滤波算法使用手册

2023 贝加莱技术团队

发布日期　2023 年 3 月 15 日
版本号　0.4.0

Contents

1. 算法_滤波算法使用手册

基本信息

本手册包含了常用的滤波算法，以及贝加莱标准库中的相关功能块的使用方法，最后提供了一个样例程序以供参考使用。

相关整理内容来源于标准化功能块 V4.15，重新整理并增加新部分后，包含以下内容：

- 惯性滤波
- 滑动均值滤波
- 基于截止频率的滤波方法
- FIR 数字低通滤波器
- 现代滤波技术
- 样例程序

点赞与反馈

如果觉得好用，请扫一扫我们一个反馈。

- 【腾讯文档】贝加莱中文 CHM 调查表
- https://docs.qq.com/form/page/DYlJlSkdlYUp6dm9o

- 内部维护链接：https://confluence.br-automation.com/pages/viewpage.action?pageId=212997713

1.1. 更新日志

版本号	变更日志	修改人	修改日期
V0.01.0	标准化功能块 V4.15 内容	刘柏严、穆珊珊、董俊清、汪其锐	
V0.02.0	整理原始内容：惯性滤波、均值滤波、巴塞尔、巴特沃斯	王宇鑫	2023.02.07
V0.03.0	（1）整理原始内容：Notch、BiQuad、FIR 滤波、MTFilter 库功能块。 （2）增加现代滤波技术的内容，包括观测器和 Kalman 滤波。	王宇鑫	2023.02.08
V0.04.0	（1）更新了 AS 样例程序	王宇鑫	2023.02.08

1.2. 惯性滤波

1. 基本信息

编写人	刘柏严	**审核人**	穆珊珊
算法归属	信号处理	**应用对象**	传感器
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**

V0.1	2023.02.07	整理标准化功能块 V4.15 版本原始内容	王宇鑫

2. 算法简介

惯性滤波（延迟滤波）是自动化项目最常用的算法之一，主要针对模拟量输入进行低通滤波，使输入信号更平滑。可分为一阶惯性滤波和高阶惯性滤波，在此介绍一阶惯性延迟滤波。

3. 内容

离散化公式为：

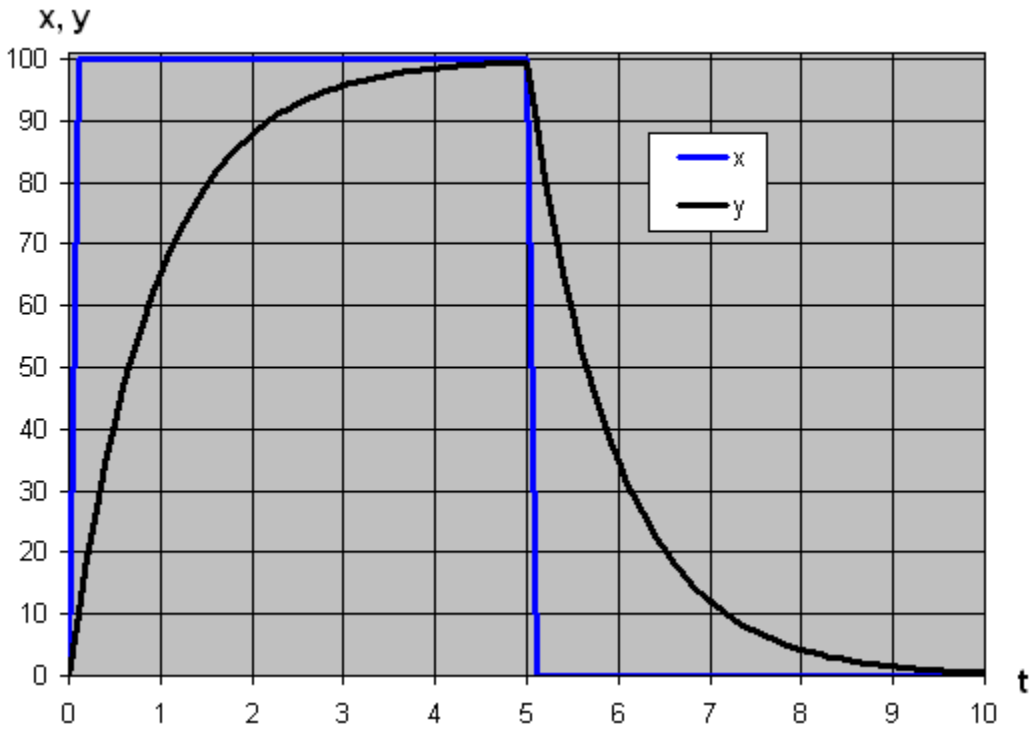
$$y(n) = c_0 * x(n) + (1 - c_0) * y_{n-1}$$

$$c_0 = \frac{T_a}{T_f}, 0 < c_0 < 1$$

对于公式（1），其中T_a表示采样时间（任务周期），T_f表示滤波时间。

滤波时间的物理含义是对于一个阶跃信号，在该时间（T_f）时达到阶跃的 65%左右。

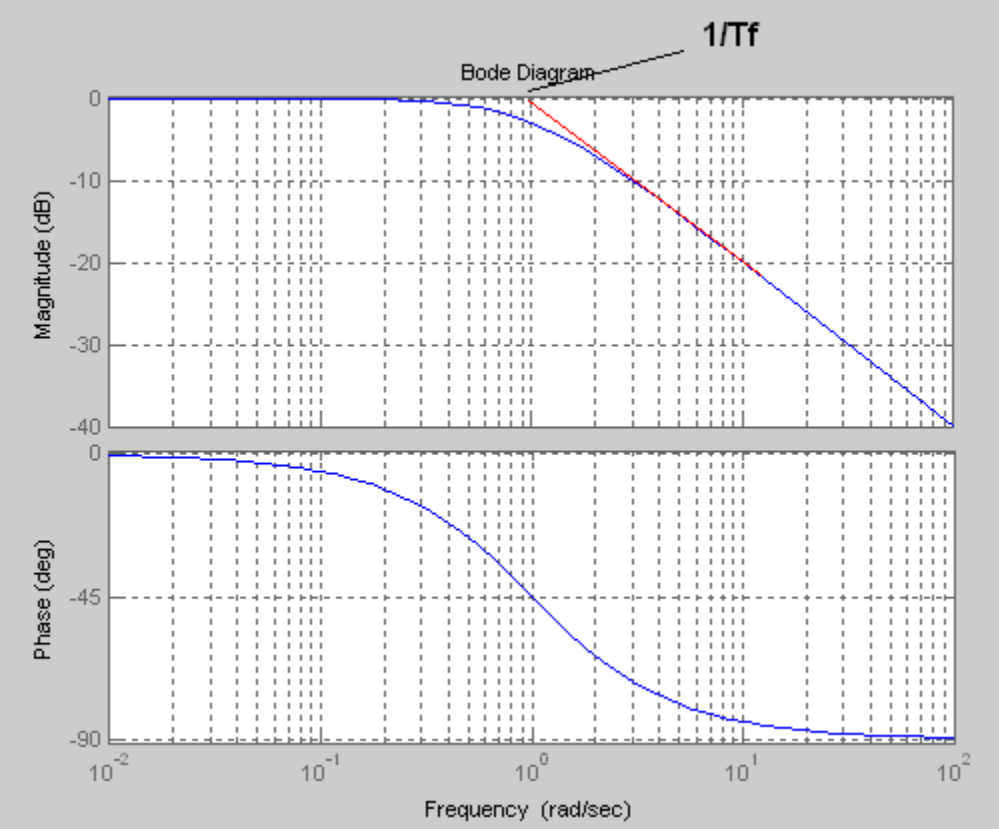
如下图滤波时间为 1s,对于阶跃 100 的响应。



如何判断滤波效果：滤波时间越大，则信号越平稳，但是延迟越多。

所以应该根据实际项目采样，滤波，然后比较看结果是否符合自己要求。

如果从频域（bode 图理解）可以认为对于频率小于$1/T_f$的信号没有衰减，对于频率大于$1/T_f$的信号会逐渐衰减。



4．程序实现/应用举例

方法 1：一阶惯性延迟滤波可以通过 LoopConR 库中的 LCRPT1()功能块实现，见 使用 LCRPT1 实现一阶惯性延迟滤波 。

方法 2：使用 MTBasics 库中的一阶惯性环节 MTBasicsPT1()功能块实现，见 使用 MTBasicsPT1 实现一阶惯性延迟滤波 。

方法 3： 自己编程。需要注意首先要取得任务周期（使用 RTinfo 函数），然后计算系数c_0，最后带入公式。例如，任务时间 10ms，滤波时间 100ms，则$c_0 = 0.1, y = 0.9 * y_{old} + 0.1 * x$。

