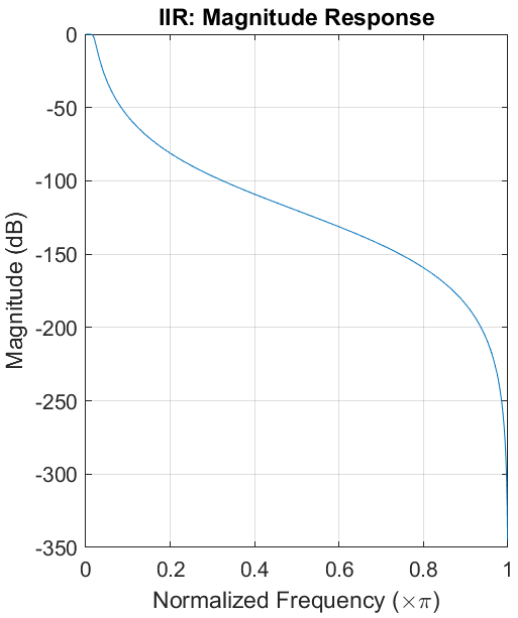
از مزایای این فیلتر می توان به داشتن پاسخ فاز خطی و همچنین پایداری سیستم اشاره کرد . این فیلتر پیاده سازی و طراحی ساده ای دارد . می توان با گذاشتن پنجره و یا نمونه برداری در حوزه فرکانس طراحی شود .
اما این فیلتر در نویز هایی با پهنای باند پایین عملکرد ضعیفی از خود نشان می دهد .

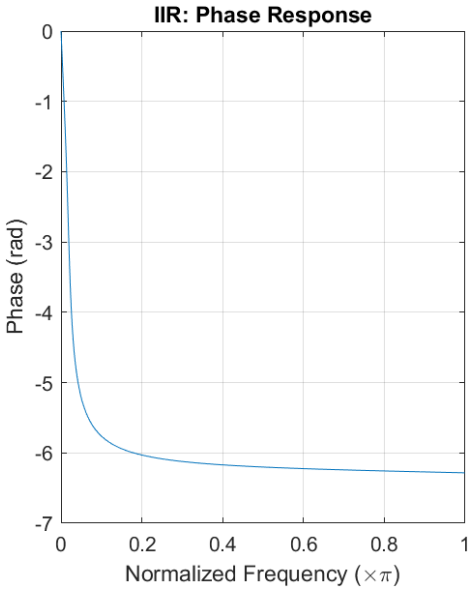
- **پاسخ ضربه نامحدود : فیلتر IIR**

در حالت گسسته با استفاده از فیدبک ترکیبی از



نمونه های ورودی و خروجی را به کار می گیرد و به صورت ترکیبی از معادلات تفاضلی می باشد .
B_k به عنوان ضرایب ورودی ها و a_l به عنوان ضرایب خروجی ها و یا فیدبک بیان می شود .
از مزایای این فیلتر می توان اشاره به نیاز به ضرایب کمتر برای رسیدن به کارایی مشابه و یا بالا تر نسبت به fir دانست .
این فیلتر معایبی همچون داشتن پاسخ فاز غیر خطی که سبب اعوجاج می شود و یا امکان وارد شدن سیستم به حالت ناپایداری به دلیل وجود فیدبک ، را دارا است .





- **فیلتر تبدیل ویولت : wavelet transform**

برای حذف نویز با تبدیل ویولت گسسته DWT به طور کلی باید سه مرحله را دنبال کنیم :

تجزیه سیگنال با DWT ، اعمال آستانه گذاری با توجه به ضرایب ، بازسازی سیگنال اصلی دنویز شده با IDWT :

ابتدا فرض کنید h[n] پاسخ ضربه یک فیلتر پایین گذر و g[n] پاسخ ضربه یک فیلتر بالا گذر می باشد .

حال ضرایب را به این صورت تعریف می کنیم :

به a[k] ضرایب تقریب و به d[k] ضرایب جزئیات می گویند .

$$a[k] = \sum_{} x[n] \cdot h[2k - n]$$

$$d[k] = \sum_{n=-\infty}^{\infty} x[n] \cdot g[2k - n]$$

حال روی ضرایب جزئیات آستانه گذاری می کنیم تا نویز را حذف کنیم (با توجه به مقدار آستانه T) :

آستانه گذاری به دو حالت نرم و سخت انجام می شود :

آستانه گذاری نرم :

$$d_{\text{th}}[k] = \begin{cases} \text{sign}(d[k]) \cdot (|d[k]| - T), & \text{if } |d[k]| \geq T \\ 0, & \text{if } |d[k]| < T \end{cases}$$

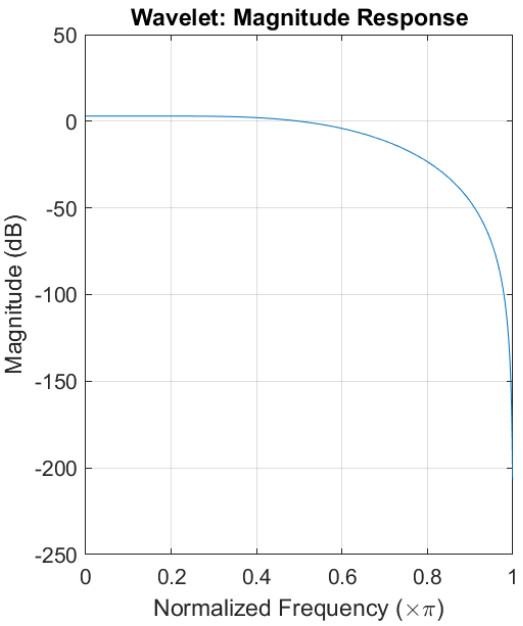
آستانه گذاری سخت :

