理想滤波器动态演示;调制与解调动态演示

理想滤波器动态演示

Basic Filters

Basic Filters

- 我们首先实现基本滤波器，该滤波器实现

```
1.接受sympy表示的符号函数

2.将传入的符号函数转换为数值函数

3.实现该数值函数的可视化展示，生成静态图像

4.包括原始信号，相位谱，幅度谱，中心化后的幅度谱
```

- Basic Filters 类接收的参数，根据下面的参数即可实现完整功能。
  - 符号函数
  - 信号持续时间
  - 信号频率
- 代码演示

```
class BasicFilter:
    def __init__(self,sym_exper , sample_frequency, T):
        # self.func = lambdify(t, sin(t), modules='numpy')
        # 将符号函数转换为数值函数
        func_symbolic = lambdify(t ,sym_exper ,modules='numpy')
```

Ideal Filter

Ideal Filter

- Ideal Lowpass Filter
- Ideal Highpass Filter
- Ideal Bandpass Filter
- Ideal Bandstop Filter
- 我们这里所有的理想滤波器均继承BasicFilter类，实现了基本的滤波器功能。
- 额外的功能为动态演示滤波器的效果。需要用户输入滤波器的截止频率带。

- 我们使用频率截止法实现理想滤波器的动态演示。通过numpy中的布尔数组实现,如下图所示。通过这种方式我们即可以实现截止频率的动态演示。

```
filter_response = np.zeros(len(self.fft_freq))
        # H(f) = 1 if |f| <= cutoff_freq, else 0
        # 这是np数组操作
        filter_response[(self.range_size + cut_frequency >= self.fft_freq) &
(self.fft_freq >= cut_frequency)] = 1.0

        # 在频域应用滤波器（相乘）
        filtered_fft = self.old_fft_vals * filter_response

        # 通过逆FFT得到滤波后的时域信号
        filtered_signal = np.fft.ifft(filtered_fft).real
```

调制与解调

AM调制与解调

- AM调制

```
"""
        AM调制
        s(t) = A_c * [1 + m * m(t)] * cos(2πf_c*t)
        """
        # 生成载波信号
        self.carrier_signal = np.cos(2 * np.pi * self.carrier_freq * self.t_vals)

        # 调制
        self.modulated_signal = (1 + self.modulation_index *
self.normalized_baseband) * self.carrier_signal

        # 计算调制信号的FFT
        self.modulated_fft = np.fft.fft(self.modulated_signal)
        self.modulated_amplitude = np.abs(self.modulated_fft)
```

使用公式$s(t) = A_c * [1 + m * m(t)] * cos(2πf_c*t)$来实现信号的调制。其中$A_c$为载波振幅，$f_c$为载波频率，$m(t)$为调制信号，$m$为调制指数。

- AM解调

```
"""
        AM解调
        参数:
            method: 'envelope' (包络检波) 或 'coherent' (相干解调)
        """
        if self.modulated_signal is None:
            self.modulate()
```

```
        if method == 'envelope':
            # 包络检波：使用希尔伯特变换
            from scipy.signal import hilbert
            analytic_signal = hilbert(self.modulated_signal)
            envelope = np.abs(analytic_signal)

            # 去除直流分量
            self.demodulated_signal = envelope - np.mean(envelope)

        elif method == 'coherent':
            # 相干解调：与载波相乘后低通滤波
            # 与载波相乘
            demod_temp = self.modulated_signal * self.carrier_signal * 2

            # 低通滤波（频域截断）
            demod_fft = np.fft.fft(demod_temp)
            filter_response = np.zeros(len(self.fft_freq))
            # 低通滤波器截止频率设为载波频率的1/5
            cutoff = self.carrier_freq / 5
            filter_response[np.abs(self.fft_freq) <= cutoff] = 1.0

            filtered_fft = demod_fft * filter_response
            self.demodulated_signal = np.fft.ifft(filtered_fft).real

        # 计算解调信号的FFT
        self.demodulated_fft = np.fft.fft(self.demodulated_signal)
        self.demodulated_amplitude = np.abs(self.demodulated_fft)

        return self.demodulated_signal
```

- 希尔伯特变换：使用包络检波的方式，其核心原理是利用了希尔伯特可以将是实信号分解成正交的复信号的原理。

AM信号的表达式为$s(t) = A_c * [1 + m * m(t)] * cos(2πf_c*t)$,使用希尔伯特变换后，信号变为$i(t) = A_c * [1 + m * m(t)] * sin(2πf_c*t)$。将原信号与$hilbert$信号结合，得到$z(t) = s(t) + j * i(t)$,可得其包络为$|Ac + m(t)|$，去除直流信号可得解调信号。

- 相干解调：在相干解调中，接收端需要一个与发送端载波完全同步（同频同相） 的本地载波。将接收到的AM信号与该载波相乘，实现下变频，再通过低通滤波器滤除高频分量，最终得到原始调制信号。其数学本质是利用了傅里叶变换的频移特性。

原始调制信号为$s(t) = A_c * [1 + m * m(t)] * cos(2πf_c*t)$,同步载波信号为$c(t) = cos(2πf_c*t)$。实现相乘操作$z(t) = s(t) * c(t) = A_c * [1 + m * m(t)] * cos(2πf_c*t) * cos(2πf_c*t)$，利用三角函数变形$cos(x)**2 = 1/2 * (1 + cos(2x))$, $z(t)$变形后为$1/2 *[Ac + m(t)]*[1 + cos(2πf_c*t)]$, 此时低频分量为$[1/2 * [Ac + m(t)]]$;高频分量为$1/2 * [Ac + m(t)] * cos(2πf_c*t)$。