1.2.1. 使用 LCRPT1 实现一阶惯性延迟滤波

功能块说明

该功能块位于 LoopConR 库中，可实现一阶惯性延迟低通滤波。

计算公式如下：

$$y = y\_old + (x - y\_old) \cdot \frac{dt}{t}$$

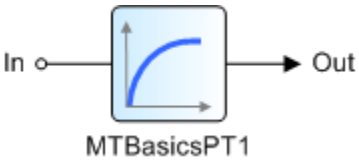
dt 是，调用该功能块 CPU 的扫描周期。

I/O 参数说明

I/O	参数名	数据类型	说明
IN	enable	BOOL	当 enable = TRUE 时，功能块使能。
IN	x	REAL	输入信号。
IN	t	REAL	时间常数，延迟时间，单位：s，该数值需要大于 COU 的扫描周期。
IN	y_set	REAL	y 的预设值。
IN	set	BOOL	设置默认值。 • 当 set = TRUE，那么 y_set 的值 将直接输出到输出信号 y，输入值 x 将不起作用； • 当 set = FALSE 时，y 才会按照公式计算，y_set 将作为 y_old 参与第一次计算。
OUT	status	UINT	功能块错误报警。
OUT	y	REAL	输出信号。

1.2.2. 使用 MTBasicsPT1 实现一阶惯性延迟滤波

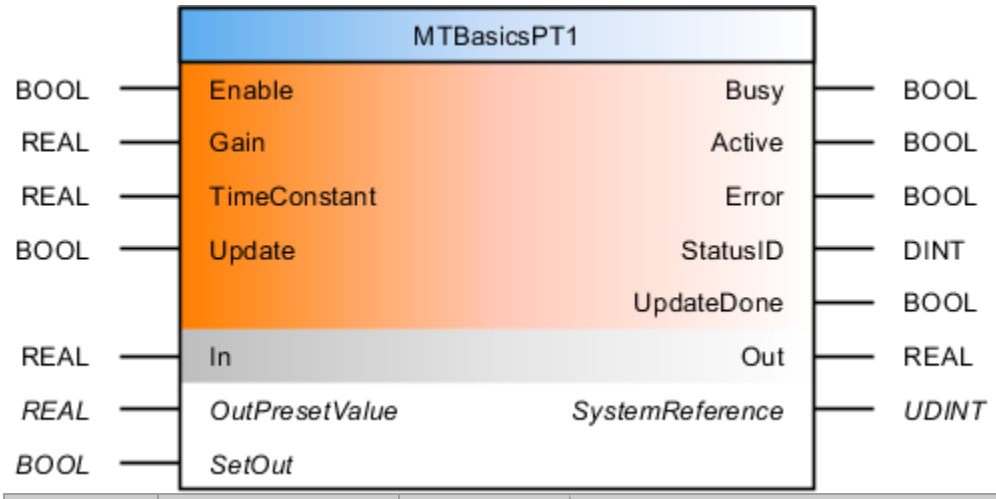
1. 功能块说明



该功能块位于 MTBasics 库中，该模块实现了 PT1 的传递函数，具有增益因子 Gain（K）和时间常数 TimeConstant（T1），可实现一阶惯性延迟低通滤波。

$$G(s) = \frac{K}{1 + T_1 s}$$

2. I/O 参数说明



I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	Gain	REAL	增益因子
IN	TimeConstant	REAL	时间常数，单位：s。允许的取值范围： • TimeConstant ≥ 任务扫描周期时间； • TimeConstant = 0，则禁用该功能块。

IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值。
IN	OutPresetValue	REAL	Out 输出的预设值 。
IN	SetOut	BOOL	在一个上升沿上将 Out 设置为 OutPresetValue。
OUT	Busy	BOOL	该功能块尚未执行完毕。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值。
OUT	SystemReference	UDINT	系统参考。

3. 调用方式

变量定义

```
VAR
  Input : REAL := 0.0; (*input signal*)
  Output : REAL := 0.0; (*output signal*)
  PT1 : MTBasicsPT1; (*function block MTBasicsPT1*)
END_VAR
```

ST 语言

```
PROGRAM _INIT
  PT1.Gain:= 2.0;
  PT1.TimeConstant:= 3.0;
  PT1.Enable := TRUE;
```

```
END_PROGRAM
PROGRAM _CYCLIC
  PT1.In := Input;
  (* call function block *)
  PT1();
  Output := PT1.Out;
END_PROGRAM
```

C 语言

```
#include<bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include<AsDefault.h>
#endif
void _INIT ProgramINIT(void)
{
  PT1.Gain = 2.0;
  PT1.TimeConstant = 3.0;
  PT1.Enable = 1;
}
void _CYCLIC ProgramCYCLIC(void)
{
  PT1.In = Input;
  /* call function block */
  MTBasicsPT1(&PT1);
  Output = PT1.Out;
}
```

1.3. 滑动均值滤波

1. 基本信息

编写人	穆珊珊	审核人	
算法归属	信号处理	应用对象	

其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V0.1	2023.02.07	整理标准化功能块 V4.15 版本原始内容	王宇鑫

2. 算法简介

滑动均值（移动均值）滤波是一种比较简单也容易实现的一种滤波方法，可以实现低通滤波，使输入信号变得平滑。这里我们主要介绍它的两种应用：

（1）反馈信号滤波；

（2）Input Shaping（输入整形）；

3. 内容

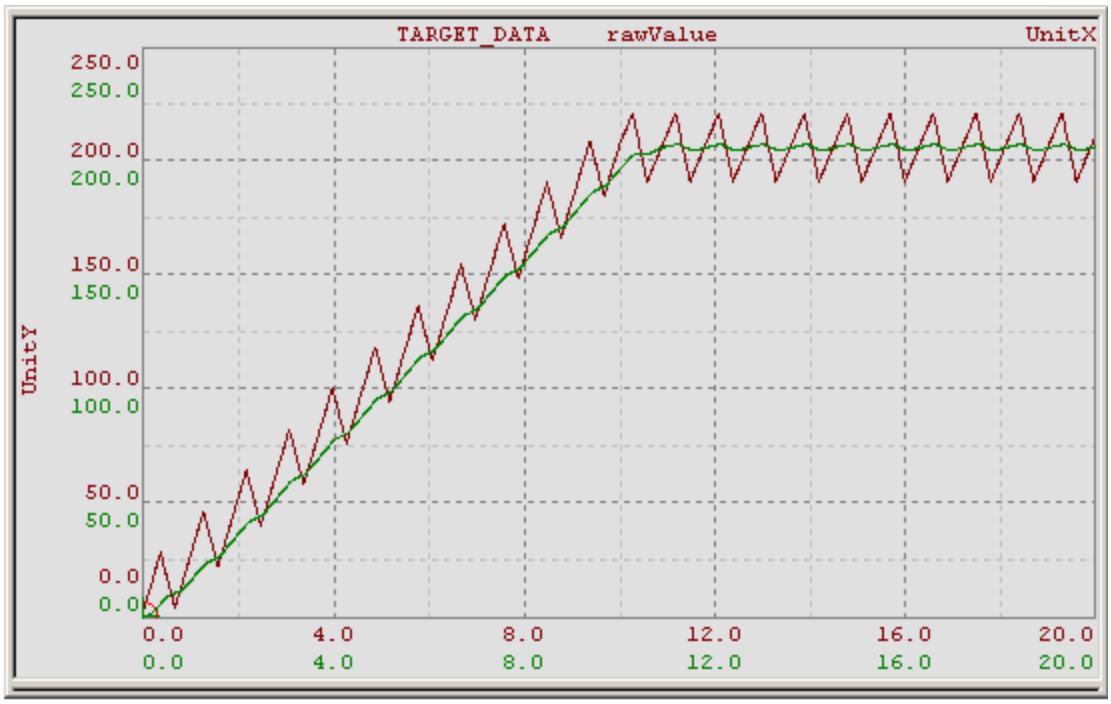
（1）反馈信号滤波。

主要用于传感器反馈信号的滤波。滑动均值滤波，顾名思义，就是对前一段时间内（多次采样）的数据取平均值，做为当前时刻采样值的一种滤波方法。使用这种方法，首先需要确定一个基数 Base，即多长时间的数据，或多少个点，取平均值。计算公式如下：