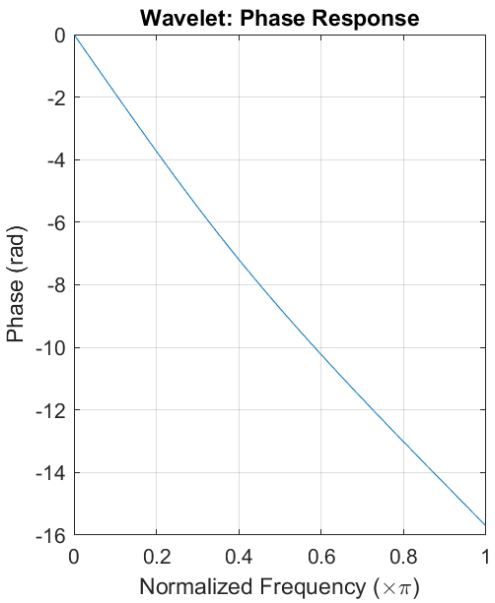
$$d_{\text{th}}[k] = \begin{cases} d[k], & \text{if } |d[k]| \geq T \\ 0, & \text{if } |d[k]| < T \end{cases}$$

حالا با تبدیل معکوس ویولت سیگنال را بازسازی می کنیم :

$$x[n] = \sum_k a[k] \cdot \tilde{h}[n - 2k] + \sum_k d[k] \cdot \tilde{g}[n - 2k]$$

در مقایسه با فیلتر های FIR و IIR این فیلتر در حذف نویز های لحظه ای و غیر ایستا قوی تر عمل می کند و با حداقل آسیب به سیگنال نویز را در لبه ها حذف می کند اما پیچیدگی محاسباتی بسیار بالایی نسبت به بقیه دارد .





2- سیگنال های ویژه :

سیگنال های ویژه مجموعه ای از سیگنال هایی می باشند که طبق تعریف نسبت به هم اورتوگونال هستند . این سیگنال های پایه ای برای نمایش سایر سیگنال ها به کار می روند . سیگنال اصلی می تواند به صورت ترکیب خطی از سیگنال های ویژه بیان شود در پردازش سیگنال ، سیگنال های ویژه را برای موارد زیر استفاده می کنند :

- نمایش فشرده تر سیگنال ها به صورتی که انرژی حامل سیگنال با تعداد کمتری مولفه بیان می شود
- کاهش ابعاد داده ها در فشرده سازی تصاویر
- بهبود عملکرد حذف نویز با تفکیک مولفه های سیگنال از نویز
- از طرفی ساده سازی سیگنال های پیچیده به مولفه های مستقل و با همبستگی صفر ، پردازش را تا حد امکان ساده می کند .

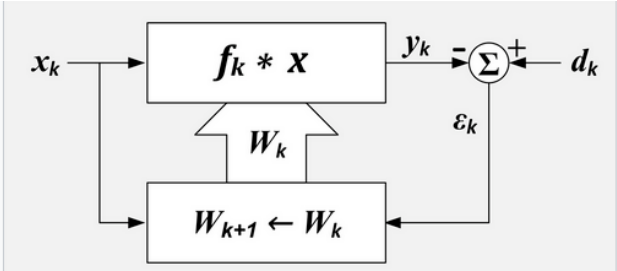
برای مثال در روش PCA(principal component analysis) سیگنال های اصلی به مولفه های تبدیل می شوند که به آن ها سیگنال های ویژه می گویند ، این سیگنال ها جهت هایی هستند که بیشتر تغییرات داده را به صورت واریانس نشان می دهند .

در فشرده سازی تصاویر ، سیگنال های PCA پایه هایی هستند که تصویر روی آن ها نمایش داده می شود . از این روش برای الگوریتم های تشخیص چهره هم استفاده می شود .

3- فیلتر تطبیقی : Adaptive Filtering

- در فیلتر تطبیقی ، ضرایب فیلتر بر حسب الگوریتم تعریف شده برای فیلتر به روز رسانی می شوند تا خطا بین سیگنال ورودی و خروجی مطلوب به حداقل برسد .

در این جا از الگوریتم LMS یعنی کمترین خطای مربعات برای تعیین خطا بین سیگنال ورودی و خروجی مطلوب به حداقل برسد .



شکل روبه رو بلوک دیاگرام فرایندی که در بالا توضیح داده شد را نشان میدهد .

x[n] : سیگنال ورودی

d[n] : سیگنال مورد انتظار

W[n] : ضرایب فیلتر

y[n] : خروجی فیلتر که به صورت : y[n] = $w^T[n]x[n]$ تعریف می شود .

e[n] : سیگنال ارور که به صورت e[n] = d[n] – y[n] تعریف می شود .

بر اساس الگوریتم کمترین خطای مربعات ، ضرایب فیلتر به صورت زیر آبدیت می شوند :

W[n+1] = w[n] + ue[n]x[n] می باشد . به u ضریب یادگیری می گویند .

سیگنال های ویژه (بردار های ویژه ماتریس همبستگی و یا مولفه های pca که قبلا توضیح داده شد) ورودی را به حوزه ای می برند که اجزای سیگنال ناهمبسته هستند و این کار باعث می شود که فیلتر تطبیقی سریع تر همگرا شود و اطلاعات مفید با نویز تداخل کمتری داشته باشد .

- **تعیین loss function :**

همانطور که بیان شد ، فیلتر تطبیقی در هر مرحله بر اساس نتیجه loss function ضرایب فیلتر را به روز رسانی میکند و از این جهت این تابع از اهمیت بالایی برخوردار هست ، برخی از loss function های معروف عبارتند از :

- **Least Mean Squares (LMS) ، پرکاربرد ترین**

این تابع از نظر ریاضیاتی به صورت زیر تعریف می شود :

$$L(e[n]) = \mathbb{E}[e^2[n]]$$

که در آن ارور به صورت روبه رو تعریف می شود :

$$e[n] = d[n] - y[n]$$

حال ارور را مینیمایز می کنیم و ضرایب فیلتر را به صورت زیر و با توجه به ضریب یادگیری به روز رسانی میکنیم :

$$w[n+1] = w[n] + \mu\, e[n]\, x[n]$$

- **Normalized LMS (NLMS)**

در این تابع ارور همانند LMS عادی تعریف می شود ولی در خط آبدیت ، به صورت زیر ، ضرایب نرمالیز شده آبدیت می شوند . δ یک عدد ثابت برای جلوگیری از صفر شدن مخرج می باشد . از این تابع برای زمانی که سیگنال ورودی چگالی توان متغیر با زمان دارد استفاده می شود .

$$w[n+1] = w[n] + \frac{\mu}{x[n]\|^2 + \delta}\, e[n]\, x[n]$$

- **Recursive Least Squares (RLS)**

Loss function به صورت رو به رو تعریف می شود :

$$L = \sum_{k=0}^{n} \lambda^{n-k} e^2[k]$$

که در آن λ عددی بین صفر تا 1 به عنوان ضریب فراموشی بیان می شود .

