華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Data Filtering and Frequency Analysis with Power Spectrum Density

数字信号处理

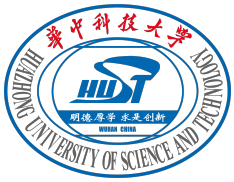
(Digital Signal Processing Projects)

学　号: D201880110

姓　名: 肖春雨

专　业: 无线电物理

院　系: 物理学院



2021-02-22

Abstract

Fourier transform opened a door to analyze signal in frequency domain, from which a series of transforms were born, such as Laplace transform for system design and Z transform for discrete analysis. In order to modify the frequency properties of a given signal, a large class of filters can be employed.

In this paper, power spectrum density is firstly introduced, which extends the Fourier transform and can analyze random signal in frequency domain. An algorithm to estimate logarithmic frequency axis power spectral density is discussed and compared with the periodogram and Welch's method.

Then, filter design with built-in functions in both MATLAB and Python is introduced, including infinite impulse response filters and finite impulse response filters. Some useful functions to convert analog filter to digital filter are mentioned and compared. Zero-phase filtering technology is discussed to compensate the delay caused by normal filtering.

Finally, a simulation was run to validate the performances of the filter to suppress high-frequency disturbances of normal 50 Hz power system. The results confirmed that the filter attenuated disturbances outside the bandwidth about three orders of magnitude.

Contents

1 Introduction

Fourier series (FS) gives a new idea to expand a continuous and periodic signal with a series of trigonometric functions, which glanced at frequency analysis at the first time. Based on the idea of FS, other transformations were proposed to extend the limit on FS: (1) Fourier transform (FT) for continuous but non-periodic signals; (2) discrete time Fourier transform (DTFT) for sampled signals; (3) discrete Fourier transform (DFT) for completely discrete analysis. These transformations make up Fourier transform family under Dirichlet conditions. Besides, Laplace transform extends the frequency into complex domain and Z transform simplifies DTFT practically[1].

Analysis in frequency domain provides more options to process signals or evaluate linear systems. For example, a spectrum given by fast Fourier transform (FFT, the fast algorithm for DFT) shows the frequency distribution of the signal; the location of poles and zeros of a system demonstrate the stability and dynamic properties.

In this paper, attentions are paid to digital signal processing. Power spectrum density is firstly introduced as a tool for frequency analysis and a new algorithm is delivered. Then the design and properties of some common used filters are discussed. Finally, an example is given to show processing procedure and the results.

2 Power Spectrum Density

DFT is used to analysis a certain signal. However, there are many kinds of stochastic signals whose statistical properties are the most concerned. As for a stationary and random signal, power spectrum density (PSD) is taken to estimate its properties in frequency domain[2].

PSD is defined as the Fourier transform of the autocorrelation function of the given signal $x(t)$

$$S_x(\omega) = \mathscr{F}\left(R_{xx}(\tau)\right) = \int_{-\infty}^{+\infty} R_{xx}(\tau)\mathrm{e}^{-\mathrm{j}2\pi\omega\tau}\,\mathrm{d}\tau \tag{1}$$

where the autocorrelation function $R_{xx}(\tau)$ is

$$R_{xx}(\tau) = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)x(t-\tau)\,\mathrm{d}t \tag{2}$$

Thus, the PSD is computed by

$$\begin{aligned} S_x(\omega) &= \int_{-\infty}^{+\infty}\left(\lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)x(t-\tau)\,\mathrm{d}t t\right)\mathrm{e}^{-\mathrm{j}2\pi\omega}\,\mathrm{d}\tau \\ &= \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)\mathrm{e}^{-\mathrm{j}2\pi\omega t}\left(\int_{-\infty}^{+\infty} x(t-\tau)\mathrm{e}^{-\mathrm{j}2\pi\omega(\tau-t)}\right)\mathrm{d}t \end{aligned}$$

$$
\begin{aligned}
&= \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} x(t) \mathrm{e}^{-\mathrm{j}2\pi\omega t} X^*(\mathrm{j}\omega)\, \mathrm{d}t \\
&= \lim_{T\to\infty} \frac{1}{2T} X(\mathrm{j}\omega) X^*(\mathrm{j}\omega) \\
&= \lim_{T\to\infty} \frac{1}{2T} |X(\mathrm{j}\omega)|^2
\end{aligned} \tag{3}
$$

Eq (3) is also known as Wiener–Khinchin theorem, which demonstrate the relationship between PSD and general frequency spectrum.

2.1 Classical Power Spectrum Density Estimation

Analog signals are analyzed based on the sampled data. Two important effects should be considered in sampling: (1) sampling should satisfy Nyquist sampling theorem, where the sampling frequency should be at least twice the maximum frequency of the origin signal. In practice, an analog anti-aliasing filter is usually used before the signal is sampled. (2) Finite sampling is equivalent to applying a window function to the infinite signal, which leads to frequency leakage effect. Because of these two effects, the PSD computed from the sampled data is just the estimation of the PSD of origin analog signal. That's why we usually say 'spectrum estimation' instead of 'spectrum calculation'.