$$y_0 = \frac{1}{n}(x_0 + x_1 + \cdots + x_{n-1})$$

其中 n 为滤波基数，y 为滤波后的值，x 为未经处理的采样值，下标 0 表示当前时刻，下标 1…n-1 表示前 n-1 个时刻。

滤波可以去除一些采样的干扰，使采样曲线变得平滑，效果可参考下图（取自 AS Help：LCRMovAvgFlt）：

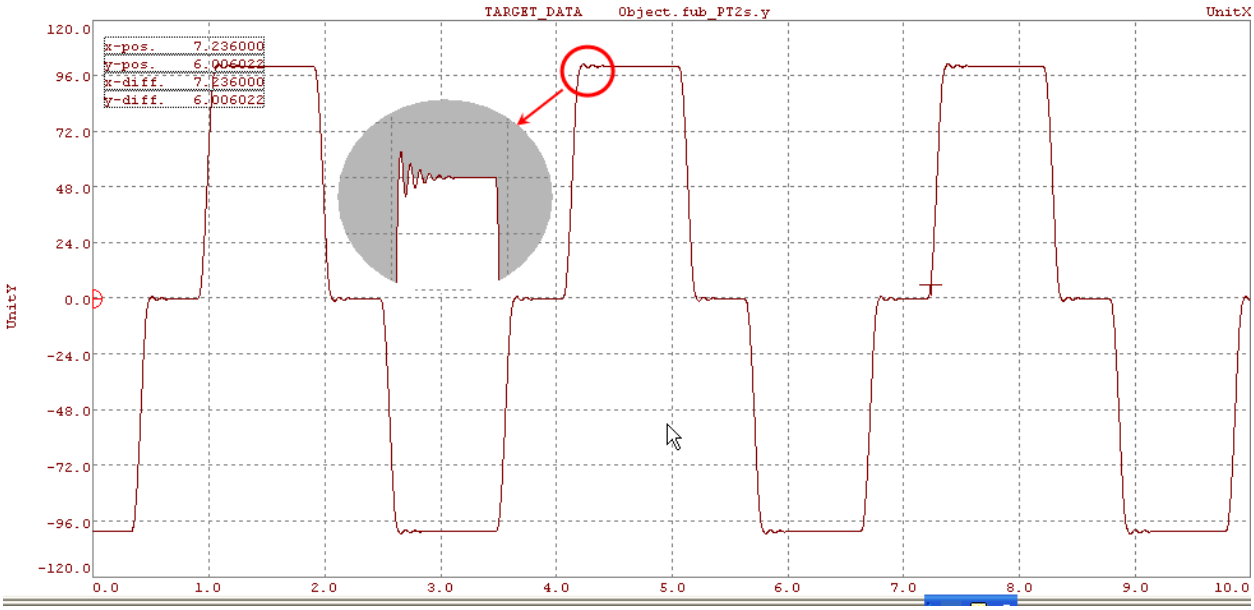


使用这种滤波方法时，很重要的一点就是选取一个合适的 Base 值。Base 如果太大，则滤波会造成比较大的信号滞后，时间延迟为滤波窗口宽度的一半乘以扫描周期；如果太小，则不能达到一个好的滤波效果。

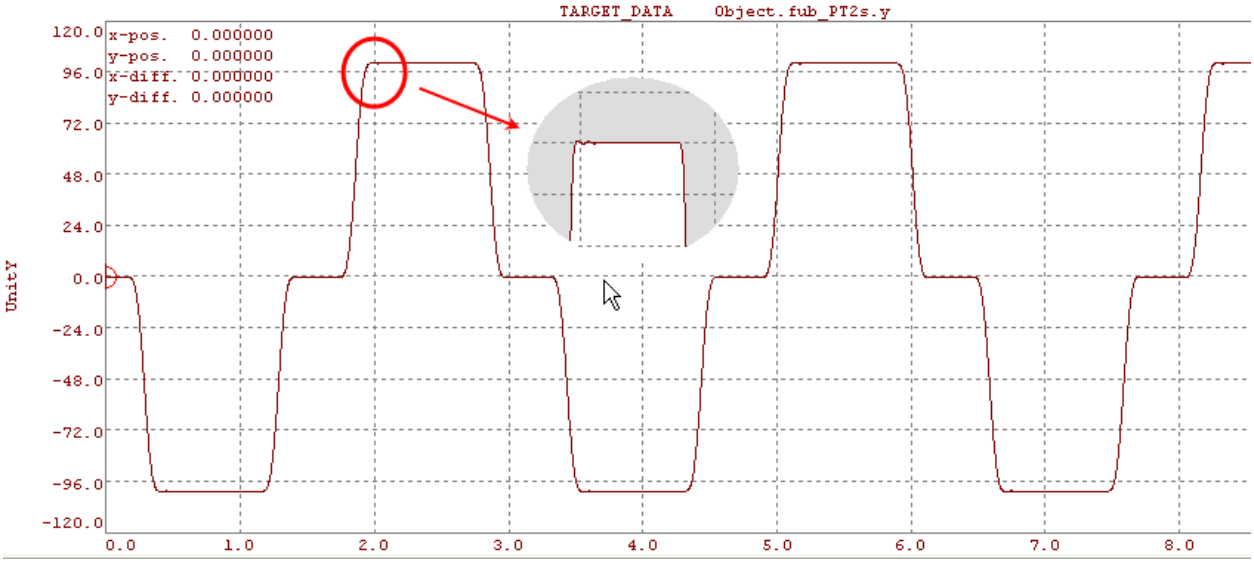
（2）前馈信号滤波（Input Shaping）

滑动均值滤波也可以用来实现 Input Shaping 的功能，将输入信号变的更加平滑。例如在运动控制中，对运行轨迹进行整形处理，消除振荡现象。整形效果可以通过如下例子说明。

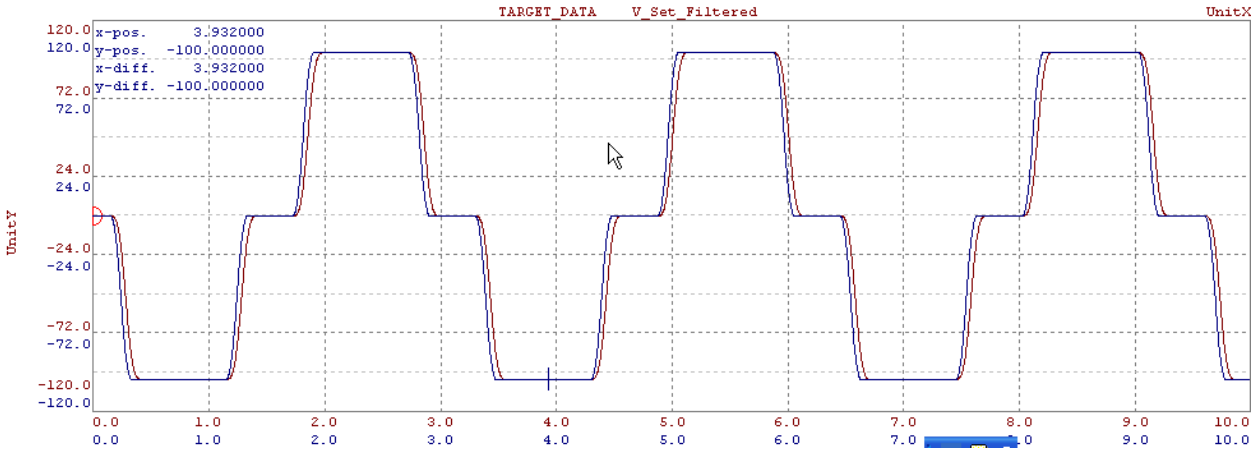
假设存在一个运动控制系统，给系统设定一个往复运动轨迹。由于对象阻尼比较低，在加减速时就会出现微小的振动现象。下图为对象运动的实际速度。