این تابع ضرر دارای همگرایی سریع تری نسبت به LSM است اما پیچیدگی محاسباتی بیشتری دارد .

- **Huber Loss (Robust M-estimator)**

Loss function به صورت زیر تعریف می شود :

$$L(e) = \begin{cases} \frac{1}{2}e^2, & \text{if } |e| \le \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right), & \text{if } |e| > \delta \end{cases}$$

این تابع برای مقاومت در برابر داده های پرت و دور از میانگین مناسب است همچنین برای نویز های غیر گوسی و نویز های ناگهانی و شدید عملکرد خوبی دارد .

- **مزایای استفاده از فیلتر تطبیقی : Adaptive Filtering**

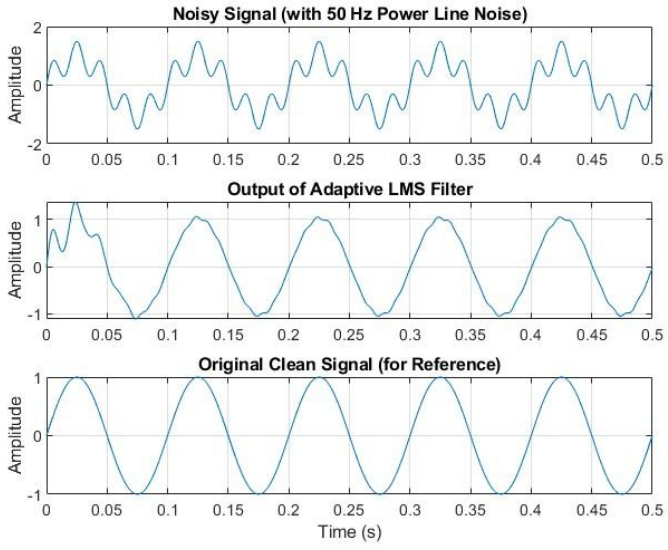
فایده اصلی استفاده از فیلتر تطبیقی این است که می تواند به صورت در لحظه و پویا (real- time) ساختار فیلتر را با تغییرات سیگنال و نویز سیگنال تطبیق دهد . بر خلاف فیلتر هایی مثل irr و fir یا ناچ فیلتر که بعد از طراحی ضرایب آن ها ثابت می ماند ، همانطور که قبلا بیان شد ، فیلتر های تطبیقی می توانند ضرایب خود را به طور مداوم بر اساس خطای تعریف شده بین سیگنال خروجی و سیگنال مورد انتظار تطبیق دهند و در محیط های غیر قابل پیش بینی عملکرد موثری داشته باشند .

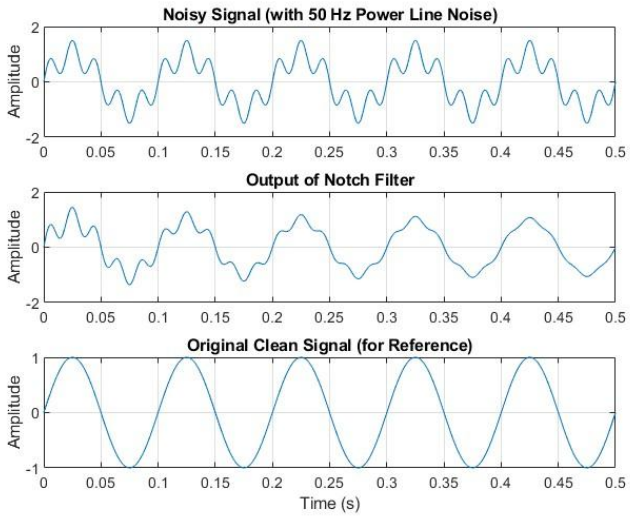
برای مثال خاص نویز برق شهر 50 هرتز ، عملکرد ناچ فیلتر را با عملکرد فیلتر تطبیقی مقایسه می کنیم :

در روش اول که از ناچ فیلتر ساده استفاده می کنیم ، این روش فقط زمانی خوب عمل می کند که نویز کاملا سینوسی باشد و فرکانس آن در طی زمان تغییرداشته باشد . اما میدانیم دامنه نویز شبکه متغیر است هر چند فرکانس آن در طی زمان تقریبا در یک حدود می باشد .

در روش دوم که از از فیلتر تطبیقی استفاده می کنیم ابتدا باید یک سیگنال مرجع مثلا موج سینوسی 50 هرتز به فیلتر بدهیم و تابع ضرر LMS را مینیمم کنیم . این فیلتر نویز را تخمین می زند و از سیگنال اصلی کم می کند . در این روش حتی با تغییر فرکانس و دامنه در طی زمان ، کنسل کردن نویز به خوبی انجام می شود و از تخریب اجزای مفید سیگنال جلوگیری می کند .

تصاویر زیر در نرم افزار متلب تولید شده است : مقایسه عملکرد فیلتر ناچ ساده و فیلتر تطبیقی در حذف نویز 50 هرتز برق شهری :





- **همگرایی در فیلتر های تطبیقی :**

در فیلتر های تطبیقی همگرایی به معنی نزدیک شدن ضرایب فیلتر به مقادیر بهینه ای است که خطای بین خروجی فیلتر و سیگنال مطلوب را به حداقل می رساند . تنظیم ضرایب فیلتر توسط الگوریتم های بهینه سازی مانند گرادیان کاهش (GD) و یا گرادیان کاهشی تصادفی (SGD) انجام می شود .

