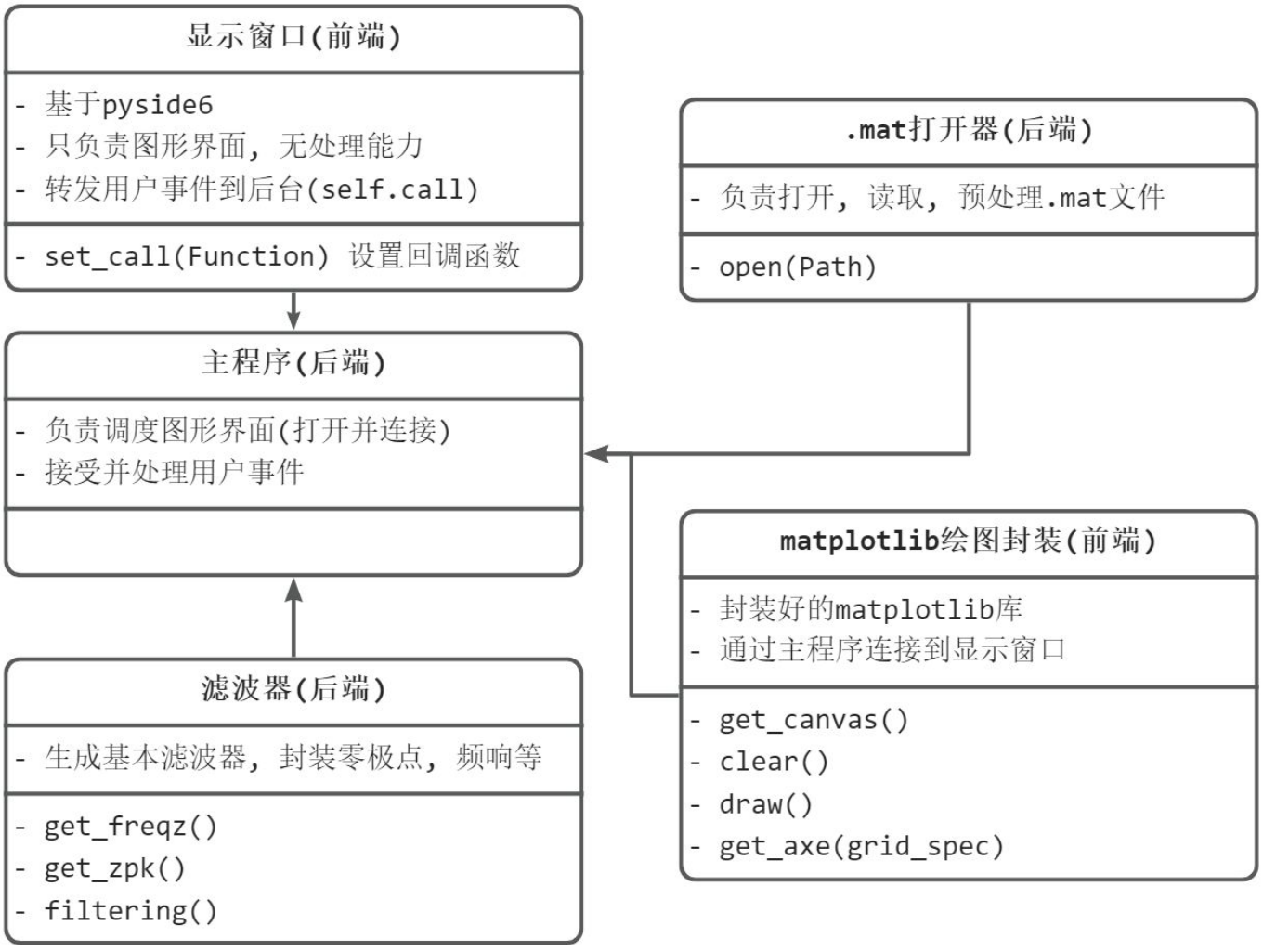
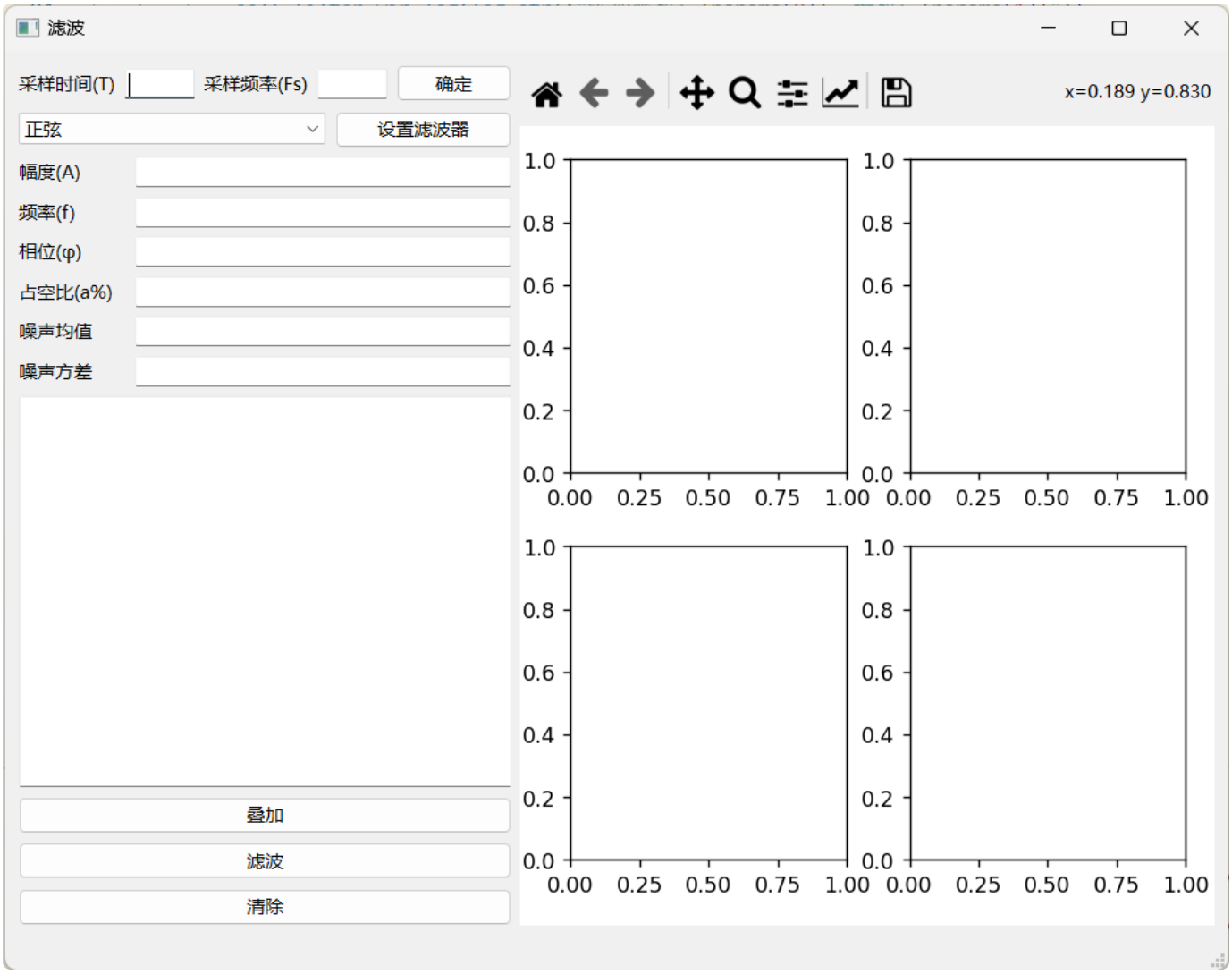
信号与信息处理课设(python实现)

> 使用环境: python 3.11.7
>
> matplotlib==3.8.2
>
> numpy==1.26.3
>
> scipy==1.11.4
>
> pyside6==6.6.1

实验一 GUI界面设计

实验输出:





实验二 典型信号的建模与产生

实验要求:

1. 正弦波信号

   生成指定频率f,幅度A,相位Φ的正弦波，时长为L,采样频率s为1000Hz,并在GUI窗口中显示。

2. 方波信号

   生成指定幅度A、占空比a的方波信号，时长为L,采样频率fs为1000Hz,并在GUI窗口中显示。

3. 混合正弦信号

   生成一频率为f1=30Hz,另一频率为f2=300Hz,幅度均为A=1,时长为L=1s,相位为0的混合正弦波信号，采样频率f$为1000Hz,并在GUI窗口中显示。

4. 生成服从高斯分布的白噪声序列，时长为L,均值为山，方差为ō，并在GUI窗口中显示。

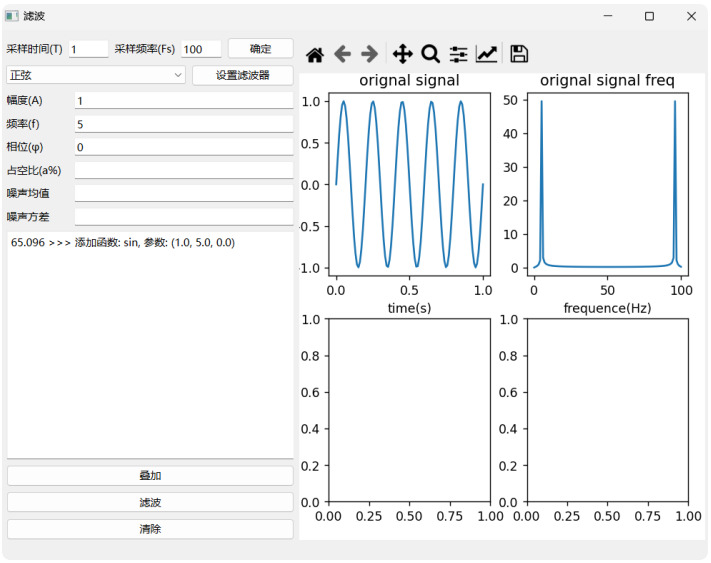
5. 生成服从均匀分布的白噪声序列，时长为工，均值为μ，方差为σ，并在GUI窗口中显示。
6. 加载己有的ssvep.mat文件信号，并在GUI窗口中显示。