为了消除速度变化时的振动，我们对系统的设定速度进行输入整形。整形后对象的响应（实际速度）中振动基本被消除。



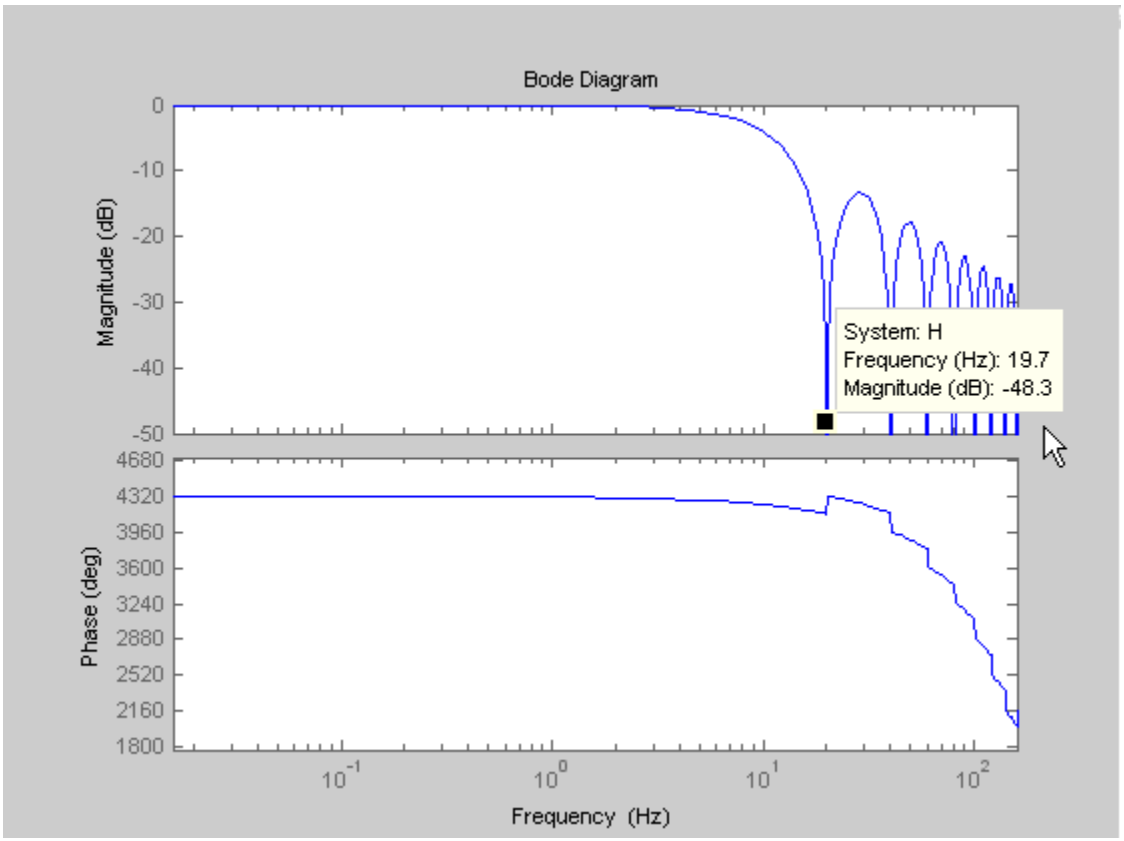
输入整形是对输入设定值的滤波，下图（红色曲线为整形后，蓝色曲线为整形前）给出了整形前后的设定值的对比，可以看出在速度发生变化的地方，整形后比整形前曲线更平滑。



使用滑动均值滤波做输入整形时，关键是 Base 这个参数的选取。Base 的取值和需要消除的振荡的频率有关。为了更好的理解 Base 和振荡频率的关系，我们先研究下滑动均值滤波的传递函数（见 AS Help：LCRMovAvgFlt）：

$$G(z) = \frac{1}{base} \cdot \frac{z^{base} - 1}{z^{base} - z^{base-1}}$$

这是离散传递函数，离散时间为 1ms，在 Base=50 的情况下，其 Bode 图如下：



我们可以得到其截止频率为 20Hz，正好是 1/Base。

由此我们得出 Base 的取值方法：

- 获取想要消除的振荡频率 f(Hz)，Base >= 1/f。

如果是在 AS 中使用 LCRMovAvgFlt 功能块实现整形滤波，则还要考虑任务周期 (T_cyc)：

- Base*T_cyc >= 1/f。

Base 的这一取值方法也适用于信号反馈滤波。

4．程序实现/应用举例

方法 1：通过 LoopConR 中的 LCRMovAvgFlt 功能块实现，见 使用 LCRMovAvgFlt 实现滑动均值滤波；

方法 2：通过 MTFilter 库中的 MTFilterMovingAverage 功能块实现，见 使用 MTFilterMovingAverage 实现滑动均值滤波 。

1.3.1. 使用 LCRMovAvgFlt 实现滑动均值滤波

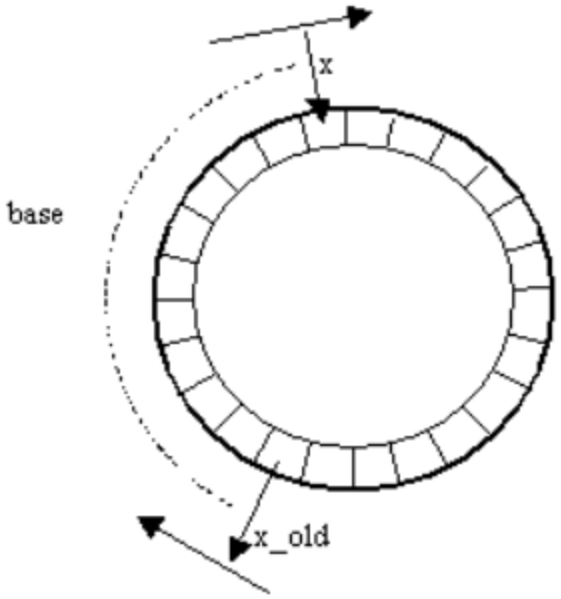
功能块说明

该功能块位于 LoopConR 库中，可实现滑动均值低通滤波。

计算公式如下：

$$y = \frac{sum\_old - x\_old + x}{base}$$

$$sum\_old = sum\_old - x\_old + x$$



该功能块使用 Base 中指定的上一个周期数生成不断更新的输入 x 的平均值，并将其输出到 y 输出端。

功能块启动时的行为（从 FALSE 启用到 TRUE）。

- Base 中的所有值都与第 1 个 x 值一起写入；。
- sum_old = base * x;
- x_old = x;
- 如果 enable=0，则 y=0。

I/O 参数说明

I/O	参数名	数据类型	说明
IN	enable	BOOL	这个功能块只有在 enable = TRUE 时才会执行： • 当 FALSE 变为 TRUE 时，y 被设置为 x； • 当 TRUE 变为 FALSE 时，y 被设置为 0。

IN	x	REAL	输入信号。
IN	base	UINT	用于生成平均值的周期数（最多允许 65535）。 如果需要超过 32 个坐标，那么功能块从 UserRAM 中分配基数*4 字节的内存来缓存中间数据。 理论延迟时间为：0.5*base*dt，dt 为扫描周期。
OUT	status	UINT	功能块错误报警。
OUT	y	REAL	输出信号。

1.3.2. 使用 MTFilterMovingAverage 实现滑动均值滤波

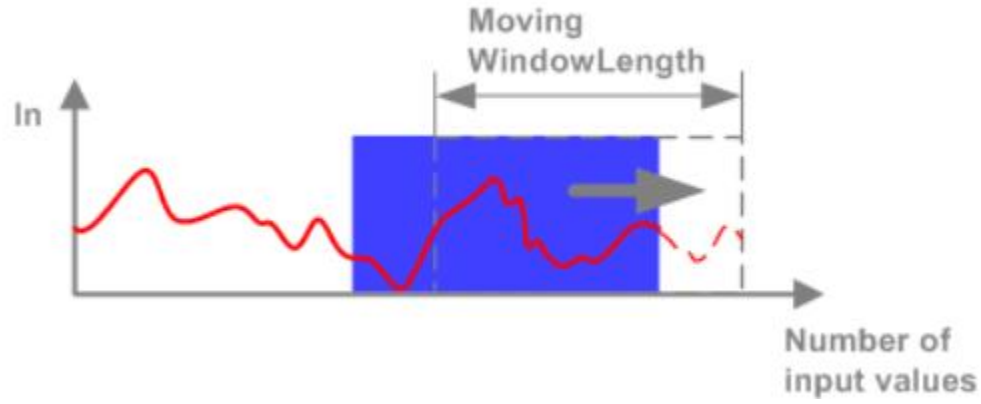
1. 功能块说明

MTFilterMovingAverage 为滑动均值滤波器。

滑动均值滤波器的时域表达式为：

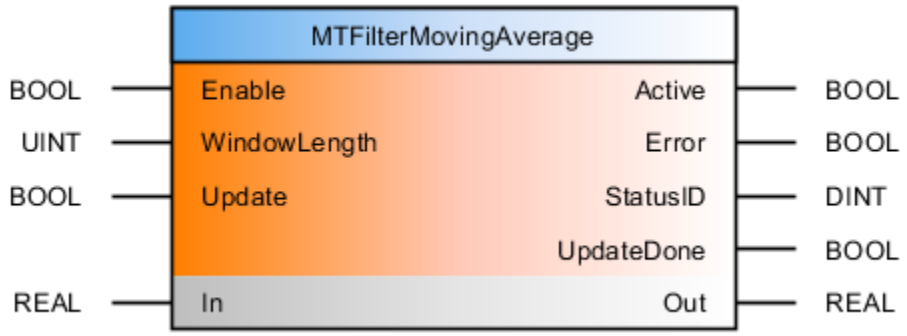
$$Out = \frac{1}{N}\sum_{i=1}^{N} In$$

其中，N 为窗口长度。



输出信号 out 是由输入信号 in 经过滑动滤波得到的，WindowLength 则为滤波窗的宽度，为可配置参数。

2. 功能块及接口说明



I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	WindowLength	UINT	移动窗口的长度，代表要平均的数值的数量，允许的数值范围：0 < WindowLength < 10000。 理论时间延迟：0.5 * WindowLength *dt，dt 为 CPU 扫描周期
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值。