The most direct method to estimate PSD is to discretize Eq (3) using DFT and cancel the limit as below.

$$
S_x(\omega) \approx \frac{1}{T} |X(k)|^2 = \frac{1}{N f_s} |X(k)|^2 \tag{4}
$$

where T is the total time of data, N is the total number of points, f_s is the sampling frequency. $X(k)$ is the DFT of $x(n)$, given by

$$
X(k) = \sum_{n=0}^{N-1} x(n) \mathrm{e}^{-\mathrm{j}2\pi \frac{n}{N} k} \tag{5}
$$

In the case where a window function $w(n)$ is applied to the sampled data, the PSD should be normalized as below. The 2 in the numerator converts the two-sided PSD to one-sided PSD, whose square root is the most used in practice.

$$
S_x(\omega) = \frac{2}{\sum_{n=0}^{N-1} w^2(n) f_s} |X(k)|^2 \tag{6}
$$

PSD can be estimated by combining Eq (5) and Eq (6). This direct method is called 'periodogram'. Both MATLAB and Python provide built-in functions to calculate the periodogram. They are named `periodogram` and `scipy.signal.periodogram`.

An example for PSD estimation is shown in Fig 2. The result of periodogram looks very noisy, especially in high frequency band. To improve the accuracy, Welch proposed

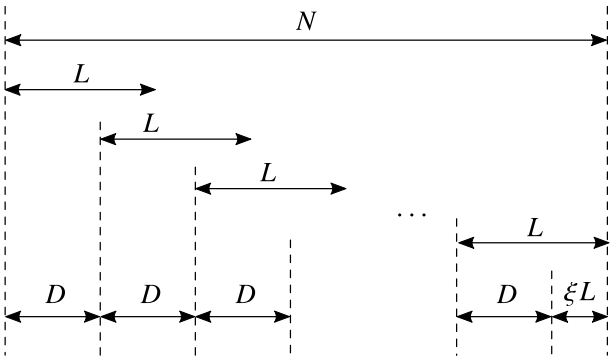


Figure 1. The idea of Welch's method to estimate PSD.

a method by dividing the data into overlapped segments and computing the average of the periodogram of each segment.

The idea of Welch's method is shown in Fig 1 and the relevant functions are `pwelch` in MATLAB and `scipy.signal.welch` in Python.

2.2 Logarithmic Frequency Axis Power Spectral Density

As shown in Fig 2, by averaging PSD using Welch's method, the curve is less noisy. However, because the data is divided into segments, the length decreases, which decreases the frequency resolution. To further improve Welch's method, a new method named Logarithmic frequency axis Power Spectral Density (LPSD) was proposed[3].

Considering that most signals occupy a wideband in frequency domain, the PSD is usually plotted with a logarithmic axis. Thus, the basic idea of the LPSD is to construct logarithmic frequency axis rather than linear axis. As a consequence, the frequency resolution changes with frequency, which allows dividing data into different segments. That is to say, the LPSD algorithm succeeds the idea of Welch's method and divides data according to the frequency point at which the PSD is estimated. The detailed procedure of the algorithm is discussed hereafter.

A logarithmic axis is made up of geometric progression, the length of which is assumed to be J_{des} and can be set manually, and it's not necessary equal to the length of data, N. The common ratio q and ideal frequency resolution can be calculated as below.

$$\lg f_{\max} - \lg f_{\min} = q\,(J_{\text{des}} - 1) \quad \Rightarrow \quad q = \left(\frac{f_{\max}}{f_{\min}}\right)^{\frac{1}{J_{\text{des}}-1}} = \left(\frac{N}{2}\right)^{\frac{1}{J_{\text{des}}-1}} \tag{7}$$

$$r_0(j) = f(j+1) - f(j) = (q-1)\,f(j) \tag{8}$$

Note that the maximum frequency is given by Nyquist frequency $f_{\max} = \frac{f_s}{2}$, and the minimum frequency is limited by the total sampling time $f_{\min} = \frac{1}{T} = \frac{f_s}{N}$. The index is denoted as j, which shows that the resolutions differ between frequencies.

However, the ideal frequency resolution $r_0(j)$ could be less than $f_{\min}$, which is not meaningful. So the resolution should be adjusted. To do so, we assume the desired number of segments to be K_{des}, and the overlap ratio is ξ. So the desired resolution can be computed according to Fig 1.

$$(K_{\text{des}} - 1)(1 - \xi) L_{\text{des}} + L_{\text{des}} = N \tag{9}$$

$$r_{\text{des}} = \frac{f_s}{L_{\text{des}}} = \frac{f_s}{N}((K_{\text{des}} - 1)(1 - \xi) + 1) \tag{10}$$

The ideal frequency resolution should be adjusted as below.

$$r'(j) = \begin{cases} r_0(j) & r_0(j) \geq r_{\text{des}} \\ \sqrt{r_0(j) r_{\text{des}}} & r_0(j) < r_{\text{des}} \text{ and } \sqrt{r_0(j) r_{\text{des}}} > f_{\min} \\ f_{\min} & \text{else} \end{cases} \tag{11}$$

Besides, the frequency resolution should consider an integer length of data and be further modified.

$$L(j) = \left\lfloor \frac{f_s}{r'(j)} \right\rfloor \quad \Rightarrow \quad r(j) = \frac{f_s}{L_(j)} \tag{12}$$

So far, the logarithmic frequencies can be computed by iterations given by Eq (8), Eq (11) and Eq (12).