实验过程:

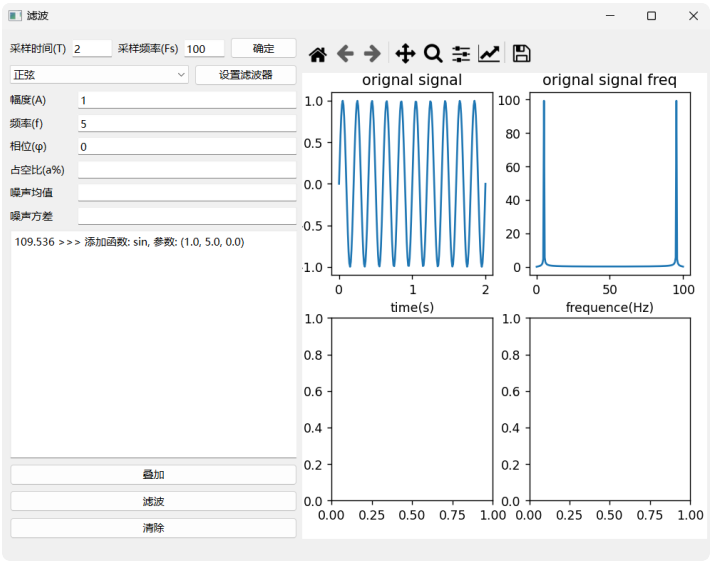
设计了MySignal类, 以表达式形式记录信号, 可动态设置采样频率与采样时间

MySignal Python

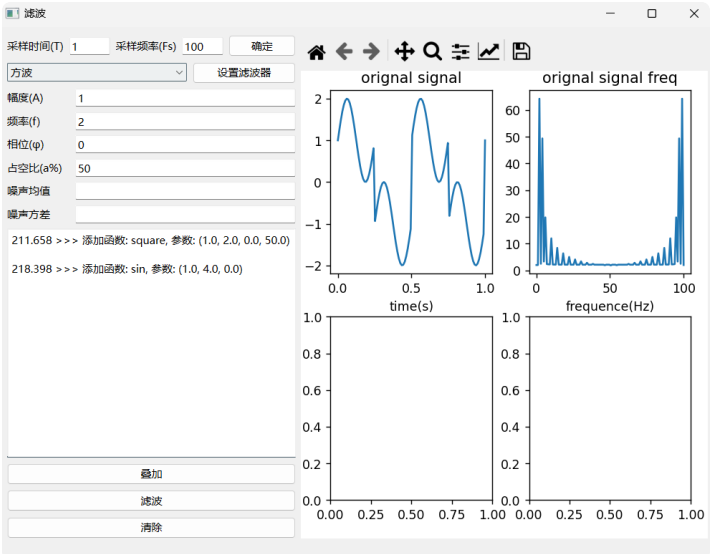
```python
class MySignal:
    def __init__(self) -> None:
        self.forms = []

    def add_formula(self, params):
        if params[0] == "file":
            sig = scio.loadmat(params[1])["s"][0]
            self.forms.append("file", sig)
        else:
            self.forms.append(params)

    def calc_value(self, sample_time, sample_freq):
        dot_number = int(sample_time * sample_freq)
        n = np.linspace(0, sample_time, dot_number)
        v_signal = np.zeros(dot_number)
        for form, para in self.forms:
            if form == "sin":
                v_signal += para[0] * np.sin(2 * np.pi * para[1] * n + para[2])
            elif form == "square":
                v_signal += para[0] * signal.square(
                    2 * np.pi * para[1] * n + para[2], duty=para[3] / 100
                )
            elif form == "gaussian":
                v_signal += para[0] * BasicSignals.awgn(np.ones(dot_number), para[1])
            elif form == "uniform":
                v_signal += para[0] * np.random.randn(dot_number)
            elif form == "file":
                v_signal += np.resize(para, dot_number)
        return n, v_signal

    def calc_freq(self, sample_time, sample_freq, auto_abs_h=True):
        dot_number = int(sample_time * sample_freq)
        _, sig = self.calc_value(sample_time, sample_freq)
        w = np.linspace(0, sample_freq, dot_number)
        h = np.fft.fft(sig)
        if auto_abs_h:
            h = np.abs(h)
        return w, h

    def clear(self):
        self.forms = []
```



正弦波(采样1s)



正弦波(采样2s)



正弦波与方波叠加

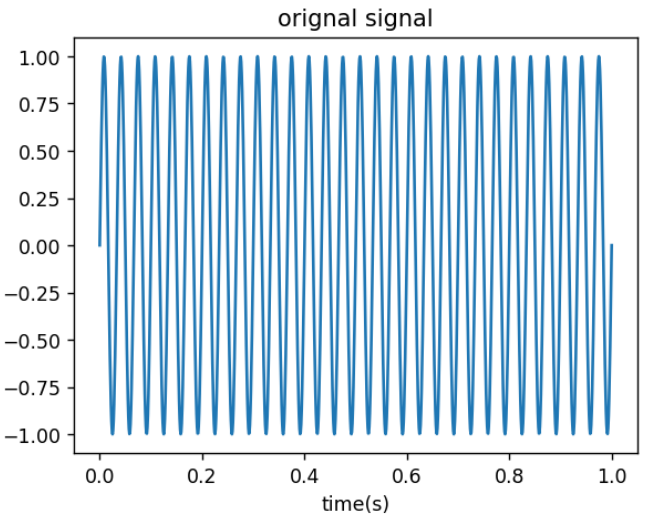
实验三 信号频谱分析

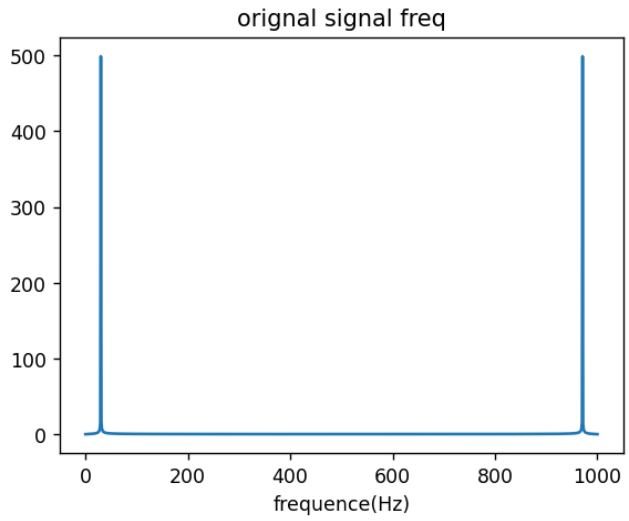
实验要求:

1. 快速傅里叶变换
   a. 正弦波、方波、混合正弦波和两种白噪声的原始波形在GUI窗口呈现。
   b. 对以上波形进行快速傅里叶变换，并在GUI窗口展示频谱分析结果（幅频响应和相位响应），横坐标为频率Hz。
2. 功率谱分析

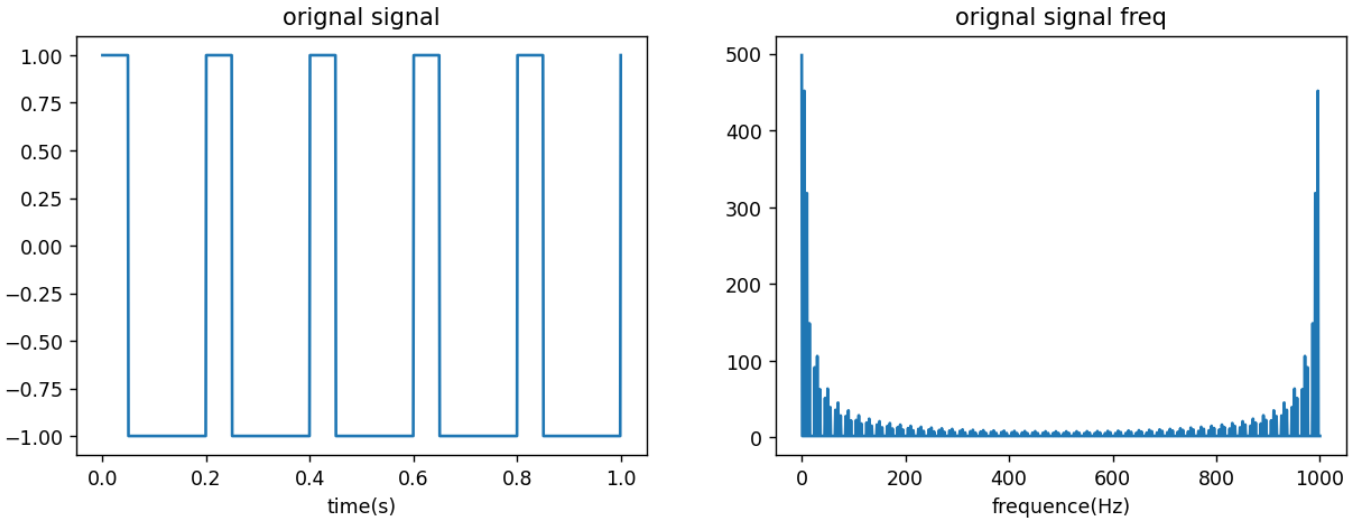
   对以上波形进行功率谱分析，并在GUI窗口展示功率谱分析幅频响应，横坐标为频率HZ。
3. 对加载的ssvep.mat信号进行快速傅里叶变换，在GUI窗口展示信号所含的频率成分是多少Hz,并学习ssvep.mat信号的特点，采样频率为1000Hz。