3. 调用方式

变量定义

```
VAR
  Input : REAL := 0.0; (*input signal*)
  Output : REAL := 0.0; (*output signal*)
  MovingAverage : MTFilterMovingAverage; (*function block MTFilterMovingAverage*)
END_VAR
```

ST 语言

```
PROGRAM _INIT
  MovingAverage.WindowLength:= 5;
  MovingAverage.Enable := TRUE;
END_PROGRAM
PROGRAM _CYCLIC
  MovingAverage.In := Input;
  (* call function block *)
  MovingAverage();
  Output := MovingAverage.Out;
END_PROGRAM
PROGRAM _EXIT
  MovingAverage.Enable := FALSE;
  MovingAverage();
END_PROGRAM
```

C 语言

```
#include<bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include<AsDefault.h>
#endif
void _INIT ProgramInit(void)
{
  MovingAverage.WindowLength:= 5;
  MovingAverage.Enable := TRUE;
}
void _CYCLIC ProgramCyclic(void)
{
  MovingAverage.In = Input;
  /* call function block */
  MTFilterMovingAverage(&MovingAverage);
  Output = MovingAverage.Out;
}
void _EXIT ProgramExit(void)
{
```

```
  MovingAverage.Enable = 0;
  MTFilterMovingAverage(&MovingAverage);
}
```

1.4. 基于截止频率的滤波方法

基本信息

本部分将介绍一些基于频率设置的一些滤波方法：

- 贝塞尔滤波
- 巴特沃斯滤波
- Notch 滤波
- BiQuad 滤波

以及贝加莱响应的功能块：

- MTFilter 库中的基于截止频率滤波功能块

1.4.1. 贝塞尔滤波

1. 基本信息

编写人	董俊清	**审核人**	
算法归属	信号处理	**应用对象**	
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V0.1	2023.02.07	整理标准化功能块 V4.15 版本原始内容	王宇鑫

2. 算法简介

贝塞尔（Bessel）滤波器时具有最大平坦的群延迟（线性相位响应）的线性滤波器。贝赛尔滤波器常用在音频系统中，因为在音频设备中，必须在不损害频带内多信号的相位关系前提下，消除带外噪声。相对其他滤波器，贝赛尔滤波器的恒定群延迟特点能最好的满足这个要求。贝赛尔滤波器在生物医学信号放大与处理过程中也得到了广泛应用。

3. 内容

滤波器设计

（1）低通滤波器

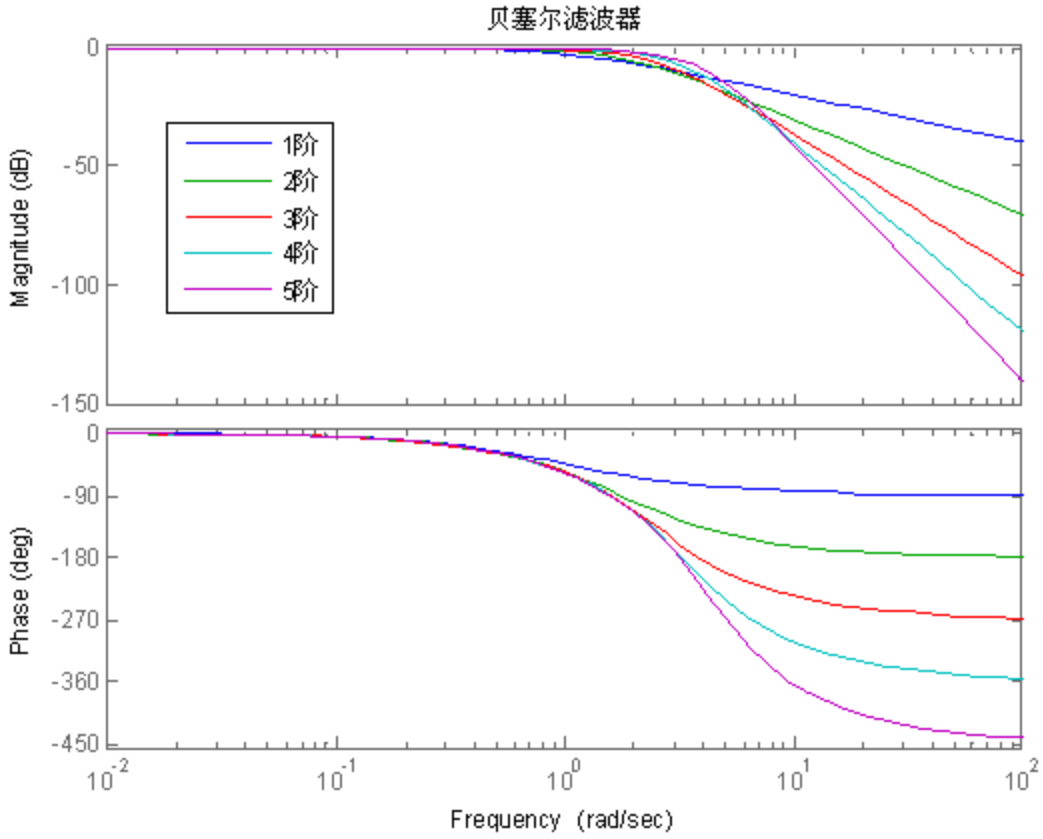
贝塞尔低通滤波器是一种全极点滤波器，其传递函数形式为：

$H(s)=\frac{1}{B_N(s)}$，其中$B_N(s)$称为贝塞尔多项式。$B_N(s)=\sum_{k=0}^{N}a^k s^k$。

$$a^k=\frac{(2N-k)!}{2^{N-k}k!(N-k)!}$$

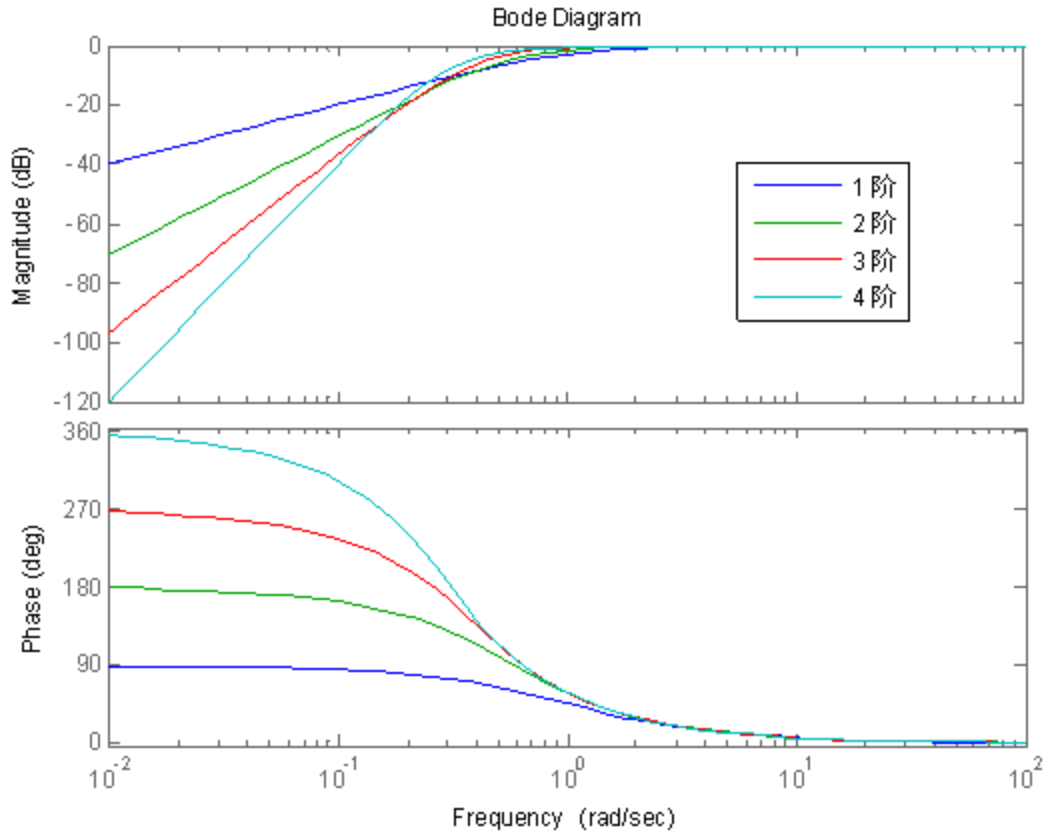
$B_N(s)$还可以用以下递推方式获得：$B_N(s)=(2N-1)B_{N-1}(s)+s^2B_{N-2}(s)$。