Similar to the Welch's method, D and L in Fig 1 differ with index j. For j-th frequency point,

$$D(j) = \lfloor (1 - \xi) L(j) \rfloor \tag{13}$$

and the number of segments is computed below.

$$K(j) = \left\lfloor \frac{N - L(j)}{D(j) + 1} \right\rfloor \tag{14}$$

Average is usually deducted in PSD estimation. The average of j-th frequency point and k-th segment is given below.

$$a(j,k) = \frac{1}{L(j)} \sum_{l=0}^{L(j)-1} x(D(j)(k-1) + l) \tag{15}$$

Deducting the average and applying window function $w(j,l)$, the segment to be transformed is gained.

$$G(j,k,l) = \big(x(D(j)(k-1) + l) - a(j,k)\big) w(j,l) \tag{16}$$

The PSD of k-th segment at frequency $f(j)$ is calculated by single point DFT as below.

$$A(j,k) = \sum_{l=0}^{L(j)-1} G(j,k,l) \mathrm{e}^{-\mathrm{j}2\pi \frac{m(j)}{L(j)} l} \tag{17}$$

Note that the frequency index $m(j) = \frac{f(j)}{r(j)}$ is not necessary to be an integer.

Finally, the PSD at j-th frequency point is given by the average of $K(j)$ segments according to Wiener–Khinchin theorem.

$$P(j) = \frac{C_{\mathrm{PSD}}(j)}{K(j)} \sum_{k=1}^{K(j)} |A(j,k)|^2 \tag{18}$$

The coefficient $C_{\mathrm{PSD}}(j)$ is used to normalize the PSD, which is given below.

$$C_{\mathrm{PSD}}(j) = \frac{2}{f_s \sum_{l=0}^{L(j)-1} w^2(j,l)} \tag{19}$$

The complete procedure of the LPSD algorithm has been discussed. The function is realized in both Python and MATLAB code, which can be found in the appendix. A comparison is given in Fig 2. The signal is obtained by filtering a white noise with elliptic filter so that the PSD should be equal to the gain of the transfer function theoretically.

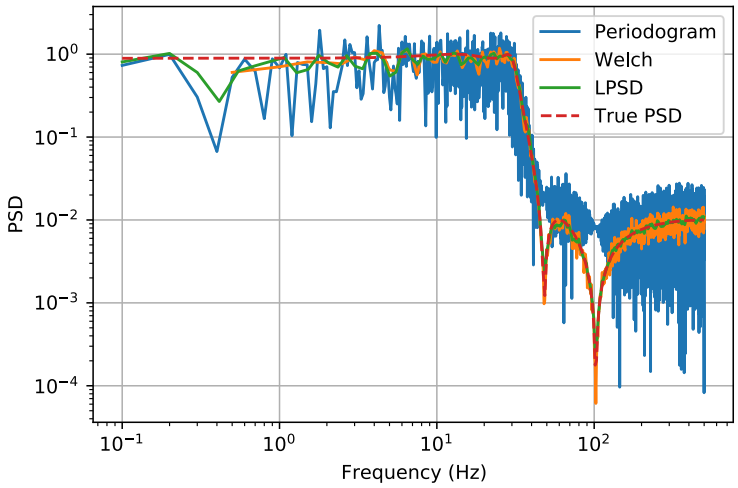


Figure 2. Comparison of three methods to estimate PSD.

It's obvious that both the periodogram and the LPSD algorithms can calculate PSD at minimum frequency limited by the sampling time. But the result of periodogram is the most noisy. Welch's method improves the accuracy but sacrifices the minimum frequency. The LPSD provides the best estimated PSD, the result of which is very close to the theoretical value. However, the cost of the LPSD is to spend more time for calculation because it uses single point DFT and cannot benefit from FFT.

3 Filter Design

Properties of a given signal in frequency domain can be obtained from its PSD. To change these properties, a filter with certain frequency response should be applied to the signal.

3.1 Common Used Filters

Since filter design has been a very proven technique, the theoretical analysis won't go far but some useful functions in MATLAB and Python will be introduced. Dynamic of an analog filter can be described in a differential function or a transfer function with Laplace transform as below.

$$y^{(n)} + a_1 y^{(n-1)} + \cdots + a_n y = b_0 u^{(m)} + b_1 u^{(m-1)} + \cdots + b_m u \quad (m < n) \tag{20}$$

$$H(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \cdots + b_n}{s^n + a_1 s^{n-1} + \cdots + a_n} \quad (m < n) \tag{21}$$

If there are poles in the transfer function, the filter would have infinite impulse response in time domain, for which we call it IIR filter for short.

To design a filter with built-in functions, properties of the filter shown in Fig 3 should be specified. Taking low-pass filters as example, the frequency of passband is denoted by f_{pass} and R_{pass} is the allowed ripple. Stopband frequency is f_{stop} and the attenuation is R_{stop}.

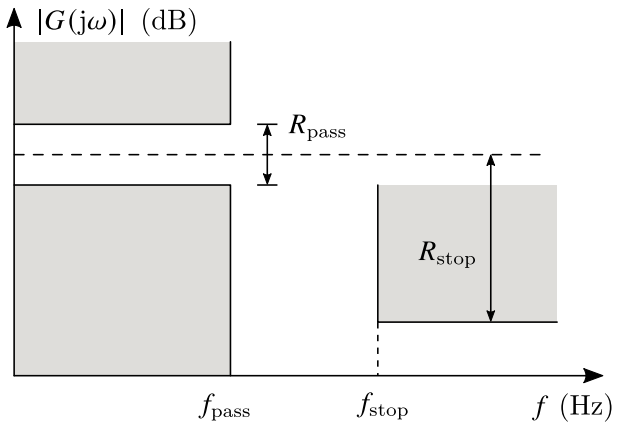


Figure 3. Filter properties specification.

Some common used filters are listed in Tab 1 with MATLAB built-in functions for filter design[4]. In these functions, `Wp` and `Ws` are respectively the angular frequency of passband and stopband ($\omega_p = 2\pi f_{\text{pass}}$ for example). Passband ripple `Rp` and stopband attenuation `Rs` are described in decibels. With the option `'s'`, the function designs an analog filter. If a digital filter is desired, functions should be called without the option `'s'` and `Wp` and `Ws` should be the digital frequency which is normalized by the sampling frequency.