- **گرادیان کاهشی : (gradient descent)**

در این الگوریتم از گرادیان تابع ارور برای یافتن مینیمم خطای تعریف و یافتن مسیر نزولی استفاده می کنند و در هر مرحله ضرایب فیلتر را به سمت کاهش خطا به روز رسانی می کنند .

$$w(k+1) = w(k) - \mu \nabla J(w(k))$$

در فرمول بالا w^k ضرایب فیلتر در گام k می باشد . و u ضریب یادگیری است و گرادیان تابع هزینه می باشد .

$$\nabla J(w(k))$$

که در آن محاسبه گرادیان بر اساس کل داده ها N نقطه انجام می شود :

$$\nabla J(w) = -\frac{1}{N} \sum_{i=1}^{N} e(i)\, x(i)$$

- **گرادیان کاهشی تصادفی(تخمینی) : (Stochastic gradient descent)**

ابتدا یک نمونه تصادفی را انتخاب می کنیم : (x(i),d(i))

سپس گرادیان لحظه ای را به صورت رو به رو محاسبه می کنیم :

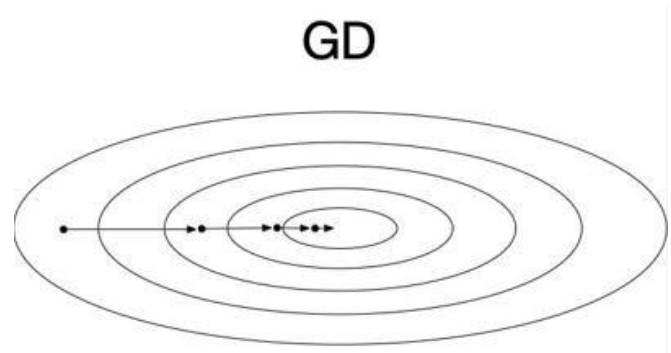
$$\hat{\nabla} J(w) = -e(i)\, x(i)$$

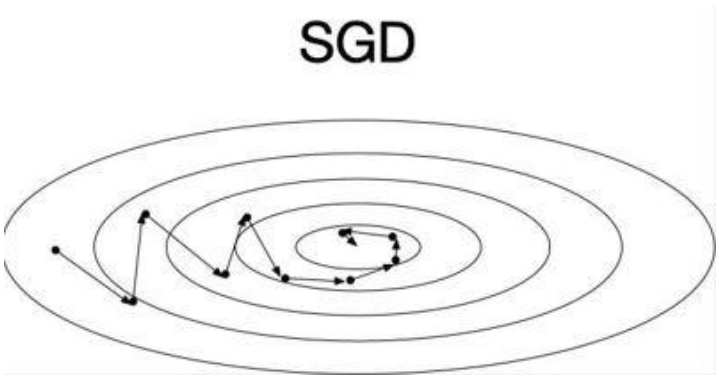
ضرایب به روز رسانی شده بر حسب گرادیان تصادفی لحظه ای به صورت رو به رو محاسبه می شود :

$$w(k+1) = w(k) - \mu\, \hat{\nabla} J(w(k))$$

الگوریتم GD دقت بالا تری دارد اما به دلیل پیمایش تمامی داده ها در هر مرحله سرعت پایین تری دارد و برای پردازش های آفلاین مناسب تر است اما الگوریتم SGD سرعت همگرایی بیشتری دارد و در فیلتر های تطبیقی برای پردازش آنلاین و با سرعت بالا استفاده می شود .

تصویر پایین شهودی خوبی از این دو حالت ارائه می دهد :



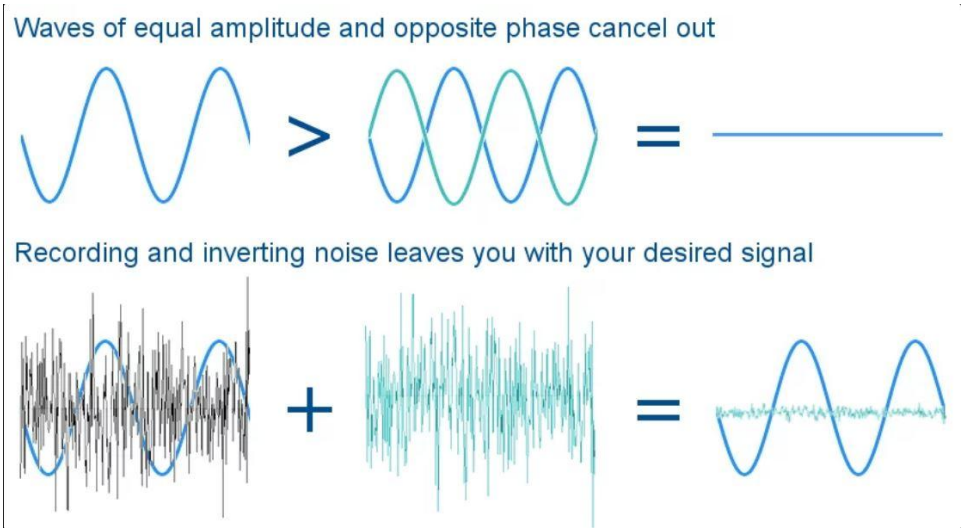


Active noise canceling -

حذف نویز فعال روشی است که با استفاده از تولید سیگنالی با فاز معکوس نسبت به نویز ، نویز مزاحم را حذف می کند .
مراحل انجام این عملیات عبارتند از :
میکروفون سیگنال ترکیبی را ضبط می کند سپس یک فیلتر تطبیقی با الگوریتم LMS نویز را تخمین میزند سپس آنتی نویز که فاز معکوس سیگنال نویز است تولید می شود و آنتی نویز باز طریق اسپیکر پخش می شود و این سیگنال با نویز محیط تداخل مخرب دارد و نویز را کاهش می دهد .