$B_0(s)=1$，$B_1(s)=s+1$。



（2）高通滤波器

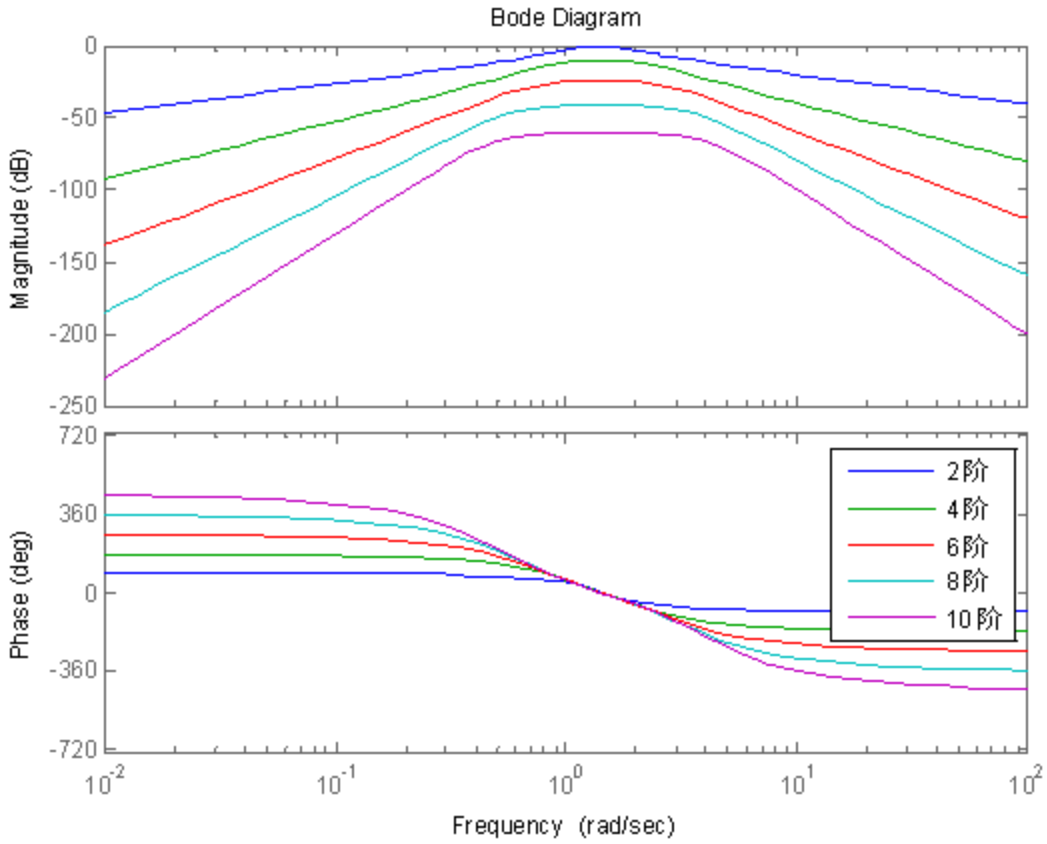
在得到的贝塞尔低通滤波器的基础上，用 $s = \frac{\omega_{HP}\omega_{LP}}{s}$ 进行替代，就可以得到贝塞尔高通滤波器。其中 ω_{LP} 是低通滤波器的截止频率， ω_{HP} 是高通滤波器的截止频率。



（3）带通滤波器

$$s = \omega_{LP}\frac{s^2 + \omega_l\omega_u}{s(\omega_u - \omega_l)}$$

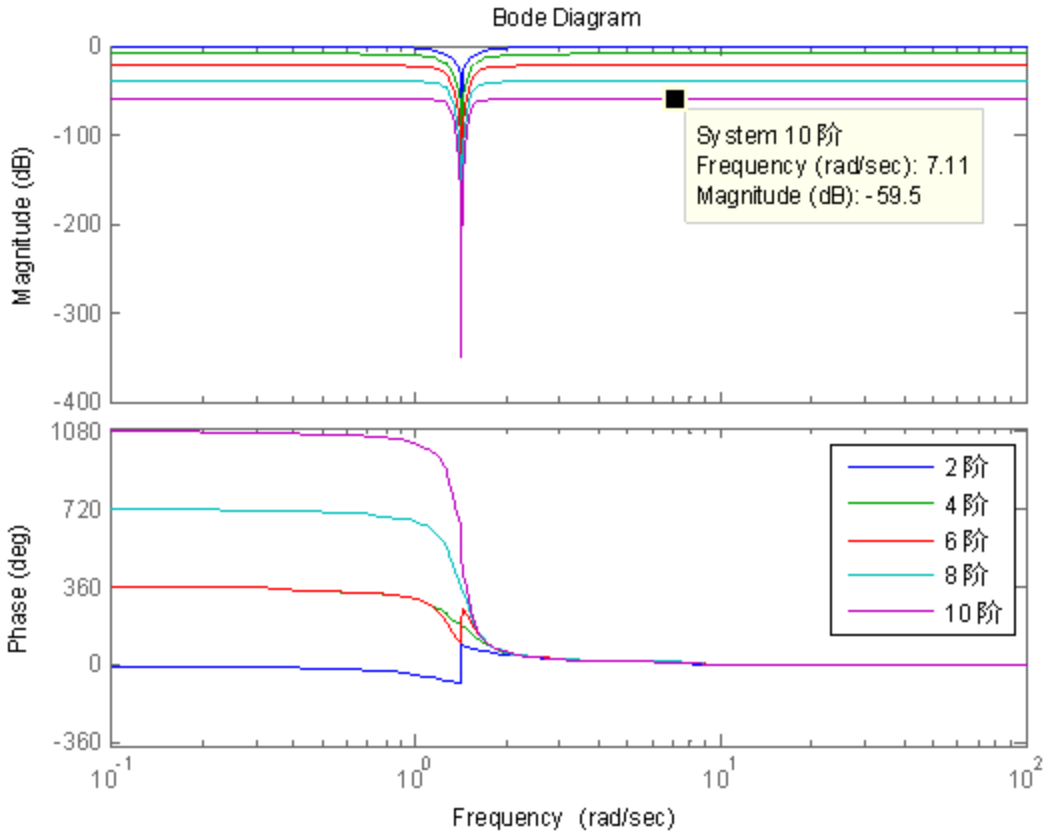
在得到的贝塞尔低通滤波器的基础上，用 进行替代，就可以得到贝塞尔带通滤波器。其中 ω_{LP} 是低通滤波器的截止频率， ω_l 是带通滤波器的下限截止频率，ω_u 是带通滤波器的上限截止频率。



（4）带阻滤波器

$$s = \omega_{LP}\frac{s(\omega_u - \omega_l)}{s^2 + \omega_l\omega_u}$$

在得到的贝塞尔低通滤波器的基础上，用 进行替代，就可以得到贝塞尔带阻滤波器。其中 ω_{LP} 是低通滤波器的截止频率， ω_l 是带阻滤波器的下限截止频率，ω_u 是带阻滤波器的上限截止频率。