实验过程:

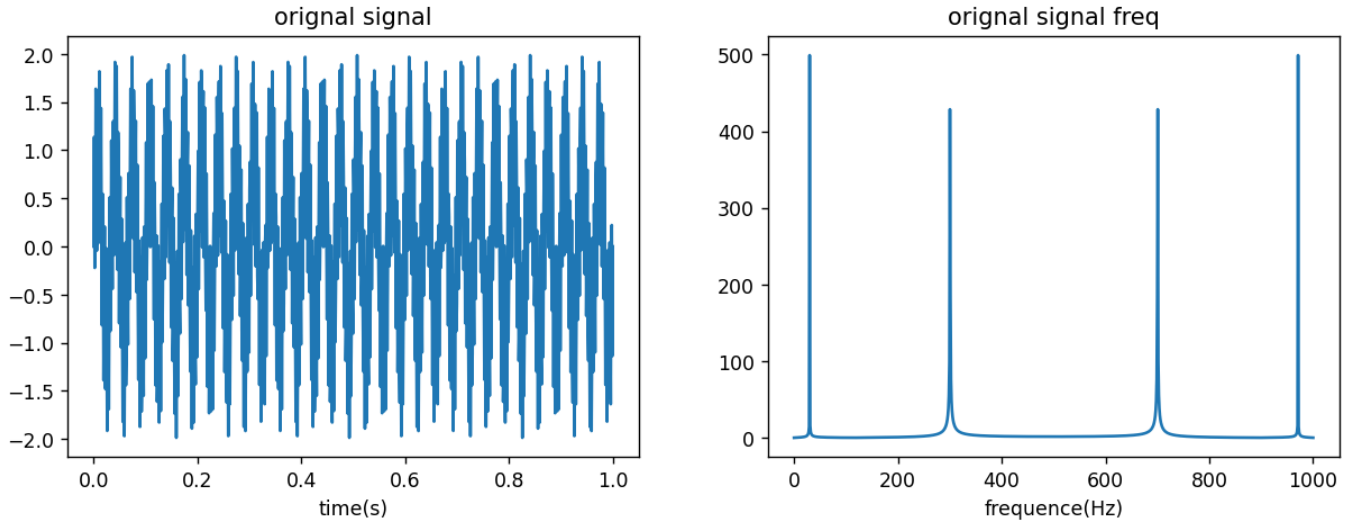




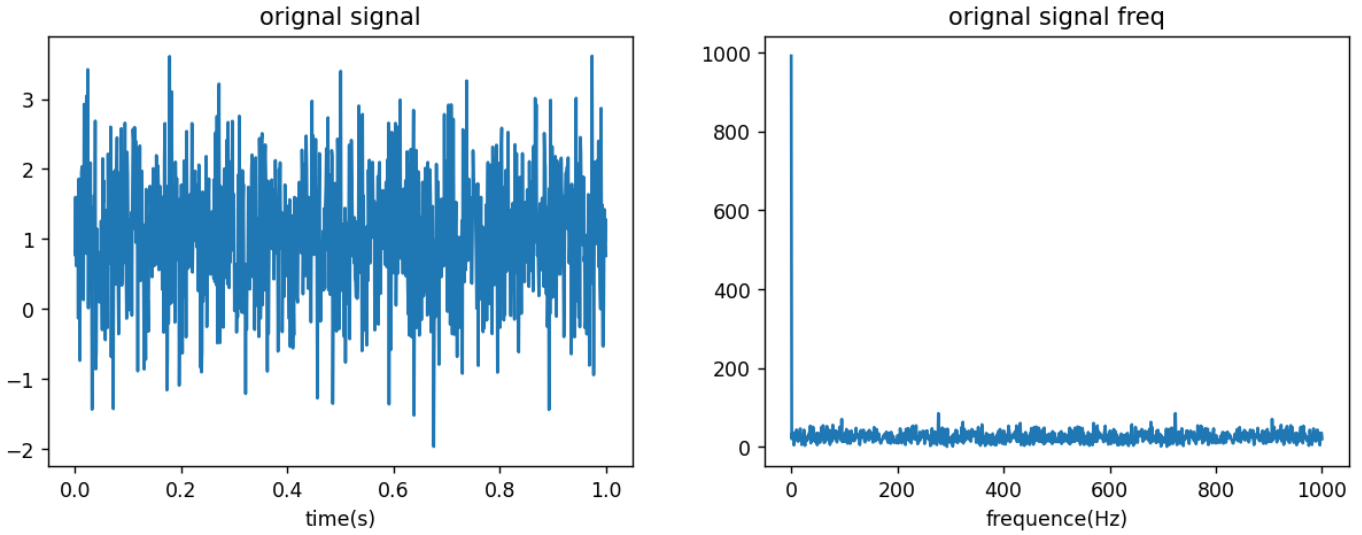
正弦波(30Hz, 采样1s)



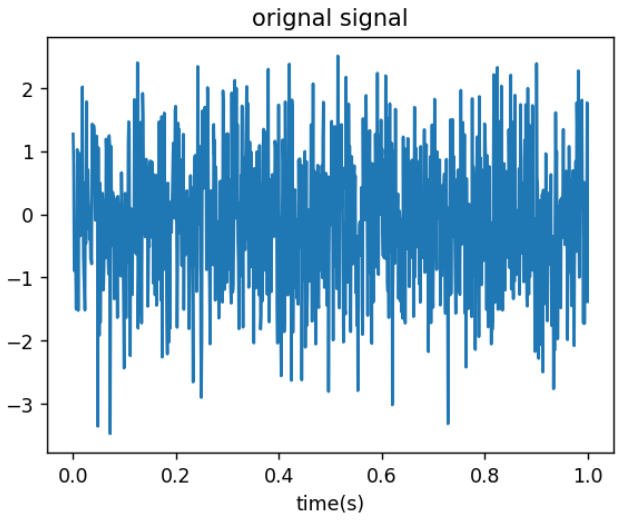
方波(5Hz, 采样1s)

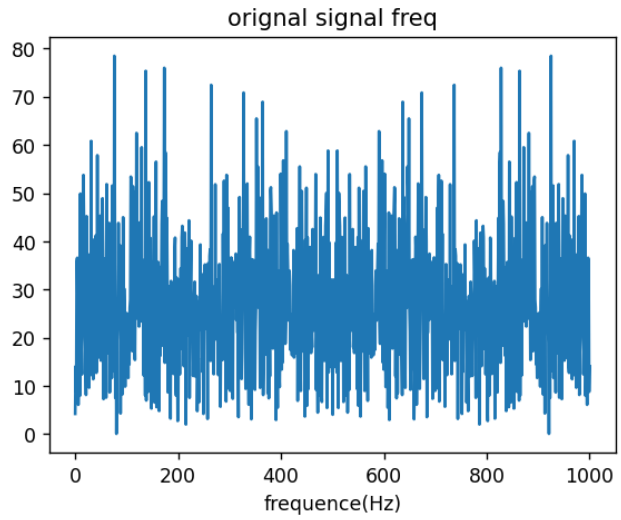


混合正弦波(30Hz, 300Hz, 采样1s)

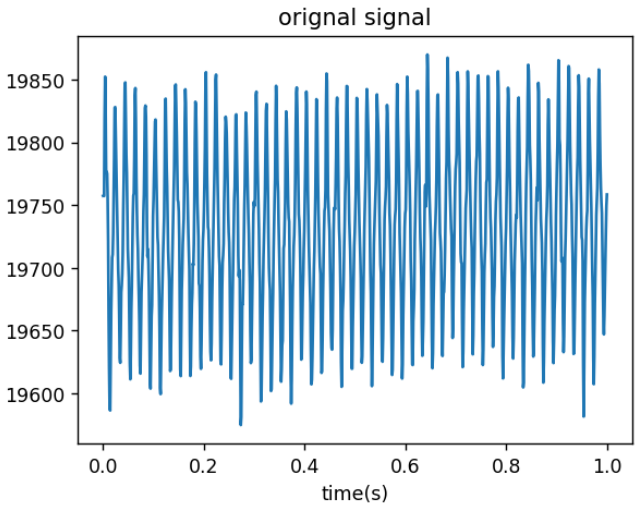


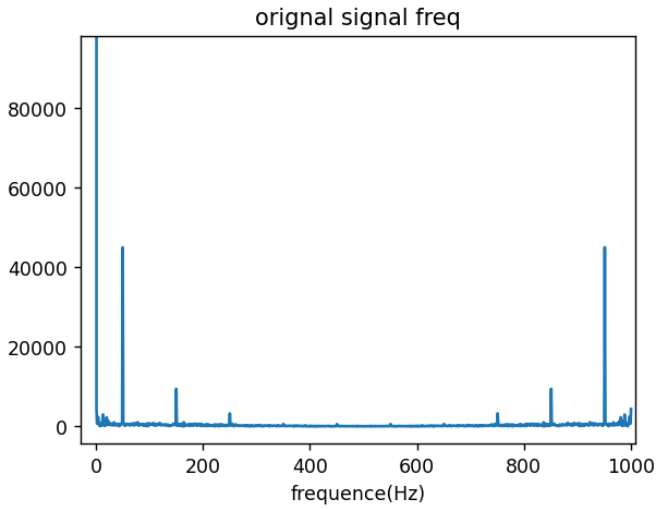
高斯白噪声





均匀白噪声





打开.mat文件

实验四 数字滤波器的设计

实验要求:

1. 设计一个数字低通滤波器，方法是先设计一个R模拟低通滤波器然后数字化。截止频率是100Hz,采样频率1000Hz。
   a. 在GUI窗口展示低通滤波器的幅频响应和相位响应。
   b. 对30Hz与300Hz混合的正弦信号进行低通滤波，在GUI窗口展示其滤波前后结果的时域和频域波形。
   c. 在GUI窗口展示2)中混合正弦信号在滤波前后相位变化，并阐述R滤波器的设计方法，以及滤波器的特点。