For Python users, there are functions with the same names in `scipy.signal` package[5]. If the package is imported by `import scipy.signal as signal`, a prefix `signal.` should be attached to the functions to be called. Besides, the programming rules also

has to be noticed. For example, the command `[b,a]=butter(n,Wn,'s')` in MATLAB is equivalent to the command `b,a=signal.butter(n,Wn,analog=True)` in Python.

Table 1. MATLAB built-in functions for IIR filter design

Name	Built-in Functions
Butterworth	`[n,Wn] = buttord(Wp,Ws,Rp,Rs,'s');`
	`[b,a] = butter(n,Wn,'s');`
Chebyshev Type I	`[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs,'s');`
	`[b,a] = cheby1(n,Rp,Wn,'s');`
Chebyshev Type II	`[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs,'s');`
	`[b,a] = cheby2(n,Rs,Wn,'s');`
Elliptic Filter	`[n,Wn] = ellipord(Wp,Ws,Rp,Rs,'s');`
	`[b,a] = ellip(n,Rp,Rs,Wn,'s');`

Comparison of four common used filters are shown in Fig 4 with options `Wp=1`,`Ws=3`, `Rp=1`, `Rp=40`. For the same target, elliptic filter has the least order but there are ripples in both passband and stopband. Butterworth filter has flat frequency response with passband, but the order is the largest, which means it takes more resources for calculation. Chebyshev filters have a compromise with order and ripple, where type I has ripples in passband and is flat in stopband, while type II is on the contrary.

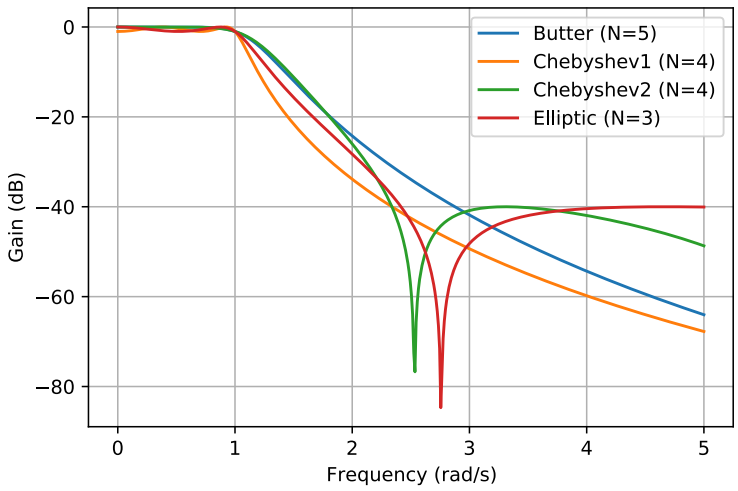


Figure 4. Comparison of four common used IIR filters.

As has been mentioned, functions mentioned above can also be used to design digital filters. Besides, some methods such as 'impulse invariance' and 'bilinear' can be used to discretize an analog filter. Take an analog Butterworth low-pass filter for example, the

discretized filter response is shown in Fig 5. Note that the discretization could lead to aliasing effects, which may change the behaviors of the filter.

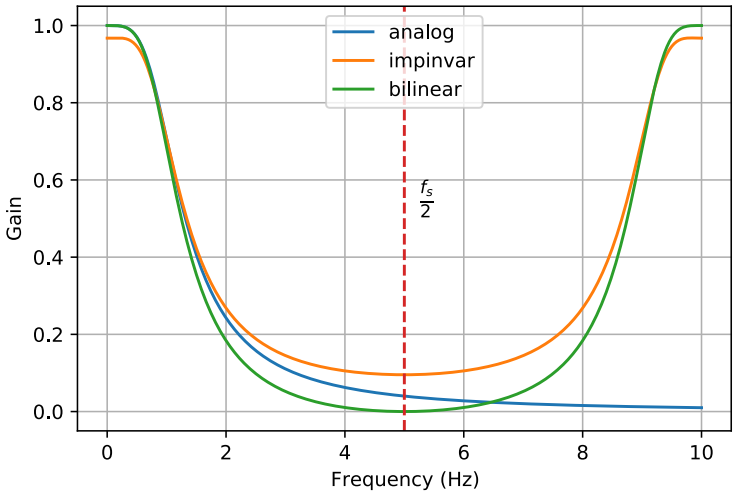


Figure 5. Digital filter design by discretization.

Digital systems have a unique property which allows all-zero system to exist. These systems are described in difference equation or transfer function with Z transform as below.

$$y(n) = b_0u(n) + b_1u(n-1) + \cdots + b_mu(n-m) \sum_{k=0}^{m} b_ku(n-k) \tag{22}$$

$$H(z) = \frac{Y(z)}{U(z)} = b_0 + b_1z^{-1} + \cdots + b_mz^{-m} = \sum_{k=0}^{m} b_kz^{-m} \tag{23}$$

The impulse response in time domain of this all-zero system is finite, thus this kind of filter is call FIR filter for short. Without poles, FIR filter is always stable.

Two methods are generally used to design a FIR filter. The first one is base on window function, whose idea is to apply a certain window function to the response of an IIR filter. MATLAB provides `fir1` function to realize this design. For Python, the function is `scipy.signal.firwin`.

Another method for FIR filter design is based on the frequency sample, where only filter order, frequencies and target responses are necessary. The function is named `fir2` in MATLAB and `scipy.signal.firwin2` in Python.