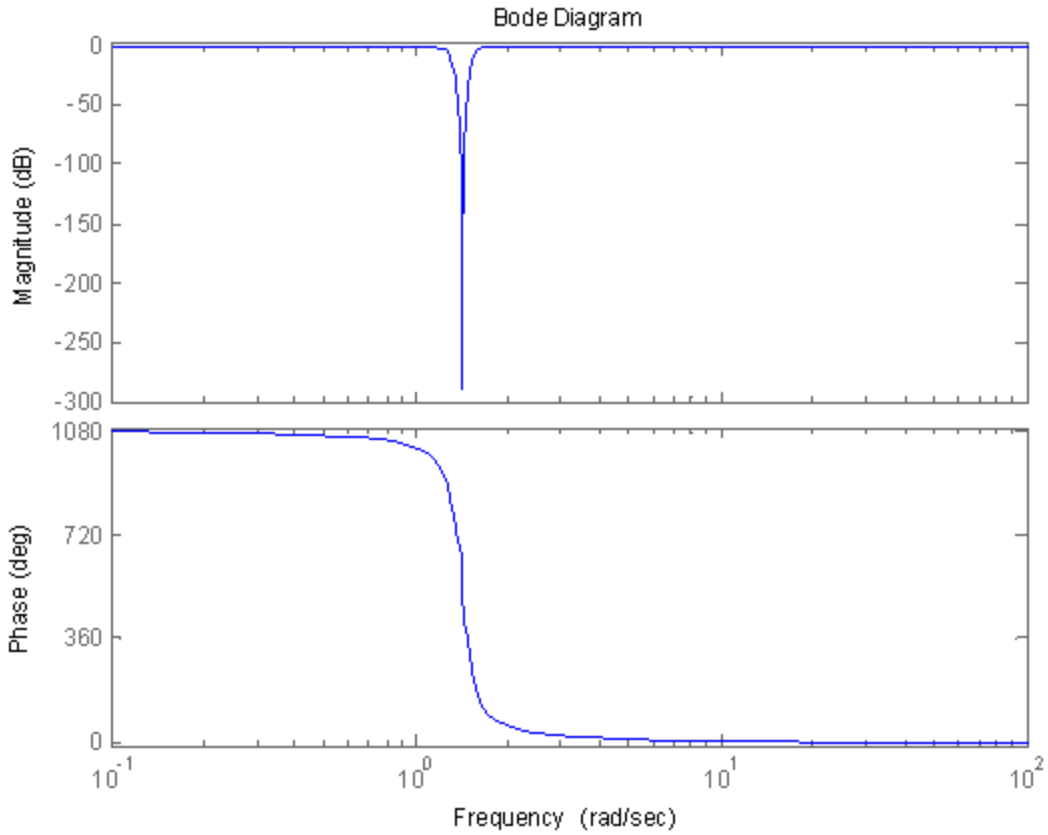


说明：

（1）本文档描述的是贝塞尔低通滤波器的设计，以及通过频率变换得到对应的贝塞尔高通、带通、带阻滤波器的方法。文档描述了贝塞尔滤波器的简单原理，以及用 m 文件实现滤波器的方法。如果要在 AS 中实现，则只需要对对应的 m 文件中的传递函数进行离散化，得到离散传递函数各幂次系数，就可以程序实现。具体可以参考“巴特沃斯数字低通滤波器设计”文中介绍的步骤。

（2）本文档中频率变换是针对贝塞尔滤波器。但是对于巴特沃斯等滤波器，该频率变换仍然可行，并且频率变换公式相同。

（3）在带通带阻滤波器传递函数中，暂时没有找到使滤波器增益为 1 的统一公式，如果实际需要，可以根据传递函数的通带增益，算出需要的增益系数，放到分子上即可。如在带阻滤波器中，10 阶传递函数在通带增益为-59.5dB。所以需要的分子增益为 59.5dB。折算到绝对坐标下为：10^(59.5/20)= 944.0609。即增益系数为 944.0609。加入增益系数后的传递函数 bode 图如下：



可见增益变为单位增益了。

4. 程序实现/应用举例

Matlab 程序（.m 文件）： Bessel.m

1.4.2. 巴特沃斯滤波

1. 基本信息

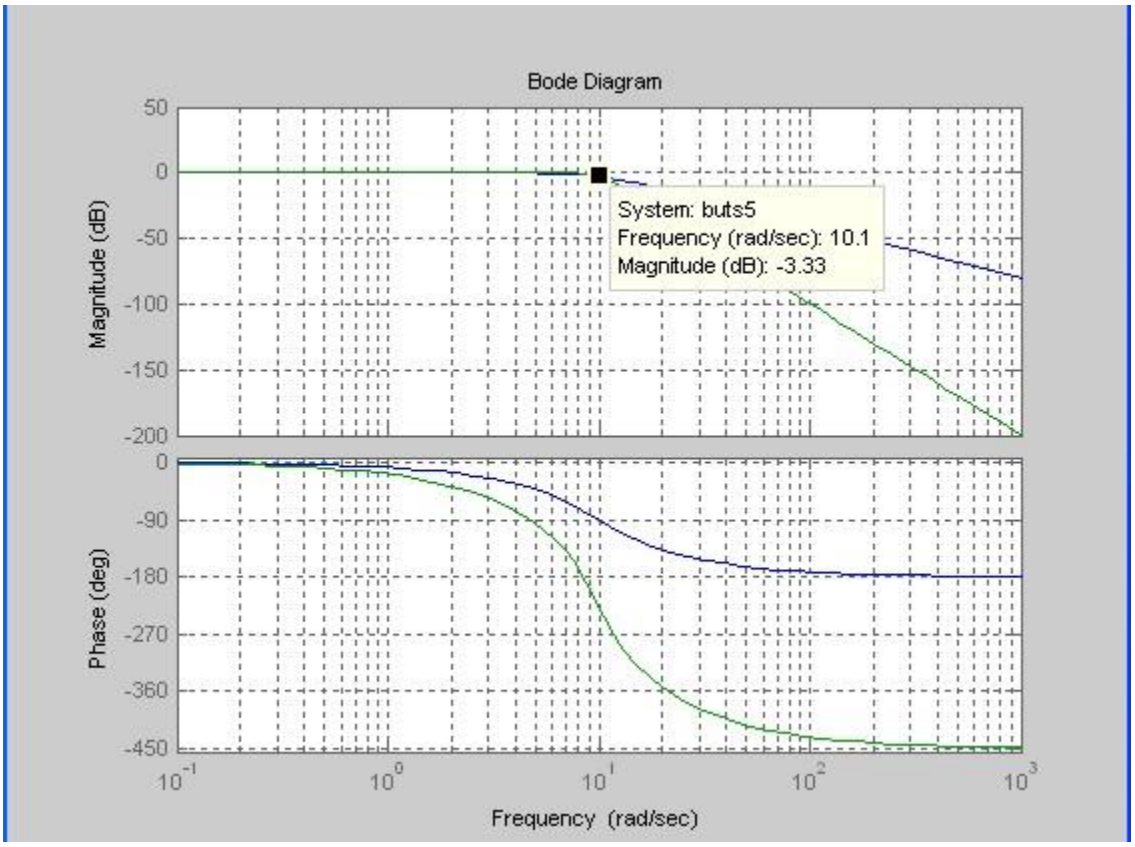
编写人	汪其锐	**审核人**	穆珊珊
算法归属	信号处理	**应用对象**	传感器
其他说明			

版本信息	修改日期	修改内容	修改人
V0.0		详细介绍了实现方法	刘柏严
V0.1	2023.02.07	整理标准化功能块 V4.15 版本原始内容	王宇鑫

2. 算法简介

巴特沃斯滤波器被称为“最平响应“滤波器，由其 Bode 图可以得出，在截止频率附近其幅频特性都是平直的。当阶数越大时，其高频信号衰减越快，但其响应滞后也就越大。

若截止频率为 10rad/s，滤波器阶数为 2，5 阶，其 Bode 图为(蓝色 2 阶 绿色 5 阶):



3. 内容

3.1 时域传递函数

巴特沃斯滤波器的时域传递函数如下：

$$H(s)=\frac{\Omega_C^N}{s^N+a_1\Omega_C s^{N-1}+a_2\Omega_C^2 s^{N-2}+\cdots+a_{N-1}\Omega_C^{N-1}s+\Omega_C^N}$$

N：滤波器阶数

Ω_C：截止频率**（单位 rad/s 注意 fft：hz）**，上述 Bode 图中-3db 点。

其各阶分母系数如下：

	a1	a2	a3	a4	a5	a6	a7	a8	a9
1	0								
2	1.414	0							
3	2	2	0						
4	2.6131	3.4142	2.6131	0					
5	3.2361	5.2361	5.2361	3.2361	0				
6	3.8637	7.4641	9.1416	7.4641	3.8637	0			
7	4.494	10.09678	14.5918	14.5918	10.0978	4.494	0		
8	5.1258	13.1371	21.8462	25.6884	21.8462	13.1371	5.1258	0	
9	5.7588	16.5817	31.1634	41.9864	41.9864	31.1634	16.5817	5.7588	0
10	6.3925	20.4317	42.8021	64.8824	74.2334	64.8824	42.8021	20.4317	6.3925

若需要更高阶的系数，可用下式在 MATLAB 中计算：

[b,a]=butter(n,1,'s'); %n 为阶数 b 为 h(s)分母系数

3.2 滤波器设计

上述巴特沃斯滤波器的传递函数是连续传函，在实际使用时，却需要使用离散形式。为此，下面介绍几种离散巴特沃斯滤波器的设计方法。在此之前，先来介绍下采样时间的设定。

为保证信号采样后能不失真的复原，根据 Nyquist 采样定律可知：

采样频率 fs>=2*fmax(信号最大频率)