2. 设计一个数字高通滤波器，方法是线性相位FR高通滤波器。截止频率是100Hz,采样频率1000Hz。
    a. 在GUI窗口展示高通滤波器的幅频响应及相频响应。
    b. 对30Hz与300Hz混合的正弦信号进行高通滤波，在GUI窗口展示其滤波前后结果的时域和频域波形
    c. 在GUI窗口展示b.中混合正弦信号在滤波前后相位变化，并阐述FR滤波器的设计方法、滤波器的特点及相位的重要性。

实验过程:

设计MyFilter类, 封装常用滤波器函数, 滤波, 显示频率响应, 零极点等基本功能

MyFilter Python

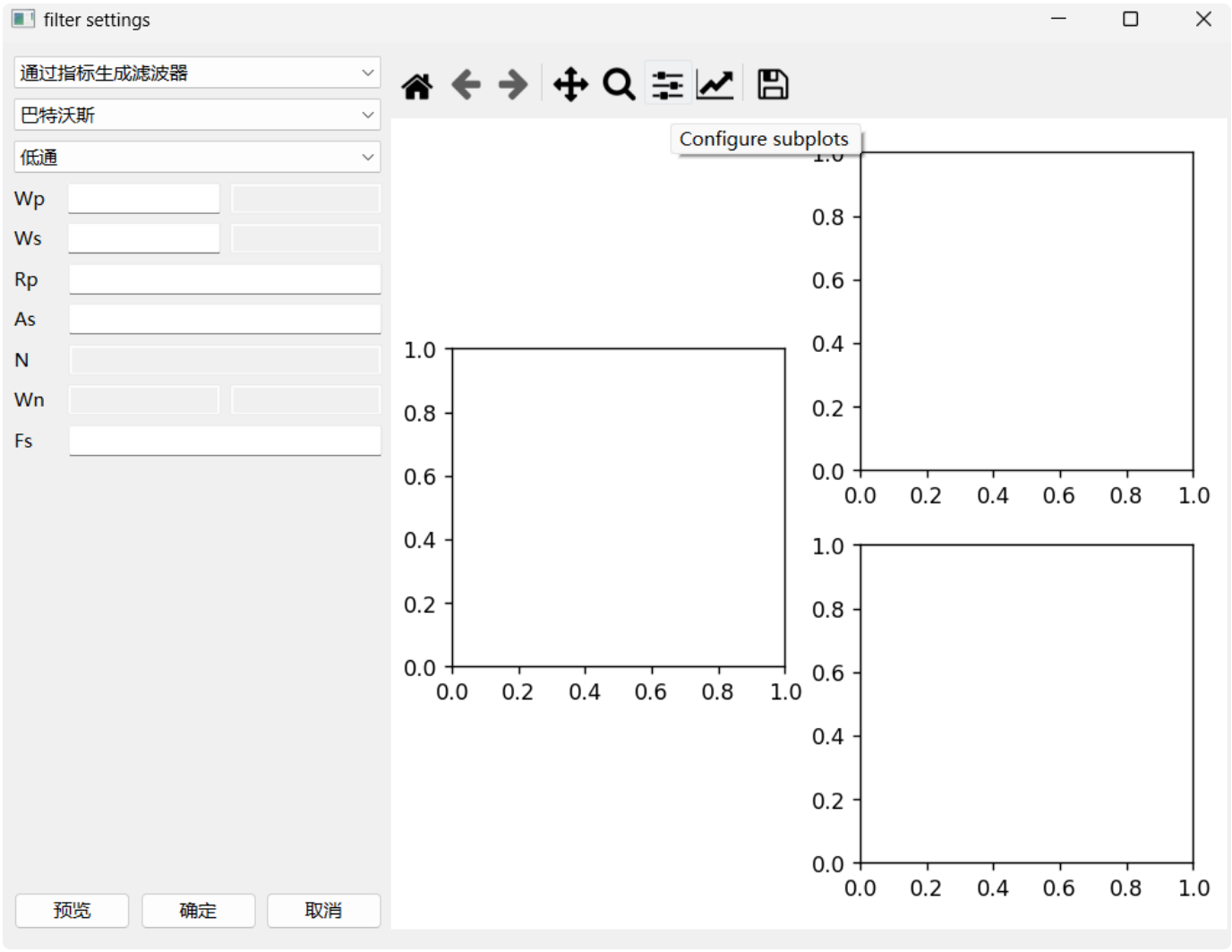
```python
from scipy import signal
import numpy as np


class _MyFilterBase:
    """滤波器基类"""

    def __init__(self) -> None:
        self.reset()

    def reset(self):
        """重置滤波器参数"""
        """滤波器指标"""
        self.pass_band: list[float] = None  # 通带
        self.stop_band: list[float] = None  # 阻带
        self.passband_max_atten: float = None  # 通带最大衰减
        self.stopband_min_atten: float = None  # 阻带最小衰减
        """滤波器参数"""
        # self.type: str = None  # 滤波器类型
        self.zero_pass: bool = None  # 直流信号的通过情况
        self.sample_freq: float = None  # 滤波器采样频率
        self.order_N: float = None  # 滤波器阶数
        self.cutoff_Wn: list[float] = None  # 滤波器截止频率
        """滤波器结果"""
        self.b = None  # 滤波器传递函数(b)
        self.a = None  # 滤波器传递函数(a)
        self.sos = None  # 滤波器传递函数(sos)

        return self

    def _is_goal_set(self):
        return (
            self.pass_band
            and self.stop_band
            and self.passband_max_atten
            and self.stop_band
        )

    def _is_param_set(self):
        return self.order_N and self.cutoff_Wn

    def _check_btype(self):
        if len(self.cutoff_Wn) == 1:
            return "lowpass" if self.zero_pass else "highpass"
        else:
```

```
            return "bandstop" if self.zero_pass else "bandpass"

    @staticmethod
    def _angular_freq_conv(freq, fs: float = None):
        """将真实频率转化为角频率, return: [w1(pi), w2(pi), ...]"""
        if isinstance(freq, (list, np.ndarray)):
            return [_MyFilterBase._angular_freq_conv(i, fs)[0] for i in f
req]
        elif isinstance(freq, (float, int)):
            return [freq * 2 / fs if fs else freq]
        else:
            raise ValueError("频率必须为Num或list[Num]")

    def set_goal(self, wp, ws, ap, as_, fs=None):
        """设置滤波器指标"""
        self.pass_band = self._angular_freq_conv(wp, fs)  # 通带
        self.stop_band = self._angular_freq_conv(ws, fs)  # 阻带
        self.passband_max_atten = ap  # 通带最大衰减
        self.stopband_min_atten = as_  # 阻带最小衰减
        self.zero_pass = self.pass_band[0] < self.stop_band[0]

        return self

    def set_param(self, n, wn, fs=None, zero_pass=True, ap=None, as_=None
):
        if self._is_goal_set():
            raise RuntimeError("滤波器指标已经设置, 使用.reset()重置滤波器")
        # self.type = type_  # 滤波器类型
        self.zero_pass = zero_pass  # 直流信号的通过情况
        self.order_N = n  # 滤波器阶数
        self.cutoff_Wn = self._angular_freq_conv(wn, fs)  # 滤波器截止频率
        self.sample_freq = fs  # 滤波器采样频率
        self.passband_max_atten = ap  # 通带最大衰减
        self.stopband_min_atten = as_  # 阻带最小衰减

        return self

    def filtering(self, sig, use_BA=False):
        if use_BA:
            return signal.filtfilt(self.b, self.a, sig)
        else:
            return signal.sosfiltfilt(self.sos, sig)

    def get_freqz(self, use_BA=False, auto_abs_h=True):
        if use_BA or not isinstance(self.sos, (np.ndarray)):
            w, h = signal.freqz(self.b, self.a)
        else:
            w, h = signal.sosfreqz(self.sos)
```