Applications with graphical user interface (GUI) are also available for filter design. They are `filterBuilder` or `filterDesigner` (or `fdatool` in old version) in MATLAB and `pyfdax` in Python[6].

FIR filter is easy to realize in digital device while IIR filter has less order. Filter type should be selected according to the requirements and equipment constraints before

it is designed. Design results should be checked, including stability, frequency response. Besides, for real-time applications, the filter must be causal.

3.2 Zero-Phase Filtering

The dynamic of the filter always leads to delay. To decrease the delay, an all-pass filter can be employed to adjust the phase response of the filter. As for sampled data, a technology name zero-phase filtering can be used. For digital filter with transfer function $H(z)$, there are four steps to realize zero-phase filtering[7].

Step (1) Filter: $X(z) \rightarrow H(z)X(z)$;

Step (2) Reverse: $H(z)X(z) \rightarrow H(z^{-1})X(z^{-1})$;

Step (3) Filter: $H(z^{-1})X(z^{-1}) \rightarrow H(z)H(z^{-1})X(z^{-1})$;

Step (4) Reverse: $H(z)H(z^{-1})X(z^{-1}) \rightarrow H(z^{-1})H(z)X(z)$.

The transfer function of the complete process $G(z)$ can be calculated as below.

$$G(z) = H(z^{-1})H(z) = |H(\omega)|\, e^{j\varphi(\omega)}\, |H(\omega)|\, e^{-j\varphi(\omega)} = |H(\omega)|^2 \tag{24}$$

There is no imaginary part in the transfer function so that no delay is introduced. From a practical point of view, when the data is reversed and filtered again, the same delay is introduced but in an opposite direction. Thus, the delay from the first filtering is compensated by the second filtering.

The function `filter` in MATLAB or `scipy.signal.lfilter` in Python can be used to filter data with specified filter. To apply zero-phase filtering, the function to be called is `filtfilit` in MATLAB or `scipy.signal.filtfilt` in Python.

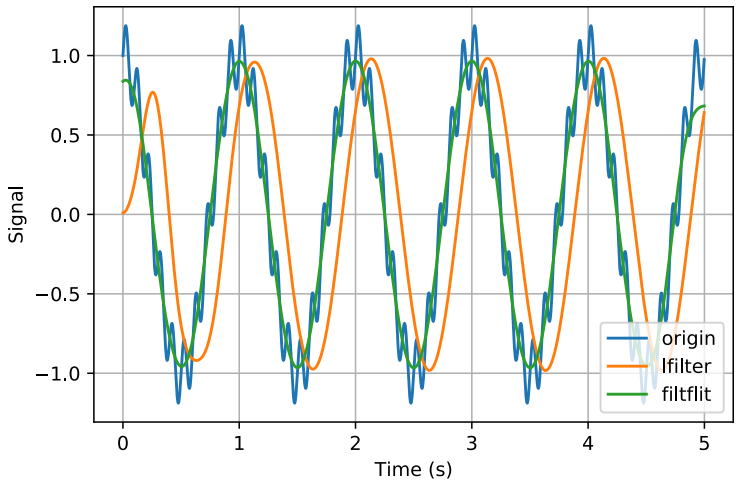


Figure 6. Comparison of normal filtering and zero-phase filtering.

A comparison of normal filtering and zero-phase filtering is shown in Fig 6. The delay of normal filtering is obvious. Attentions should also be paid to the transient performances. There is transient dynamic in the beginning due to the error of initial states. However, there is also transient dynamic in the end with zero-phase filtering, which is caused by the filtering of the reversed data at the third step. Unfortunately, zero-phase filtering is not causal, which means it cannot be used in real-time applications.

4 Results

Due to nonlinear properties, the AC current or voltage from the 50 Hz power supply is polluted by some harmonic frequencies. To ensure the filter suppress the noise adequately, a simulation has been done whose input signal is listed in Tab 2.

Table 2. Input signal for filter validation

Frequency (Hz)	50	100	150	250	White Noise
Amplitude (A)	1.0	0.3	0.2	0.1	0.05 (RMS)
Phase (deg)	0	60	−60	−90	—

The length of data is $N = 1000$ with sampling frequency $f_s = 1\,\text{kHz}$. The target is to filter out components with frequency above 150 Hz. A Chebyshev Type II digital filter is chosen because IIR filter has less order and Chebyshev Type II provides flat frequency within passband. By Specifying $f_{\text{pass}} = 110\,\text{Hz}$, $f_{\text{stop}} = 130\,\text{Hz}$, $R_{\text{pass}} = 1\,\text{dB}$ and $R_{\text{stop}} = 60\,\text{dB}$, the designed filter is shown in Fig 7.