此时对信号$z(t)$进行低通滤波，截止频率设为$carrier_freq/5$，即可得到解调信号。

FM调制与解调

- FM调制：

```
"""
        FM调制
        瞬时频率: f(t) = f_c + k_f * m(t)
        瞬时相位: ϕ(t) = 2π∫f(t)dt = 2πf_c*t + 2πk_f*∫m(t)dt
        """
        # 计算基带信号的积分（使用累积求和近似）
        dt = self.T / self.N
        integral_m = np.cumsum(self.normalized_baseband) * dt

        # 瞬时相位
        phase = 2 * np.pi * self.carrier_freq * self.t_vals + \
                2 * np.pi * self.freq_deviation * integral_m

        # FM调制信号
        self.modulated_signal = np.cos(phase)

        # 计算调制信号的FFT
        self.modulated_fft = np.fft.fft(self.modulated_signal)
        self.modulated_amplitude = np.abs(self.modulated_fft)
```

先计算瞬时频率$f(t) = f_c + k_f * m(t)$，再计算瞬时相位$\phi(t) = 2π∫f(t)dt = 2πf_c*t* + *2πk_f*∫m(t)dt$，最后使用$cos(\phi(t))$得到FM调制信号。

- FM解调：

```
"""
        FM解调 - 使用微分检波法
        基本思想：瞬时频率 = d(ϕ(t))/dt
        """
        if self.modulated_signal is None:
            self.modulate()

        # 使用希尔伯特变换获取解析信号
        from scipy.signal import hilbert
        analytic_signal = hilbert(self.modulated_signal)

        # 计算瞬时相位
        instantaneous_phase = np.unwrap(np.angle(analytic_signal))

        # 计算瞬时频率（相位的微分）
        dt = self.T / self.N
        instantaneous_freq = np.diff(instantaneous_phase) / (2 * np.pi * dt)

        # 补齐长度
        instantaneous_freq = np.append(instantaneous_freq, instantaneous_freq[-1])

        # 去除载波频率，得到基带信号
        self.demodulated_signal = instantaneous_freq - self.carrier_freq
```

```
        # 归一化到合理范围
        self.demodulated_signal = self.demodulated_signal / self.freq_deviation

        # 计算解调信号的FFT
        self.demodulated_fft = np.fft.fft(self.demodulated_signal)
        self.demodulated_amplitude = np.abs(self.demodulated_fft)
```

FM信号表达式为$s(t) = A_c * cos(2πf_ct + 2πk_f∫m(t)dt)$。 其中FM的瞬时频率为$f(t) = f_c + k_f * m(t)$，包含了调制信号的信息。 所以，FM解调的基本思想是利用瞬时频率的微分来恢复调制信号。我们先使用希尔伯特变换获取解析信号，再计算瞬时相位，最后利用瞬时频率的微分来恢复调制信号。

希尔伯特变换后的解析信号为$z(t) = s(t) + j * i(t)$，其中$i(t)$为$sin(2πf_ct)$。*解析信号的相位是$2πf_c*t + 2πk_f*∫m(t)dt$。对相位求微分，即可得到瞬时频率$f(t) = f_c + k_f * m(t)$。这时候，我们只需要去除载波频率f_c，即可得到原始调制信号$m(t)$。

PM调制与解调

- PM调制：

```
    """
        PM调制
        瞬时相位: φ(t) = 2πf_c*t + k_p*m(t)
        """
        # 瞬时相位
        phase = 2 * np.pi * self.carrier_freq * self.t_vals + \
                self.phase_deviation * self.normalized_baseband

        # PM调制信号
        self.modulated_signal = np.cos(phase)

        # 计算调制信号的FFT
        self.modulated_fft = np.fft.fft(self.modulated_signal)
        self.modulated_amplitude = np.abs(self.modulated_fft)

        return self.modulated_signal
```

PM调制的瞬时相位为$\phi(t) = 2πf_ct + k_pm(t)$，使用$cos(\phi(t))$得到PM调制信号。

- PM解调：

```
    """
        PM解调 - 通过相位检波
        """
        if self.modulated_signal is None:
            self.modulate()

        # 使用希尔伯特变换获取解析信号
        from scipy.signal import hilbert
        analytic_signal = hilbert(self.modulated_signal)
```

```
        # 计算瞬时相位
        instantaneous_phase = np.unwrap(np.angle(analytic_signal))

        # 去除载波相位
        carrier_phase = 2 * np.pi * self.carrier_freq * self.t_vals
        self.demodulated_signal = instantaneous_phase - carrier_phase

        # 归一化
        self.demodulated_signal = self.demodulated_signal / self.phase_deviation

        # 计算解调信号的FFT
        self.demodulated_fft = np.fft.fft(self.demodulated_signal)
        self.demodulated_amplitude = np.abs(self.demodulated_fft)
```

PM解调的基本思想与FM解调类似。在PM解调中，我们先使用希尔伯特变换获取解析信号，再计算瞬时相位，PM的瞬时相位为 $\phi(t) = 2πf_ct + k_pm(t)$ 。我们只需要去除载波相位 $2πf_c*t$ ，即可得到原始调制信号 $m(t)$ 。