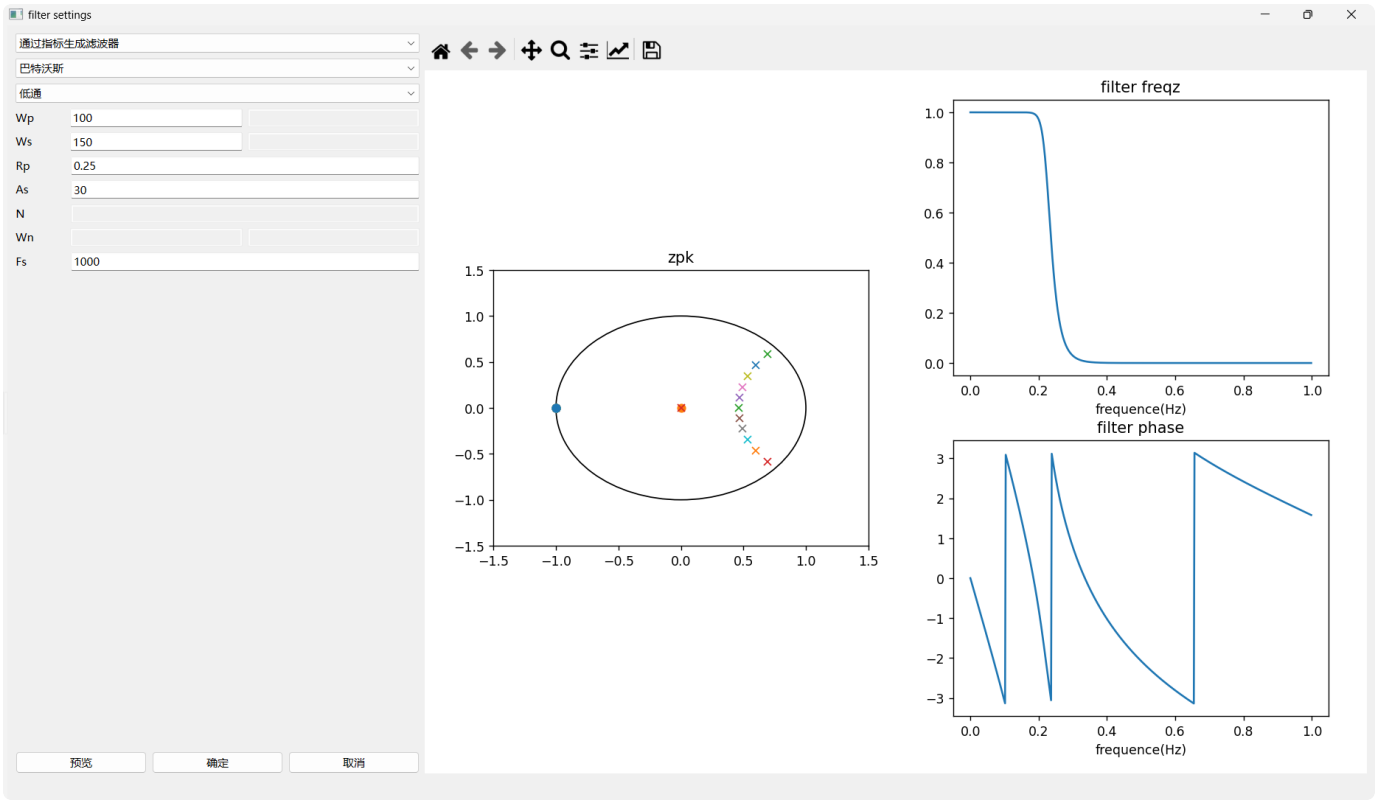
```
92          return (w / np.pi, abs(h) if auto_abs_h else h)
93
94      def get_zpk(self, use_BA=False):
95          if use_BA or not isinstance(self.sos, (np.ndarray)):
96              return signal.tf2zpk(self.b, self.a)
97          else:
98              return signal.sos2zpk(self.sos)
99
100
101 class MyButter(_MyFilterBase):
102     def __init__(self) -> None:
103         super().__init__()
104
105     def _generate_from_signal(self, skip_first_stage=False):
106         if not skip_first_stage:
107             self.order_N, self.cutoff_Wn = signal.buttord(
108                 self.pass_band,
109                 self.stop_band,
110                 self.passband_max_atten,
111                 self.stopband_min_atten,
112             )
113             self.cutoff_Wn = self._angular_freq_conv(self.cutoff_Wn)
114
115         self.b, self.a = signal.butter(
116             self.order_N, self.cutoff_Wn, btype=self._check_btype(), outp
    ut="ba"
117         )
118         self.sos = signal.butter(
119             self.order_N, self.cutoff_Wn, btype=self._check_btype(), outp
    ut="sos"
120         )
121
122     def digitalize(self):
123         if self._is_param_set():
124             self._generate_from_signal(skip_first_stage=True)
125         elif self._is_goal_set():
126             self._generate_from_signal(skip_first_stage=False)
127
128         return self
129
130
131 class MyCheby1(_MyFilterBase):
132     def __init__(self) -> None:
133         super().__init__()
134
135     def _generate_from_signal(self, skip_first_stage=False):
136         if not skip_first_stage:
137             self.order_N, self.cutoff_Wn = signal.cheb1ord(
```

```
                self.pass_band,
                self.stop_band,
                self.passband_max_atten,
                self.stopband_min_atten,
            )
            self.cutoff_Wn = self._angular_freq_conv(self.cutoff_Wn)

        self.b, self.a = signal.cheby1(
            self.order_N,
            self.passband_max_atten,
            self.cutoff_Wn,
            btype=self._check_btype(),
            output="ba",
        )
        self.sos = signal.cheby1(
            self.order_N,
            self.passband_max_atten,
            self.cutoff_Wn,
            btype=self._check_btype(),
            output="sos",
        )

    def digitalize(self):
        if self._is_param_set() and self.passband_max_atten:
            self._generate_from_signal(skip_first_stage=True)
        elif self._is_goal_set():
            self._generate_from_signal(skip_first_stage=False)

        return self


class MyCheby2(_MyFilterBase):
    def __init__(self) -> None:
        super().__init__()

    def _generate_from_signal(self, skip_first_stage=False):
        if not skip_first_stage:
            self.order_N, self.cutoff_Wn = signal.cheb2ord(
                self.pass_band,
                self.stop_band,
                self.passband_max_atten,
                self.stopband_min_atten,
            )
            self.cutoff_Wn = self._angular_freq_conv(self.cutoff_Wn)

        self.b, self.a = signal.cheby2(
            self.order_N,
            self.stopband_min_atten,
```

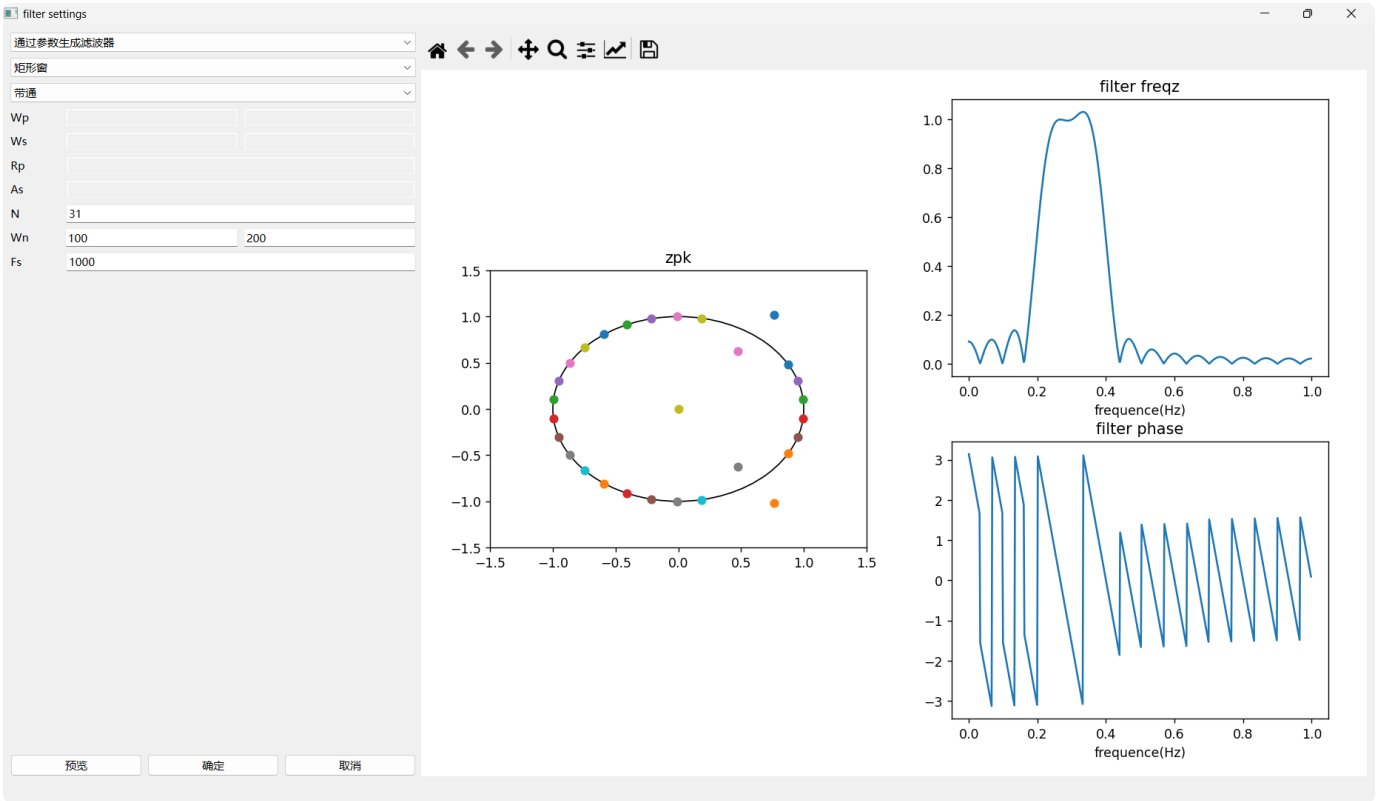
```
            self.cutoff_Wn,
            btype=self._check_btype(),
            output="ba",
        )
        self.sos = signal.cheby2(
            self.order_N,
            self.stopband_min_atten,
            self.cutoff_Wn,
            btype=self._check_btype(),
            output="sos",
        )

    def digitalize(self):
        if self._is_param_set() and self.stopband_min_atten:
            self._generate_from_signal(skip_first_stage=True)
        elif self._is_goal_set():
            self._generate_from_signal(skip_first_stage=False)

        return self


class MyEllip(_MyFilterBase):
    def __init__(self) -> None:
```


filter settings
通过指标生成滤波器
巴特沃斯
低通
Wp
Ws
Rp
As
N
Wn
Fs
预览
确定
取消
Configure subplots
0.0
0.2
0.4
0.6
0.8
1.0