Let $s[k]$ denote the noise signal. Then the active noise cancellation is described as follows:

1. $x[k] = s[k] + n[k]$ (1)
2. $\hat{n}[k] = \sum_{i=0}^{M-1} w_i[k]\, r[k-i]$ (2)
3. $e[k] = x[k] - \hat{n}[k] = s[k] + n[k] - \hat{n}[k]$ (3)
4. $w[k+1] = w[k] + \mu\, e[k]\, r[k]$ (4)
5. $\lim_{k\to\infty} \hat{n}[k] = n[k]$ (5)



4- رمز گشایی فایل صوتی با استفاده از LSB Steganography :

ابتدا به بررسی مفهوم نهان نگاری و یا steganography می پردازیم :

نهان نگاری به معنی مخفی کردن اطلاعات در داده های معمولی به شکلی است که وجود آن اطلاعات پنهان هم مشخص نباشد .

رمزنگاری مفهوم متفاوتی با نهان نگاری دارد . به این معنی که در رمز نگاری اطلاعات می خواهند رمزگذاری شوند ولی در نهان نگاری صرفا با الگوریتم خاصی در دیتا بزرگتری پنهان می شوند .

یکی از الگوریتم برای نهان

های مورد استفاده نگاری LSB می باشد

:

در این روش ابتدا فایل صوتی را به صورت WAV می خوانیم . سپس سمپل های 8 بیتی را از صوت خارج می کنیم . بعد از آن اخرین بیت کم ارزش هر کدام از نمونه ها را کنار هم می گذاریم و دوباره به صورت 8 بیتی به دسیمال تبدیل می کنیم و کد اسکی مربوط را دکود می کنیم :

برای مثال : فرض کنید سمپل های صوتی به صورت زیر از فایل صوتی نمونه برداری شده اند :

Sample 1: 11010010 Sample 2: 01101100 Sample 3: 10111101 Sample 4: 11100001

Sample 5: 01010111 Sample 6: 00111000 Sample 7: 10011010 Sample 8: 01100011

حالا بیت های LSB را کنار هم قرار می دهیم :

LSB bits: 0 0 1 1 1 0 0 1

Binary: 00111001 = Decimal: 57

'9' = (57)ASCII پس پیام مخفی شده کاراکتر 9 بوده است .

از مزایای این روش می توان اشاره به سادگی پیاده سازی و حفظ کیفیت صادا اشاره کرد . به این معنی که تغییرات ایجاد شده قابل شنیدن نیستند . همچنین اگر فایل صوتی حجم بالایی داشته باشد . می توان اطلاعات زیادی را در فایل های بزرگ صوتی پنهان کرد .

از معایب این روش می توان به قابل شناسایی بودن پیام اشاره داشت زیرا رمزگذاری نشده است و همچنین در صورت فشرده سازی و اثر کردن نویز خارجی پیام مخفی از بین می رود .

5- حذف نویز با استفاده از فیلتر ها بر روی صدای ضبط شده

- **Adaptive filter**

ابتدا فایل صدای ضبط شده را در پایتون لود می کنیم و بعد از نرمالیز کرده سیگنال ورودی و تک کاناله کردن آن با استفاده از تابعی مجزا نویز پاور لاین 50 هرتز را به آن اضافه می کنیم : دامنه نرمالیز شده نویز 0.8 فرض شده است .

```
#function to add power line noise
def add_noise_50(original, fs):
    return original + (0.8 *np.sin(100*np.pi*np.arange(len(original))/fs))

signal_with_noise= add_noise_50(original, fs)
```

به صورت ریاضیاتی در اصل خواهیم داشت :

$$x[n] = A \cdot \sin\left(2\pi f_0 \cdot \frac{n}{f_s}\right)$$

where:

- A: noise amplitude (e.g., 0.8)
- $f_0 = 50$ Hz: power line frequency
- f_s: sampling frequency

حال باید با استفاده از فیلتر تطبیقی تعریف شده توسط الگوریتم گرادیان کاهشی و تابع ضرر LMS کد را با تعاریف ریاضیاتی که در بخش های قبل بررسی کردیم تطبیق می دهیم :

```
def adaptive_lms(d,x,mu,M):
```

- $d[n]$: noisy signal
- $x[n]$: reference noise
- μ: learning rate

```
for i in range(M, N):
        x_1 = np.flip(x[i - M:i])
```

$$\mathbf{x}[n] = [x[n], x[n-1], \ldots, x[n-M+1]]^\top$$

```
        y_n = 0.0
        for j in range(M):
            y_n += w[j] * x_1[j]
```

$$y[n] = \sum_{k=0}^{M-1} w_k[n] \cdot x[n-k]$$

```
e[i] = d[i] - y_n
y[i] = e[i]
```