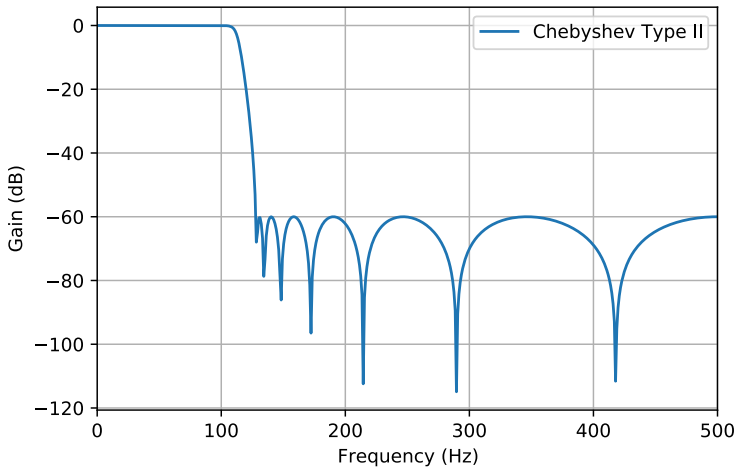
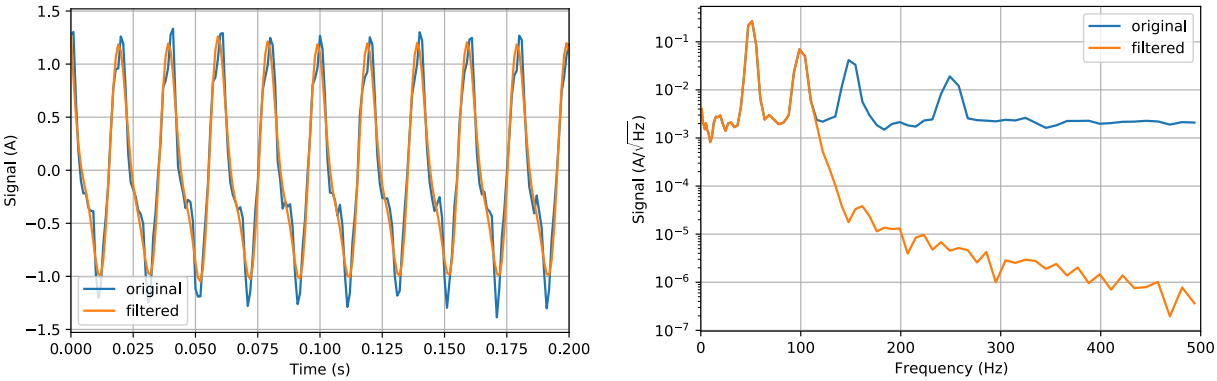


Figure 7. Filter frequency response.



(a) Signals in time domain.

(b) Signals in frequency domain.

Figure 8. Comparison of original and filtered signal.

Original signal is compared with filtered signal in Fig 8 in both time and frequency domain. The zero-phase filtering technology was employed so that no delay is introduced. Because the high-frequency disturbances are removed, the filtered curve seems more smooth. For the PSD, it's obvious that the filter suppress high-frequency noise by about three orders of magnitude as stopband attenuation is set to 60 dB.

5 Conclusions

In this paper, the LPSD algorithm is employed to estimate PSD, which takes advantage of minimum frequency range from the periodogram and succeeds the idea of averaging from Welch's method. Unable to embed FFT, the LPSD takes more time to calculate but provides more precise estimation with logarithmic frequency axis. Filters are designed with built-in functions and zero-phase filtering is used to filter data, which can compensate the delay caused by normal filtering. Zero-phase filtering is not causal, thus it cannot be used in real-time applications. A simulation was designed to validate the filter to suppress high-frequency disturbances from nonlinear effects of the 50 Hz power supply. The results confirmed the properties of the filter and the PSD showed the disturbances in high-frequency band was suppressed by three orders of magnitude.

References

[1] 刘树棠. 信号与系统. 西安交通大学出版社, 2010.

[2] 高晋占. 微弱信号检测. 清华大学出版社, 2004.

[3] Tröbs M, Heinzel G. Improved spectrum estimation from digitized time series on a

logarithmic frequency axis. Measurement, 2006, 39(2): 120-129. DOI: 10.1016/j.measurement.2005.10.010.

[4] Matlab documentation. https://ww2.mathworks.cn/help/.

[5] Scipy documentation. https://docs.scipy.org/doc/scipy/reference/tutorial/signal.html.

[6] Python filter design analysis tool. https://github.com/chipmuenk/pyFDA.

[7] 宋知用. MATLAB 数字信号处理 85 个实用案例精讲: 入门到进阶. 北京航空航天大学出版社, 2016.

(All files will be uploaded to this repository.)

A Source Code

To save pages, only codes that are relevant to the text are given below. I will upload all the codes to my GitHub repository after I get score of *Digital Signal Processing* to avoid being judged cheating.