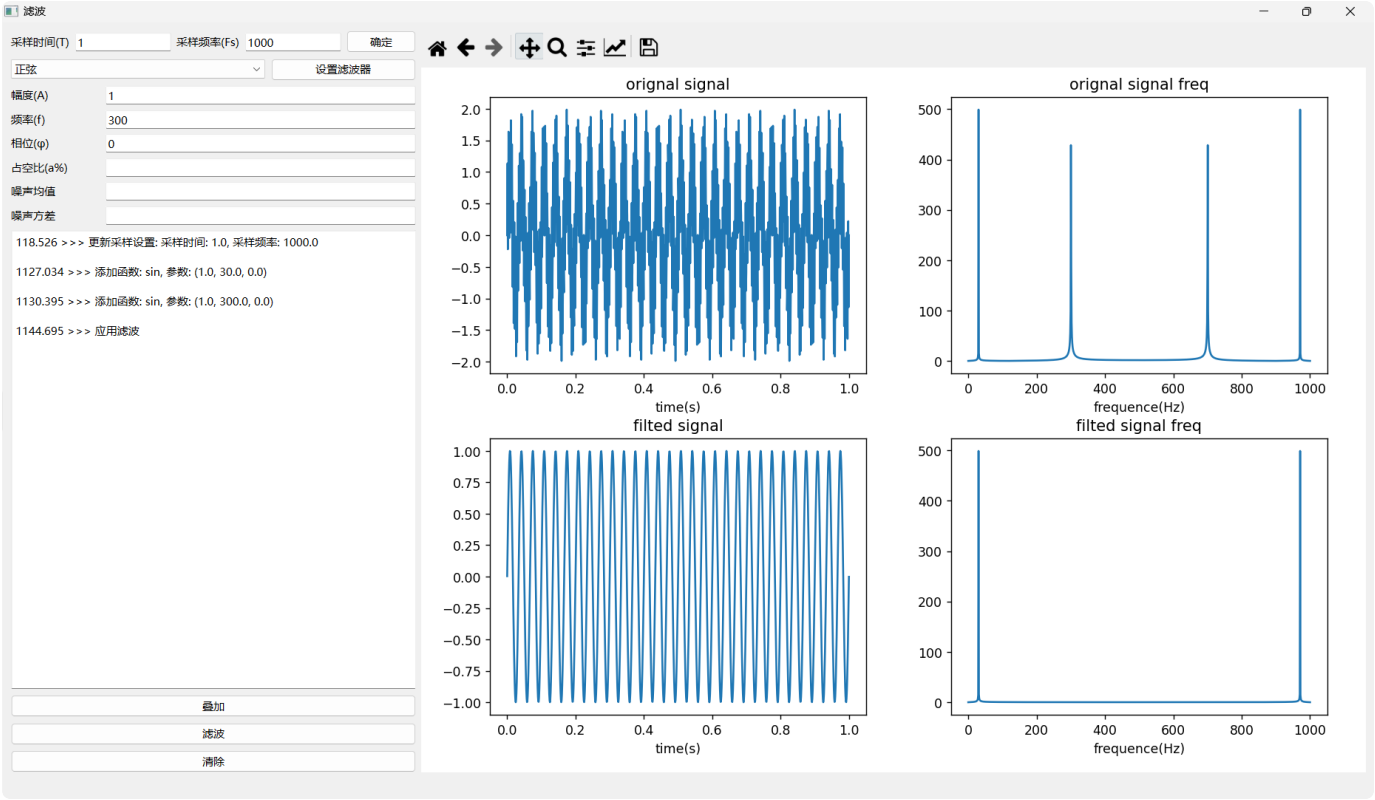
滤波器设置界面展示



巴特沃斯低通滤波器, Wp=100, Ws=150, Rp=0.25, As=30, Fs=1000



矩形窗带通滤波器: N=31, Wn1=100, Wn2=200, Fs=1000



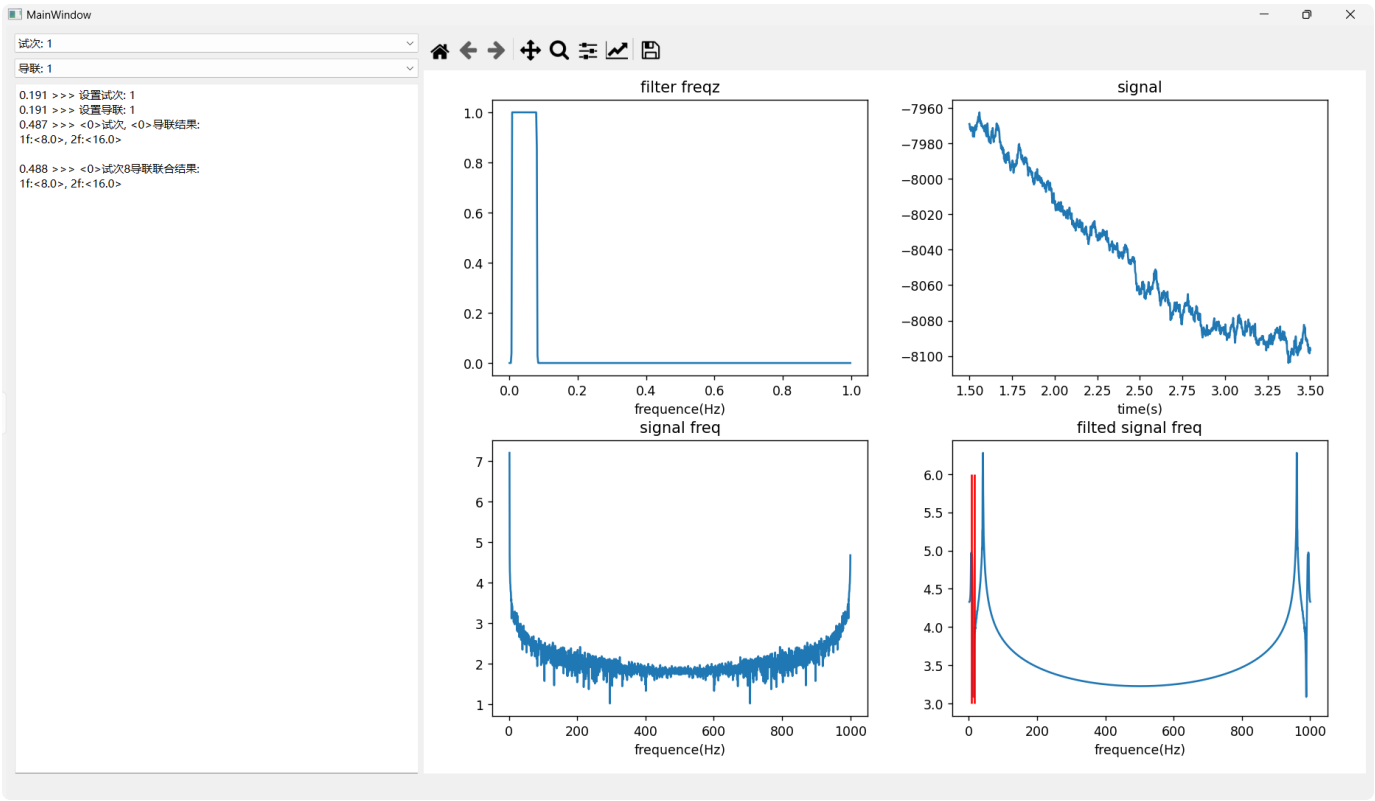
滤波效果

实验五 脑电信号的目标识别

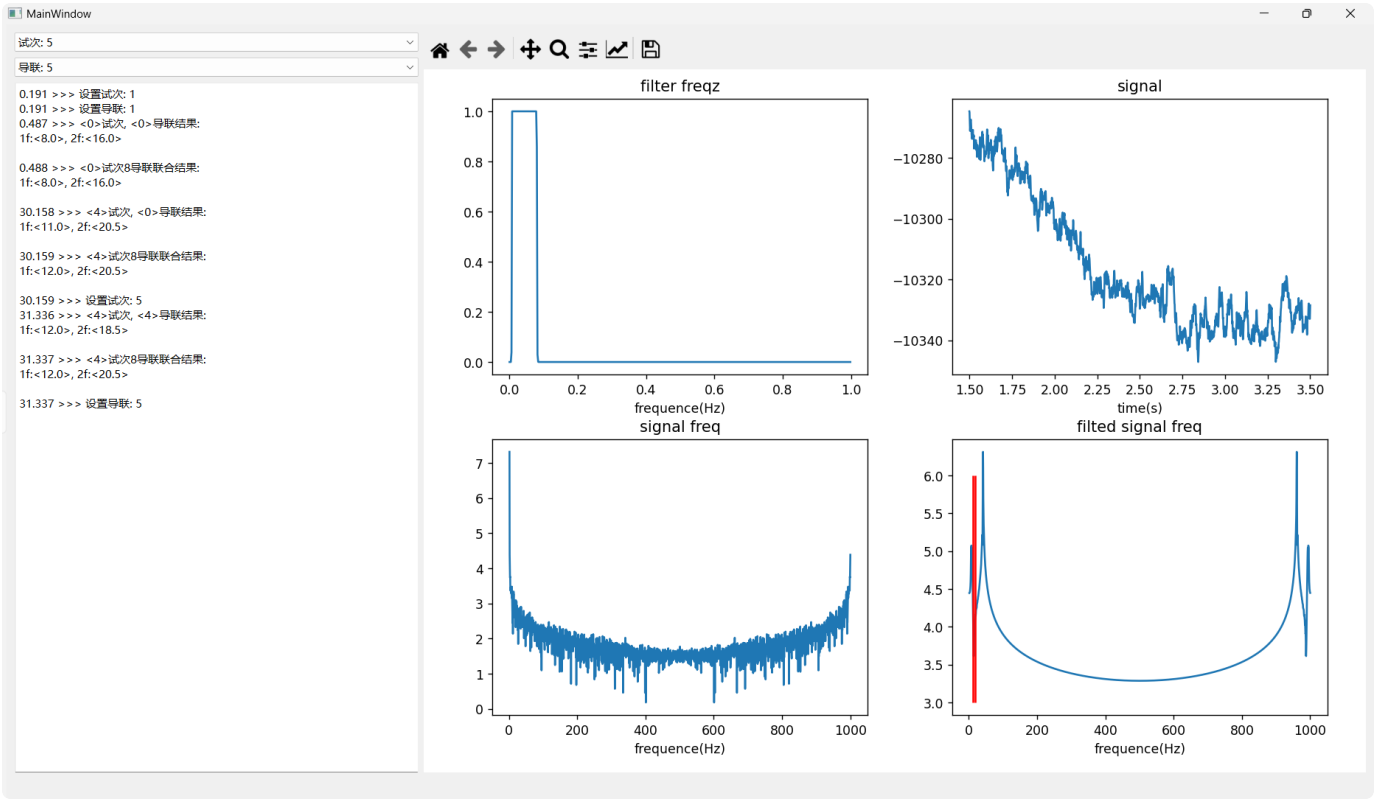
实验要求:

1. 脑电信号进行带通滤波，滤波范围3–40HZ。
2. 利用FFT或功率谱periodogram对每个试次每个通道的脑电信号进行频谱分析，查看7–15Hz范围内最高峰值是多少，并与所给刺激频率比对，8个通道投票最多的目标即为该试次所分类出来的目标结果。另外，也可考虑基倍频联合检测，从而提高目标识别准确率。
3. 对20个试次分别进行目标分类，根据频谱信息判断目标类别并与其真实标签label进行比较，计算准确率。
4. GUI界面呈现滤波器的幅频响应；20个试次一个通道的频谱图（可选一个识别率高的通道），并标出峰值频率；呈现20个试次中每个导联目标分类类别，以及8导联联合目标识别结果。

实验过程:



1试次, 1导联(识别结果见左侧)



5试次, 5导联(识别结果见左侧)

实验六 VR眼动数据的目标识别

实验要求:

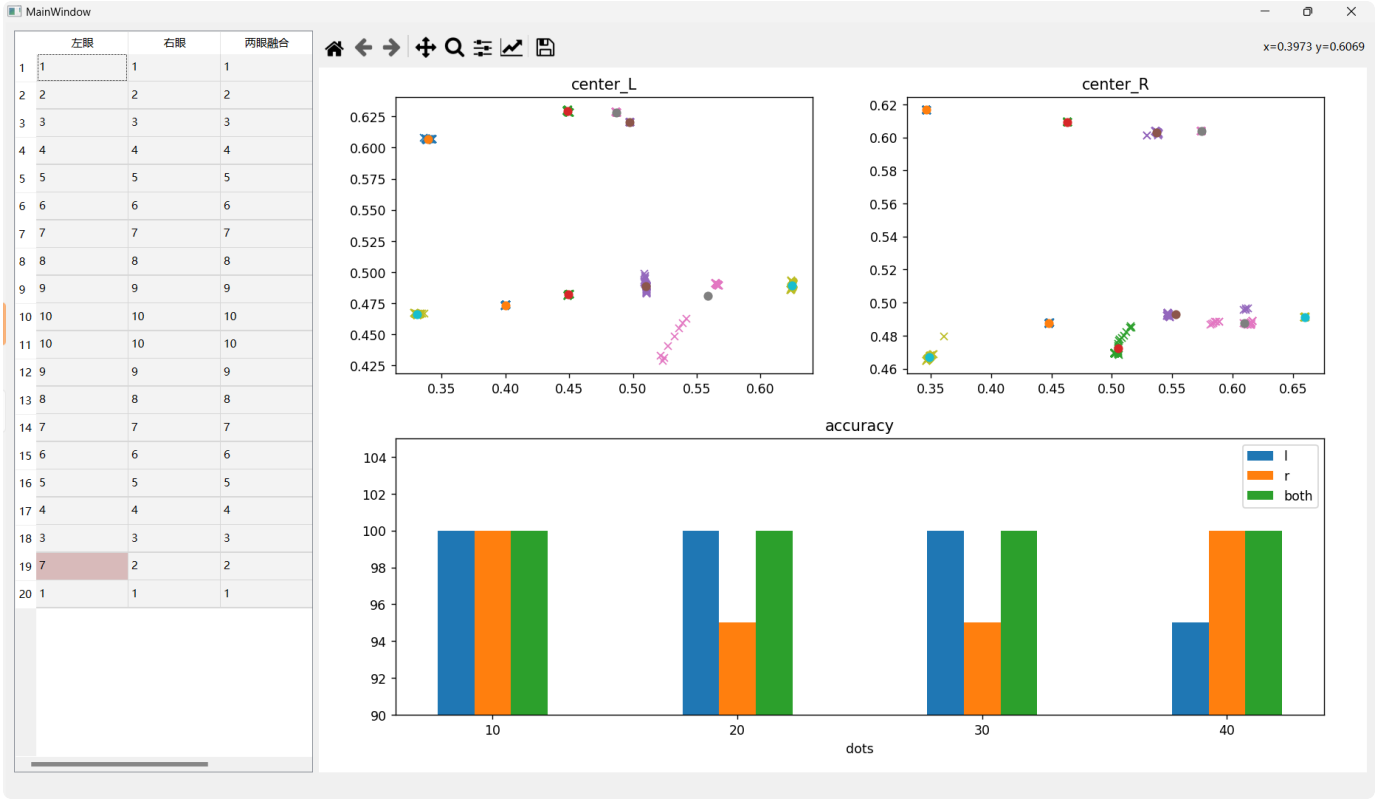
1. 求正式实验阶段目标分类的准确率，左右眼分别计算分类准确率，然后左右眼融合计算分类准确率（各占50%）
2. 分析不同时间点10、20、30、40条件下的分类准确率，并画出曲线。
3. GUI界面呈现校准阶段10个试次各自的中心位置(XY坐标)，以及正式实验阶段20个试次目标分类类别，可表格呈现，包括单左眼、单右眼和左右眼融合的分类结果。另外呈现不同时间点下的分类准确率曲线。

实验过程:

设计KMeans类:

MyKMeans Python

```python
import numpy as np


# sample: (x, y), samples: list[sample]
class MyKMeans:
    def __init__(self) -> None:
        self.centers = {}

    def add_center(self, samples, key):
        x_center = np.mean([i[0] for i in samples])
        y_center = np.mean([i[1] for i in samples])
        self.centers[(x_center, y_center)] = key
        return (x_center, y_center)

    def find_nearest_center(self, sample):
        x_sample, y_sample = sample

        min_distance = np.inf
        nearest_center = None
        for center in self.centers.keys():
            x_center, y_center = center
            distance = (x_center - x_sample) ** 2 + (y_center - y_sample) ** 2
            if distance < min_distance:
                min_distance = distance
                nearest_center = center
        return self.centers[nearest_center]
```



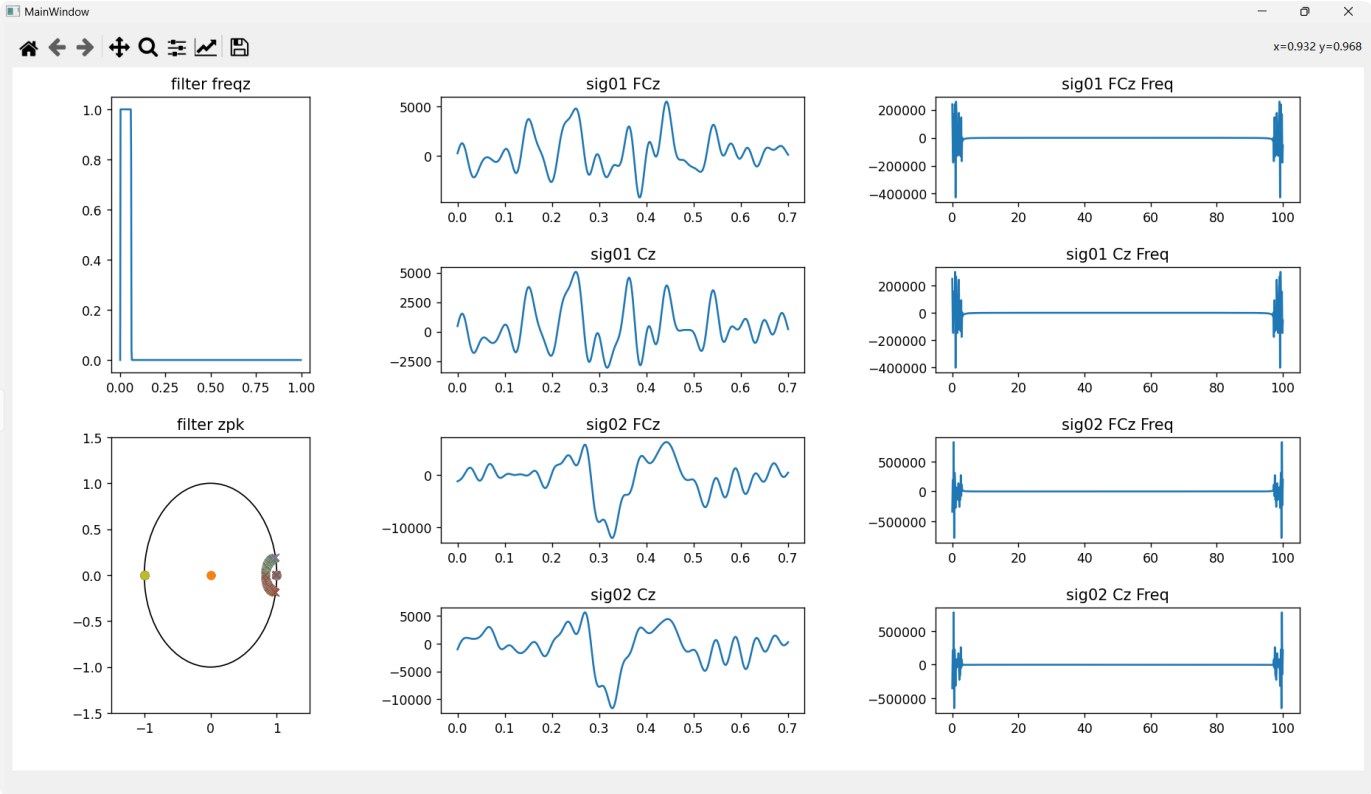
显示结果(center_L/R 中, 'x' 为训练数据, 'o'为聚类中心)