$$e[n] = d[n] - y[n]$$

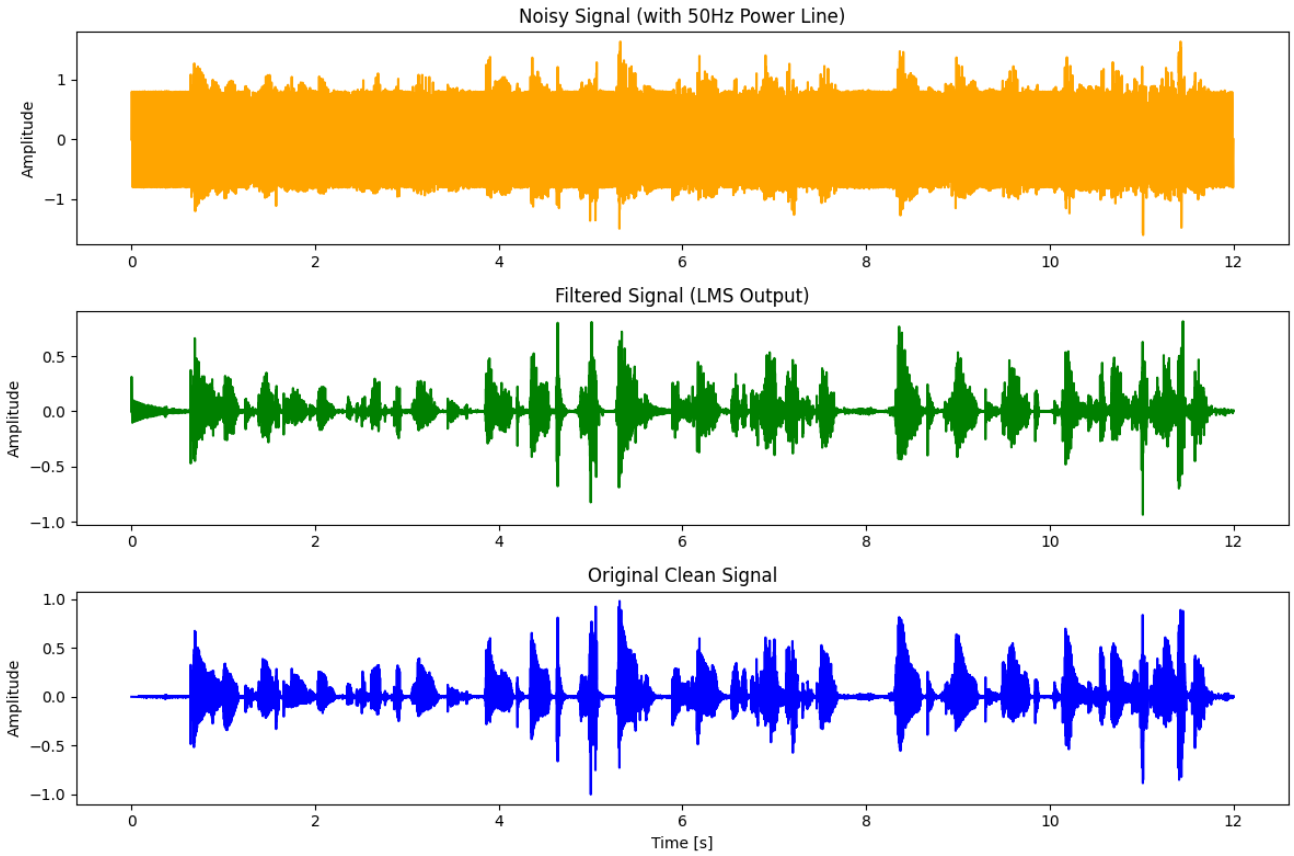
```
for j in range(M):
      w[j] += 2* mu*e[i]*x_1[j]
```

$$w_k[n+1] = w_k[n] + 2\mu \cdot e[n] \cdot x[n-k], \quad \text{for } k = 0, 1, \ldots, M-1$$

سپس یک سیگنال رفرنس نویز 50 هرتز به فیلتر می دهیم و فیلتر را با مقادیر u = 0.001 و m=32 پیاده سازی میکنیم :

```
# making signal refrence noise
t = np.arange(len(signal_with_noise))/fs
ref = np.sin(100*np.pi*t)
# filtering process
output = adaptive_lms(signal_with_noise, ref,0.001,32)
```

خروجی به صورت نمودار به شکل زیر است مشاهده می شود فیلترینگ به نحو خوبی انجام شده است :



تفاوت در چند دهم ثانیه اول به دلیل همگرا شدن فیلتر و فرایند یادگیری فیلتر می باشد .

- فیلتر fir

برای ساخت فیلتر FIR از تابع آماده lfilter موجود در کتابخانه scipy.signal استفاده مینماییم.این تابع در ورودی علاوه بر سیگنال اولیه به ضرایب فیلتر نیاز دارد که هم میتوان به صورت دو آرایه تک بعدی،که بیانگر ضرایب چندجمله ای های صورت و مخرج تابع تبدیل گویای H(z) هستند ورودی داد؛ و گزینه دیگر که اینجا استفاده شده ساخت ضرایب با استفاده از تابع firwin است که به ترتیب درجه فیلتر،باند فرکانسی که باید در فیلتر ناچ حذف شود(یا در حالت کلی این باند باند خاصی است،یا باید حذف شود یا نگه داشته شود و بقیه باند حذف شود و...) و فرکانس نمونه برداری را دریافت میکند و خودش ضرایب لازم برای فیلتر را میسازد.

```
stopband=[48,52]
fir_coeff = firwin(1001, stopband , pass_zero='bandstop',fs=myfs)
signal_filtered_FIR = lfilter(fir_coeff, 1.0, signal_with_noise)
```

-فیلتر IIR:

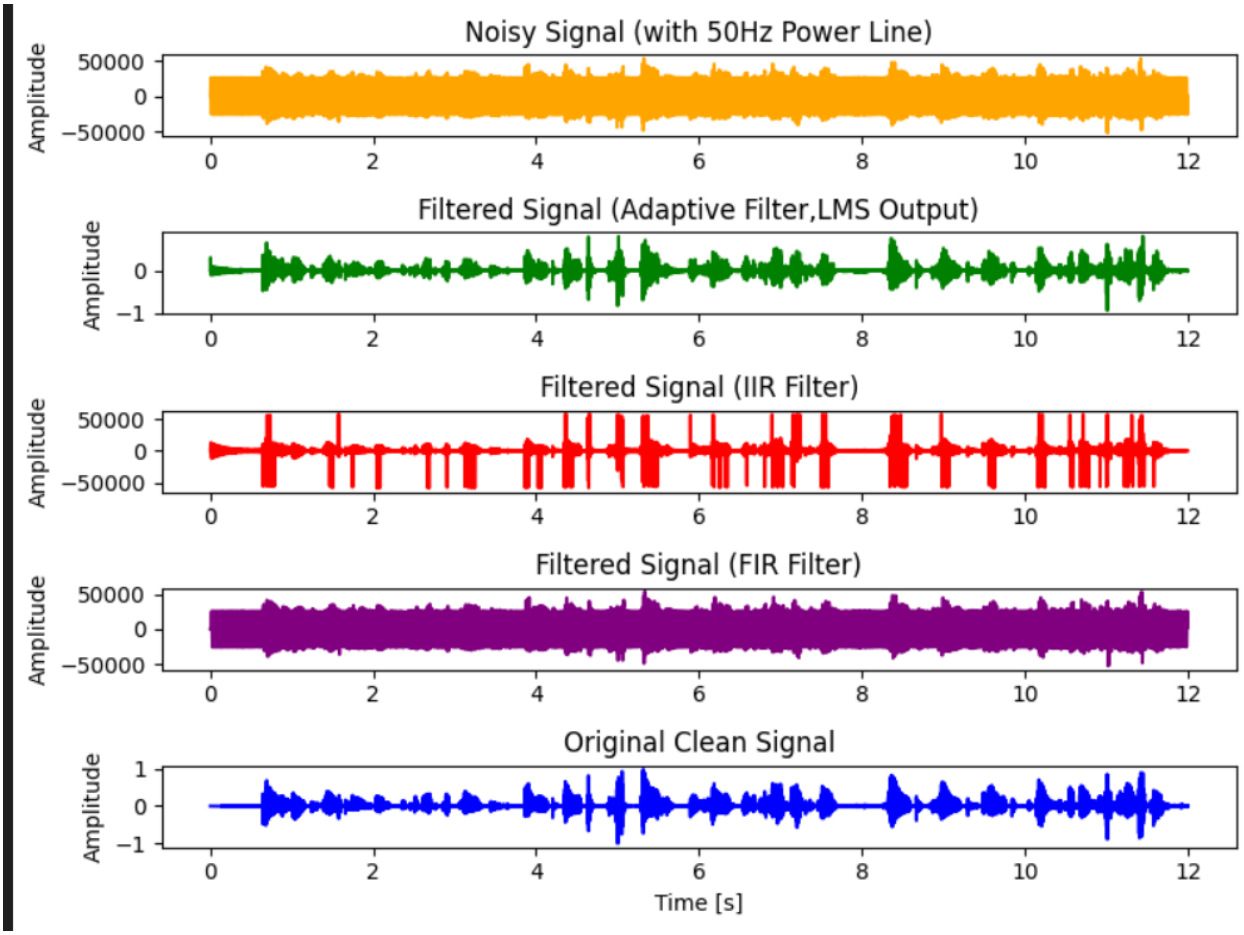
برای ساخت این فیلتر از دستور filtfilt که کاربرد کاملا مشابهی با دستور lfilter دارد استفاده میکنیم،و مجددا به جای اینکه ضرایب فیلتر را دستی تنظیم کنیم از دستور iirNotch استفاده میکنیم که فرکانسی که باید حذف شود و ضریب کیفیت(Quality Factor) را دریافت میکند(ضریب کیفیت هرچه بالاتر باشد فیلتر فرکانس گزین تر و با کیفیت تر،اما پیچیده تر است).

فرکانسی که تابع iirNotch به عنوان پارامتر دریافت میکند فرکانس نرمالیزه شده است.فرکانس $\frac{fs}{2}$ که حداکثر فرکانس موجود است(با فرض برقراری شرط نایکوئیست) به فرکانس 1 مپ میشود،لذا فرکانس 50 هرتز که میخواهیم حذف کنیم معادل فرکانس نرمالیزه $\frac{50}{\frac{fs}{2}}$ است.

```
b, a = iirnotch(w0, Q)

signal_filtered_IIR = filtfilt(b, a, noisySignal)
```

نتیجه گیری:در انتها درمقایسه خروجی ها داریم:



خاتمه:فیلتر FIR با فاصله زیادی ضعیفتر از فیلترهای IIR و Adaptive هستند اما دو فیلتر اخیر هردو عملکرد مناسبی داشته اند؛ با این حال خروجی فیلتر Adaptive به سیگنال اصلی نزدیکتر می باشد.فیلتر FIR کمترین پیچیدگی محاسباتی و بدترین نتیجه را دارد،در حالی که فیلتر adaptive بیشترین پیچیدگی محاسباتی و بهترین نتیجه را دارد.البته پیچیدگی هردو فیلتر IIR,adaptive به ضریب کیفیت بستگی دارد و میتواند متناسب با توان محاسباتی تغییر کند.تابع لازم برای پخش کردن هر سیگنال نیز در انتهای کد موجود است.

2)رمزگشایی:

2.1)رمزگشایی سیگنال بدون نویز:

در این قسمت ابتدا سیگنال را با کتابخانه librosa لود کردیم که از اول سیگنال را به صورت نرمالیزه شده لود میکند،یعنی اندازه هر المان آرایه سیگنال که عددی صحیح قابل نمایش در 16 بیت است(عددی صحیح بین $2^{15}-1$ و -2^{15}) را بر ماکسیمم عدد آرایه تقسیم میکند تا هر آرایه به عدد حقیقی بین 1,1- تبدیل شود.ابتدا این سیگنال را در 100 ضرب میکنیم و به نزدیکترین عدد صحیح رند میکنیم:

```
attenuated_sig=100*signal1 #attenuated signal has float elements between -100,100
for i in range (0,len(attenuated_sig)):
  attenuated_sig[i]=np.round(attenuated_sig[i]) #here we round elements so they be integers
```

حال المان های 2625*n سیگنال را جدا میکنیم،که البته به دلیل اینکه اندیس گذاری پایتون از 0 شروع میشود کد باید المان های 2625*n-1 را جدا کند:

```
mylen=len(attenuated_sig)/2625
mylen=int(mylen)

decoded_signal=[0]*mylen
for i in range (0,len(decoded_signal)):
  decoded_signal[i]=attenuated_sig[i*2625+2624]
```

حال هرالمان انتخاب شده را باید به صورت دودویی دربیاوریم و آخرین رقم(کم اهمیت ترین رقم-LSB) را جدا کنیم.آخرین رقم در هر عدد فرد(چه مثبت چه منفی)برابر 1 و در هر عدد زوج برابر صفر است.این قسمت کد این کار را انجام میدهد:

```
for i in range (0,len(decoded_signal)):
  if(decoded_signal[i]%2==0):
   decoded_signal[i]=0
  else:
   decoded_signal[i]=1 #We only need the last digit of each element,which is 0 if the number is even and 1 if it is odd
```