A.1 LPSD Algorithm

Python Version

```
'''
Personal fuction library in Python
Contents
    (1) pxx,f = lpsd(data,fs,Jdes=1000)

Last Update: 2021-02-10
'''

# %%
import numpy as np
import matplotlib.pyplot as plt

# %% function01: LPSD
'''
LPSD: Logarithmic frequency axis Power Spectral Density
pxx,f = lpsd(data,fs,Jdes=1000)
   data --- Input data series, one dimension vector
   fs   --- Sample frequency, unit: Hz
   Jdes --- Desired frequency points, default: 1000
   pxx  --- One-sided PSD, unit: *^2/Hz
   f    --- Frequency points related to PSD points, unit: Hz
   Default window function is hanning window.

Ref: Improved spectrum estimation from digitized timeseries on a
   logarithmic frequency axis
     Article DOI: 10.1016/j.measurement.2005.10.010

Remark: Unfamiliar with Python by now, the functiondoesn't support
   matrix data by now.
    [MATLAB version supports matrix data.]

XiaoCY 2021-02-10
```

```
'''

# subfuction
# calculate frequency points
def getFreqs(N,fs,Jdes,Kdes,ksai):
    fmin = fs/N
    fmax = fs/2
    r_avg = fs/N*(1+(1-ksai)*(Kdes-1))

    g = (N/2)**(1/(Jdes-1))-1

    f = np.zeros(Jdes)-1
    L = np.copy(f)
    m = np.copy(f)
    j = 0
    fj = fmin
    while fj < fmax:
        rj = fj*g
        if rj < r_avg:
            rj = np.sqrt(rj*r_avg)
        if rj < fmin:
            rj = fmin

        Lj = np.floor(fs/rj)
        rj = fs/Lj
        mj = fj/rj

        f[j] = fj
        L[j] = Lj
        m[j] = mj

        fj = fj+rj
        j += 1

    idx = f<0
    f = np.delete(f,idx)
    L = np.delete(L,idx)
    m = np.delete(m,idx)
    return f,L,m

# main
```

```python
def lpsd(data,fs,Jdes=1000):
    Kdes = 100
    ksai = 0.5
    N = len(data)

    f,L,m = getFreqs(N,fs,Jdes,Kdes,ksai)

    J = len(f)
    pxx = np.zeros(J)
    for j in range(J):
        Dj = np.floor((1-ksai)*L[j])
        Kj = np.floor((N-L[j])/Dj+1)
        w = np.hanning(L[j])
        C_PSD = 2/fs/np.dot(w,w)
        l = np.arange(0,L[j])
        W1 = np.cos(-2*np.pi*m[j]/L[j]*l)
        W2 = np.sin(-2*np.pi*m[j]/L[j]*l)
        A = 0.0
        for k in range(int(Kj)):
            G = data[int(k*Dj):int(k*Dj+L[j])].copy()
            G = G-np.mean(G)
            G = G*w
            A += np.dot(G,W1)**2 + np.dot(G,W2)**2
        pxx[j] = A/Kj*C_PSD

    return pxx,f

if __name__ =='__main__':
    fs = 10
    x = np.random.randn(1000)*np.sqrt(fs/2)
    pxx,f = lpsd(x,fs)

    plt.figure
    plt.loglog(f,np.sqrt(pxx))
    plt.grid()
    plt.xlabel('Frequency (Hz)')
    plt.ylabel(r'PSD ($\rm*/\sqrt{Hz}$)')
```

MATLAB Version

```matlab
% Use LPSD mothod to plot power spectral density
% [Pxx,f] = iLPSD(Data,fs,Jdes)
%    Data --- Input data, processed by column
```

```
4  %    fs   --- Sample frequency, unit: Hz
5  %    Jdes --- Desired frequency points, default: 1000
6  %    Pxx  --- One-sided PSD, unit: *^2/Hz
7  %    f    --- Frequency points related to PSD points,unit: Hz
8  %    Default window function is hanning window.
9  % Demo:
10 %    iLPSD(data,fs)
11 %       Plot PSD using default settings.
12 %    h = iLPSD(data,fs,Jdes)
13 %       Plot PSD with desired points of 1000
14 %    [Pxx,f] = iLPSD(data,fs)
15 %       Return PSD points, not plot any figure
16
17 % Ref: Improved spectrum estimation from digitizedtime series on a
       logarithmic frequency axis
18 %      Article DOI: 10.1016/j.measurement.2005.10.010
19
20 % XiaoCY 2020-04-21
21
22 %% Main
23 function varargout = iLPSD(varargin)
24
25     nargoutchk(0,2);
26     narginchk(2,3);
27
28     data = varargin{1};
29     fs = varargin{2};
30     if nargin == 3
31         Jdes = varargin{3};
32     else
33         Jdes = 1000;
34     end
35
36     Kdes = 100;
37     ksai = 0.5;
38
39     [N,nCol] = size(data);
40     if N==1 && nCol~=1
41        data = data';
42        N = nCol;
43        nCol = 1;
```

```
44      end
45
46      [f,L,m] = getFreqs(N,fs,Jdes,Kdes,ksai);
47
48      J = length(f);
49      P = zeros(J,nCol);
50      for j = 1:J
51          Dj = floor((1-ksai)*L(j));
52          Kj = floor((N-L(j))/Dj+1);
53          w = hann(L(j));
54          C_PSD = 2/fs/sum(w.^2);
55          l = (0:L(j)-1)';
56          W1 = cos(-2*pi*m(j)/L(j).*l);
57          W2 = sin(-2*pi*m(j)/L(j).*l);
58          A = zeros(1,nCol);
59          for k = 0:Kj-1
60              G = data(k*Dj+1:k*Dj+L(j),:);
61              G = G-mean(G);          % G = detrend(G);
62              G = G.*w;
63              A = A + sum(G.*W1).^2+sum(G.*W2).^2;
64          end
65          P(j,:) = A/Kj*C_PSD;
66      end
67
68      switch nargout
69          case 0
70              PlotPSD(P,f)
71          case 1
72              varargout{1} = PlotPSD(P,f);
73          case 2
74              varargout{1} = P;
75              varargout{2} = f;
76          otherwise
77              % Do Nothing
78      end
79  end
80
81  %% Subfunctions
82  % get logarithmic frequency points
83  function [f,L,m] = getFreqs(N,fs,Jdes,Kdes,ksai)
84      fmin = fs/N;
```

```
85      fmax = fs/2;
86      r_avg = fs/N*(1+(1-ksai)*(Kdes-1));
87
88      g = (N/2)^(1/(Jdes-1))-1;
89
90      f = zeros(Jdes,1)-1;
91      L = f;
92      m = f;
93      j = 1;
94      fj = fmin;
95      while fj < fmax
96          rj = fj*g;
97          if rj < r_avg
98              rj = sqrt(rj*r_avg);
99          end
100         if rj < fmin
101             rj = fmin;
102         end
103
104         Lj = floor(fs/rj);
105         rj = fs/Lj;
106         mj = fj/rj;
107
108         f(j) = fj;
109         L(j) = Lj;
110         m(j) = mj;
111
112         fj = fj+rj;
113         j = j+1;
114     end
115     f(f<0) = [];
116     L(L<0) = [];
117     m(m<0) = [];
118 end
119
120 % plot PSD
121 function varargout = PlotPSD(P,f)
122     hLine = loglog(f,sqrt(P));
123     grid on
124     xlabel('Frequency (Hz)')
125     ylabel('PSD ([Unit]/Hz^{1/2})')
```

```
126
127     if nargout == 1
128         varargout{1} = hLine;
129     end
130 end
```