实验七 脑电ERP信号的提取方法

实验要求:

1. 设计数字滤波器，滤除低频和高频噪声。带宽0.5~30Hz。采样率1000Hz,低通截止频率30Hz,高通截止频率0.5Hz。给出低高通滤波器的频率响应。
2. 将每个试次的脑电信号进行滤波，利用时域叠加平均方法提取EP波形，即试次维度累加后平均。
3. 将滤波后每个试次的脑电信号进行频谱分析，然后将所有试次的频率响应进行叠加平均，给出波形。注意横坐标为0–1000Hz。

实验过程:

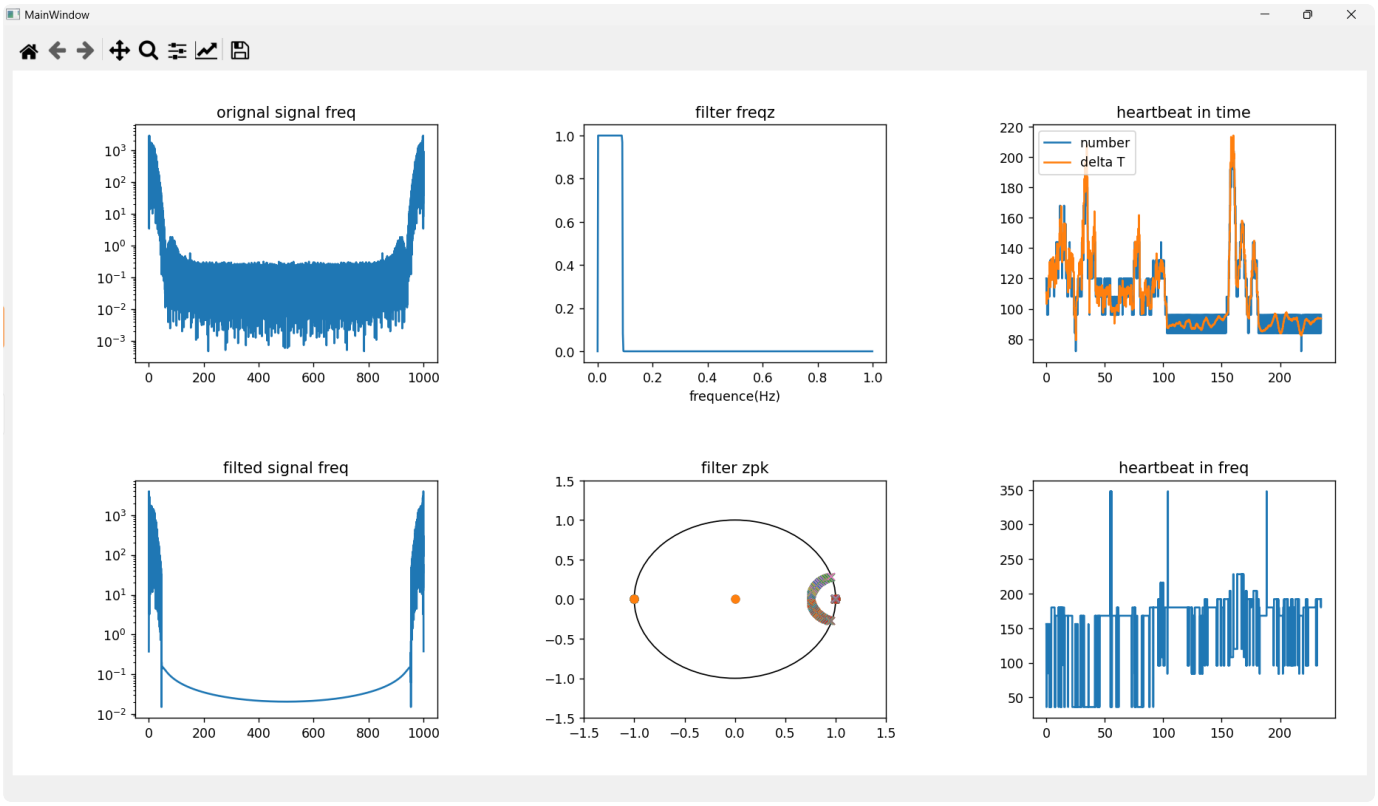


实验八 心电信号心率检测方法

实验要求:

1. 对心电数据进行t频谱分析，给出频谱分析图。
2. 设计高低通数字滤波器，给出高低通滤波器的频率响应。
3. 滤波后的心电信号，频谱分析，并将滤波前后心电信号的波形进行对比。并查看心电信号和噪声所在频段。
4. 利用时域或频域方法实时估计心率。时域方法，一个时间窗5s钟，寻找心电信号R峰对应的时间点，通过时间差和R峰个数计算心率值，然后移动时间窗，步长为1点，逐点计算心率值，最后绘制随时间变化的心率值曲线。频域方法，一个时间窗5s钟的数据进行频谱分析，找出最大峰值对应的频率，根据此信息计算出心率值，然后然后移动时间窗，步长为1点，逐点计算心率值，最后绘制随时间变化的心率值曲线。
5. 比较时域和频域两种方法的差异。

实验过程:



实验九 空域滤波器设计

实验要求:

题目要求:

传感器均匀排布的天线接收到空间中两个同频相位不一样的正弦信号，要求设计一个空域滤波器，提取其中一路信号同时抑制另外一路信号。具体实验步骤如下：

1. 写出天线的接收信号模型（矩阵形式）
2. 设计空域滤波器权矢量w
3. 仿真画出结果图，展示滤波效果，包括空域滤波器的方向图、以及空域滤波器的接和最终输出信号的对比图
4. 写出此方法的优缺点

仿真实验设计:

本实验对具体参数的设计如下：假设一个均匀线阵有12阵元，布阵；取100个快拍估计协方差矩阵R，两个信号为同频信号相位不同的信号，其中期望信号的入射方向为30°，干扰信号的入射方向为20°

实验过程:

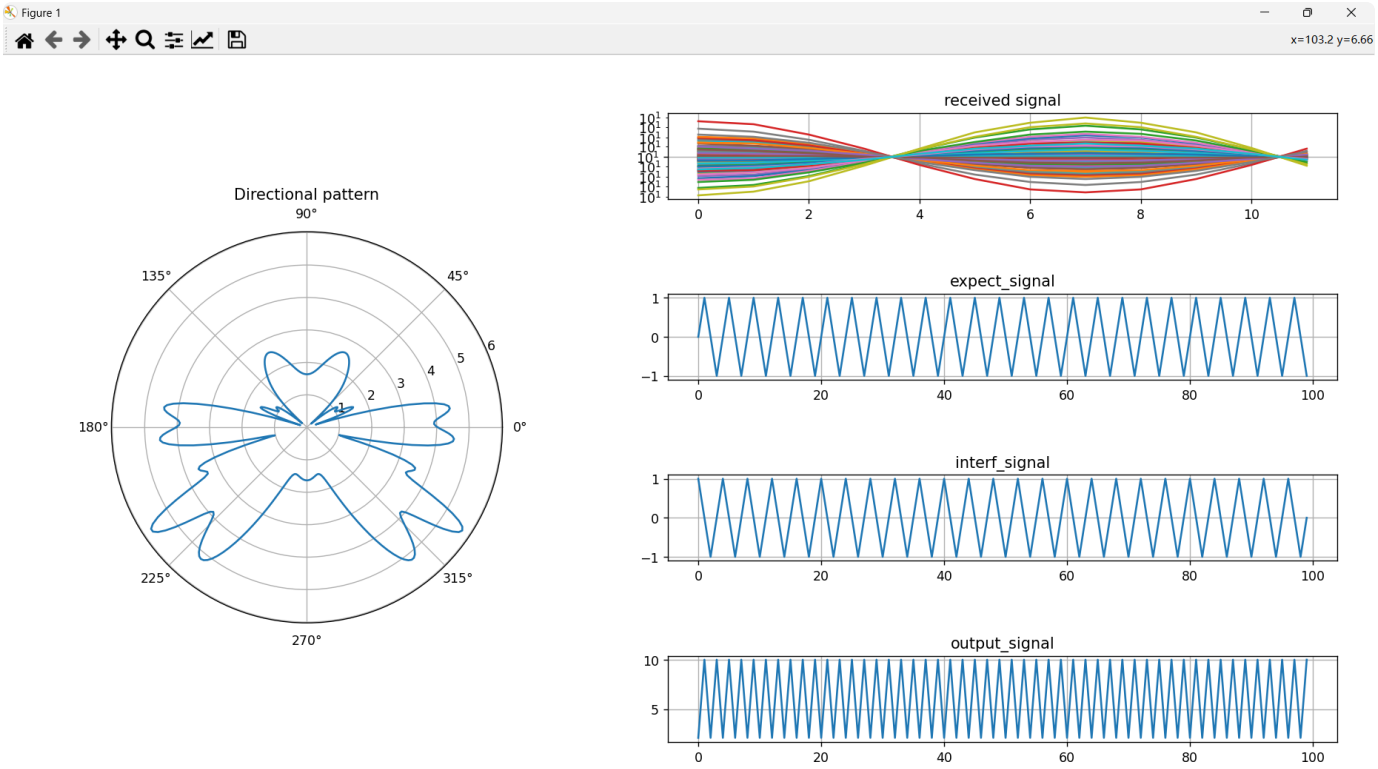


Figure 1
x=103.2 y=6.66
Directional pattern
90°
135°
45°
180°
0°
225°
315°
270°
1
2
3
4
5
6
received signal
expect_signal
interf_signal
output_signal