در انتها باید بیت هارا 8تا 8تا به یکدیگر بچسبانیم تا تشکیل کد اسکی دهند،و هر دسته 8تایی را به نمایس دهدهی تبدیل کنیم که تابع binaryToDecimal به این هدف نوشته شده است و آرایه 8 عضوی ورودی میگیرد و یک عدد خروجی میدهد.اگر آرایه [a7,a6,a5,...a0] داشته باشیم کد اسکی متناظر،برابر است با

$$\sum_{i=0}^{7} 2^i a_i$$

که این تابع حساب میکند.در انتها کافیست برای نمایش پیاک از یک استرینگ تهی شروع کنیم،هر کد اسکی را با دستور chr(x) به کاراکتر متناظر تبدیل کنیم و به استرینگ بیفزاییم.در انتها پیام مخفی شده به صورت زیر است:

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python312/python.exe c:/Users/L
Signals & System
```

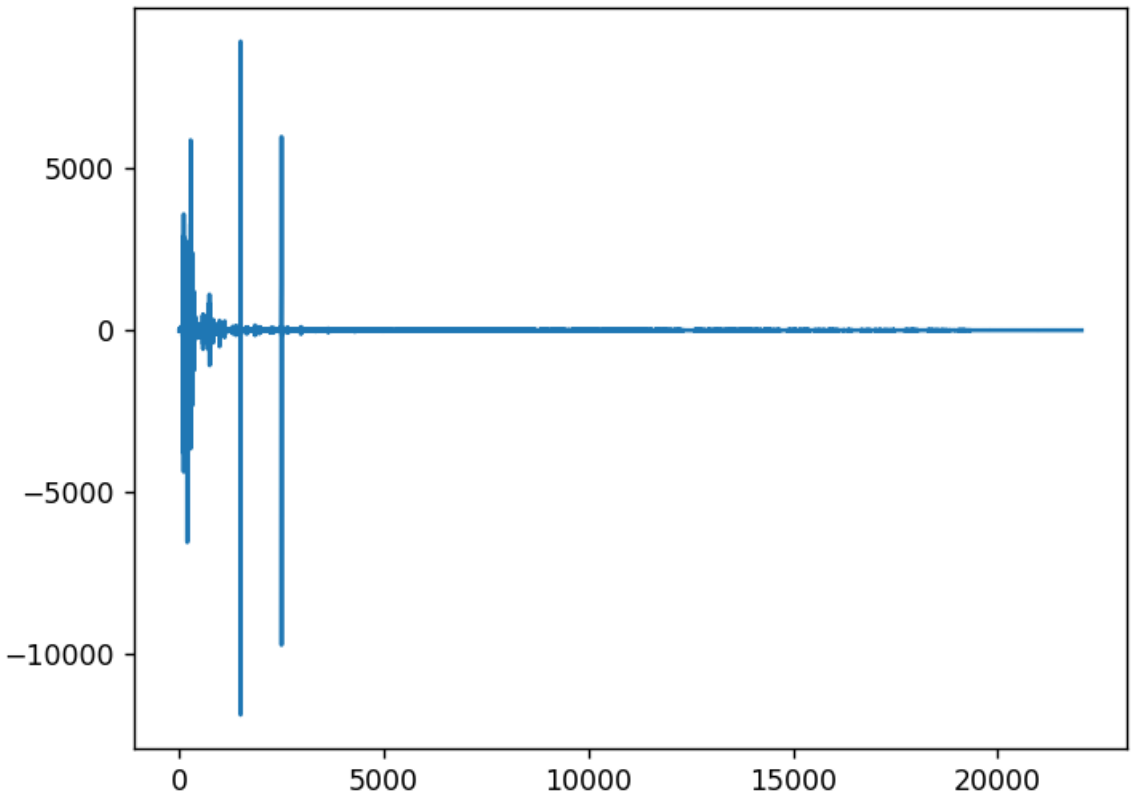
2.2)رمزگشایی سیگنال نویزدار:

الگوریتم های انجام شده روی سیگنال برای استخراج المان های خاص،و سپس بیت آخر این المان ها و چسباندن 8تا 8تا به یکدیگر هیچ تفاوتی با حالت قبل ندارد،تنها تغییر این است که ابتدا باید نویز روی سیگنال را حذف کنیم و سپس روی سیگنال بدون نویز عملیات سابق را انجام دهیم.

ابتدا برای تشخیص نویز یک تبدیل fft از سیگنال میگیریم:

```
fft_data = np.fft.fft(signal_with_noise)
N=len(signal_with_noise)
freqs = np.fft.fftfreq(N, 1/sample_rate1)
for i in range(0,len(freqs)):
   if(abs(fft_data[i])>7000):
       print(freqs[i])
plt.plot(freqs[0:N//2],fft_data[0:N//2])
plt.show()
```

تبدیل fft:



مشاهده میشود 2 فرکانس پرت با دامنه زیاد وجود دارند.برای تشخیص دقیق این فرکانس ها دو خط هایلایت شده نوشته شده اند.خروجی کد:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

2500.05
-2500.05
-1500.05625
-1499.925
```

پس باید با فیلتر Adaptive فرکانس های 2500 و 1500 هرتزی را حذف کنیم.

پس دو بار سیگنال را از فیلتر رد میکنیم و هر بار رفرنس فیلتر اداپتیو را فرکانسی قرار میدهیم که باید حذف شود:

```
# filtering process
output = adaptive_lms(signal_with_noise, ref,0.00005,32) #here we remove 1500Hz
noise
output=adaptive_lms(output,ref2,0.00005,32)#here we remove 2500Hz noise
```

حال سیگنال output همان سیگنال قسکت قبل است و با کار مشابه به خروجی زیر میرسیم:

```
PS C:\Users\LENOVO> & C:/Users/LENOVO
Signals & System
```

پایان