A.2 Main Code

Python Version

```
1  '''
2  Main code for DSP project
3  (See ../DSP_Project_Reqirement&Guidance(2020 Fall).pdfProject-1 for
       detail.)
4
5  XiaoCY 2021-02-18
6  '''
7
8  #%%
9  import numpy as np
10 import matplotlib.pyplot as plt
11 import scipy.signal as signal
12 import springlib as sp
13
14 savepdf = False
15 N = 1000
16 fs = 1000.0
17 pi = np.pi
18
19 t = np.arange(N)/fs
20 x1 = np.cos(2*pi*50*t)
21 x2 = 0.3*np.cos(2*pi*100*t + pi/3)
22 x3 = 0.2*np.cos(2*pi*150*t - pi/3)
23 x4 = 0.1*np.sin(2*pi*250*t)
24 xn = 0.05*np.random.randn(N)
25 x = x1+x2+x3+x4+xn
26
27 #%% design IIR filter
28 Wp = 110./(fs/2)              # passband cornerfrequency (normalized)
29 Ws = 130./(fs/2)              # stopband cornerfrequency (normalized)
30 Rp = 1.                       # passband ripple (dB)
31 Rs = 60.                      # stopband attenuation(dB)
```

```
32
33 Nz,Wn = signal.cheb2ord(Wp,Ws,Rp,Rs)
34 b,a = signal.cheby2(Nz,Rs,Wn)
35
36 f = np.linspace(0,fs/2,500)
37 _,g = signal.freqz(b,a,f,fs=fs)
38 gdb = 20*np.log10(np.abs(g))
39
40 plt.figure()
41 plt.plot(f,gdb,label='Chebyshev Type II')
42 plt.grid()
43 plt.legend()
44 plt.xlabel('Frequency (Hz)')
45 plt.ylabel('Gain (dB)')
46 plt.xlim(0,fs/2)
47 if savepdf:
48     plt.savefig('prjFilter.pdf')
49 plt.show()
50
51 #%% filter data
52 y = signal.filtfilt(b,a,x)
53
54 plt.figure()
55 plt.plot(t,x,label='original')
56 plt.plot(t,y,label='filtered')
57 plt.grid()
58 plt.legend()
59 plt.xlabel('Time (s)')
60 plt.ylabel('Signal (A)')
61 plt.xlim(0,0.2)
62 if savepdf:
63     plt.savefig('prjSignal.pdf')
64 plt.show()
65
66
67 pxx,fx = sp.lpsd(x,fs)
68 pyy,fy = sp.lpsd(y,fs)
69
70 plt.figure()
71 plt.semilogy(fx,np.sqrt(pxx),label='original')
72 plt.semilogy(fy,np.sqrt(pyy),label='filtered')
```

```
plt.grid()
plt.legend()
plt.xlabel('Frequency (Hz)')
plt.ylabel(r'Signal ($\rm A/\sqrt{Hz}$)')
plt.xlim(0,fs/2)
if savepdf:
    plt.savefig('prjPSD.pdf')
plt.show()
```

MATLAB Version

```
% Main code for DSP project
% (See ../DSP_Project_Reqirement&Guidance(2020 Fall).pdf Project-1 for detail.)

% XiaoCY 2021-02-18

%%
clear;clc

N = 1e3;
fs = 1e3;

t = (0:N-1)'/fs;
x1 = cos(2*pi*50*t);
x2 = 0.3*cos(2*pi*100*t + pi/3);
x3 = 0.2*cos(2*pi*150*t - pi/3);
x4 = 0.1*sin(2*pi*250*t);
xn = 0.05*randn(N,1);
x = x1+x2+x3+x4+xn;

%%
Wp = 110/(fs/2);                % passband corner frequency (normalized)
Ws = 130/(fs/2);                % stopband corner frequency (normalized)
Rp = 1;                         % passband ripple (dB)
Rs = 60;                        % stopband attenuation (dB)

[Nz,Wn] = cheb2ord(Wp,Ws,Rp,Rs);
[b,a] = cheby2(Nz,Rs,Wn);

f = linspace(0,fs/2,500);
```

```
g = freqz(b,a,f,fs);
gdb = 20*log10(abs(g));

figure('Name','filter')
plot(f,gdb,'DisplayName','Chebyshev Type II')
grid on
legend
xlabel('Frequency (Hz)')
ylabel('Gain (dB)')

%%
y = filtfilt(b,a,x);

figure('Name','time')
plot(t,x,'DisplayName','original')
hold on
grid on
plot(t,y,'DisplayName','filtered')
legend
xlabel('Time (s)')
ylabel('Signal (A)')
xlim([0,0.1])

figure('Name','PSD')
[pxx,f] = iLPSD([x,y],fs);
semilogy(f,sqrt(pxx))
grid on
legend('original','filtered')
xlabel('Frequency (Hz)')
ylabel('PSD (A/Hz^{1/2})')
```