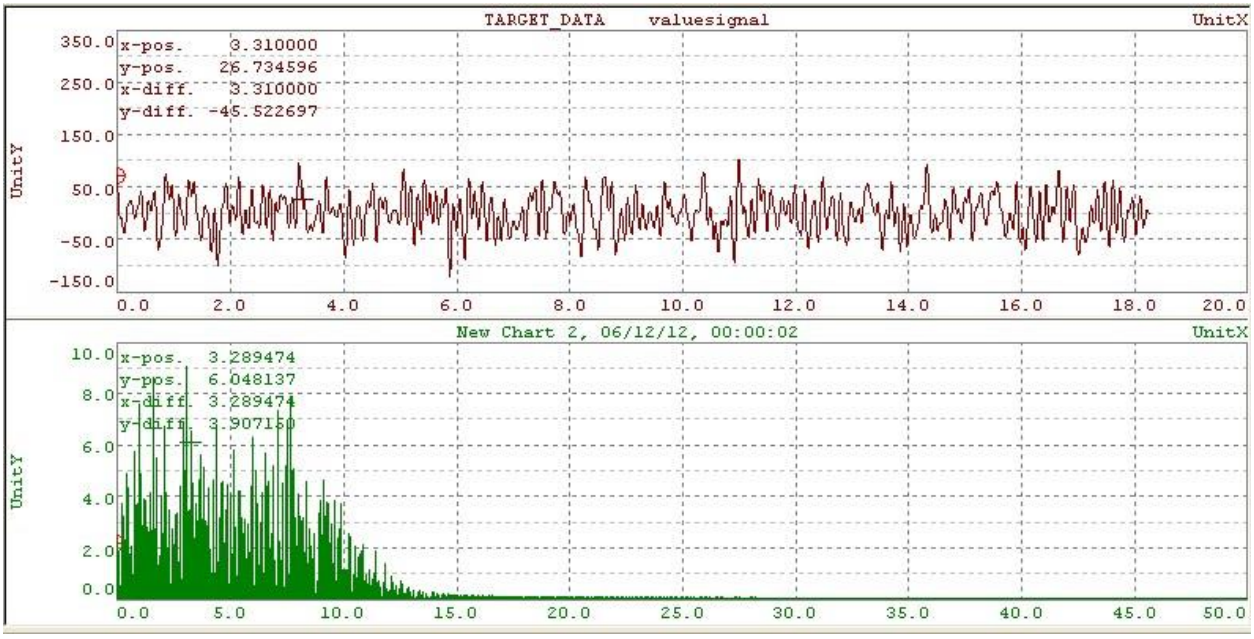
若采用白噪声信号作为输入，假设需要的截止频率为 10Hz，则采样频率至少为 20Hz(实际一般至少选 3 倍 Wc)，所以可设定采样时间为 10ms。

方法 1：由 matlab 函数直接得出 h(z)

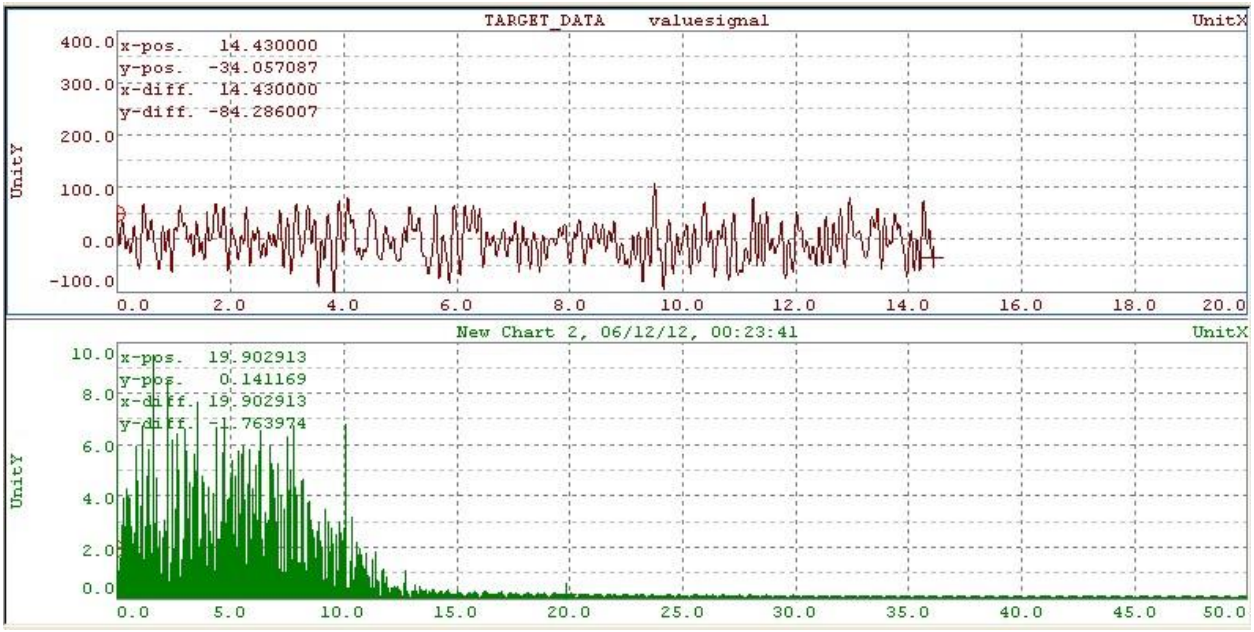
[b,a]=butter(n,2*wc*t,'low') % 0<2*wc*t<1 b 为 h(z)分子系数 a 为分母系数 wc：截止频率；t：采样时间。

注：matlab 原公式为[b,a]=butter(n,wn,'low') 这里面的 0<wn<1（wn 是归一化的截止频率，即 1 表示 1/2*fs）！由于采样频率 fs>=2*fmax(信号最大频率), 即 1/t>2*wc, 所以 0<2*wc*t<1。

由上式得出的分子分母系数虽然与方法一相同，但方法二得出的小数位数固定，所以不容易发散，可设计更高阶滤波器!



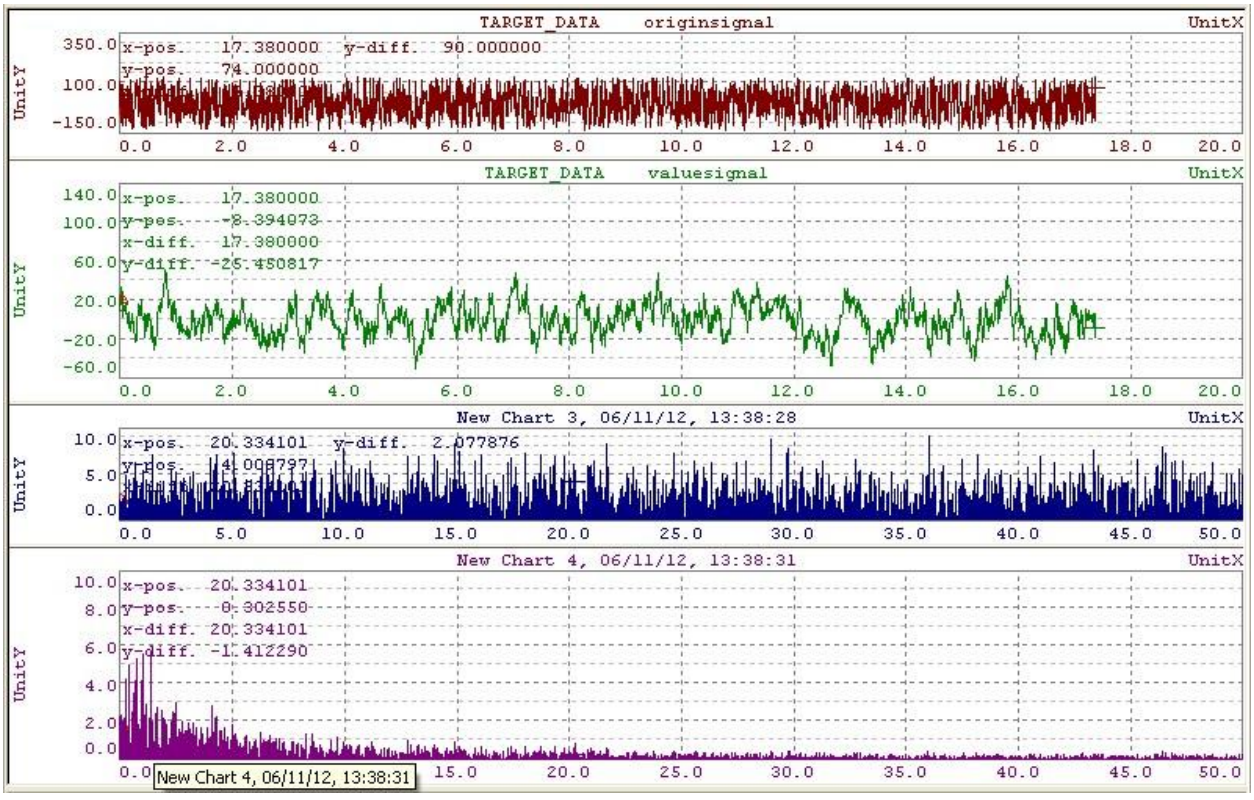
巴特沃斯 9 阶滤波器滤波图像



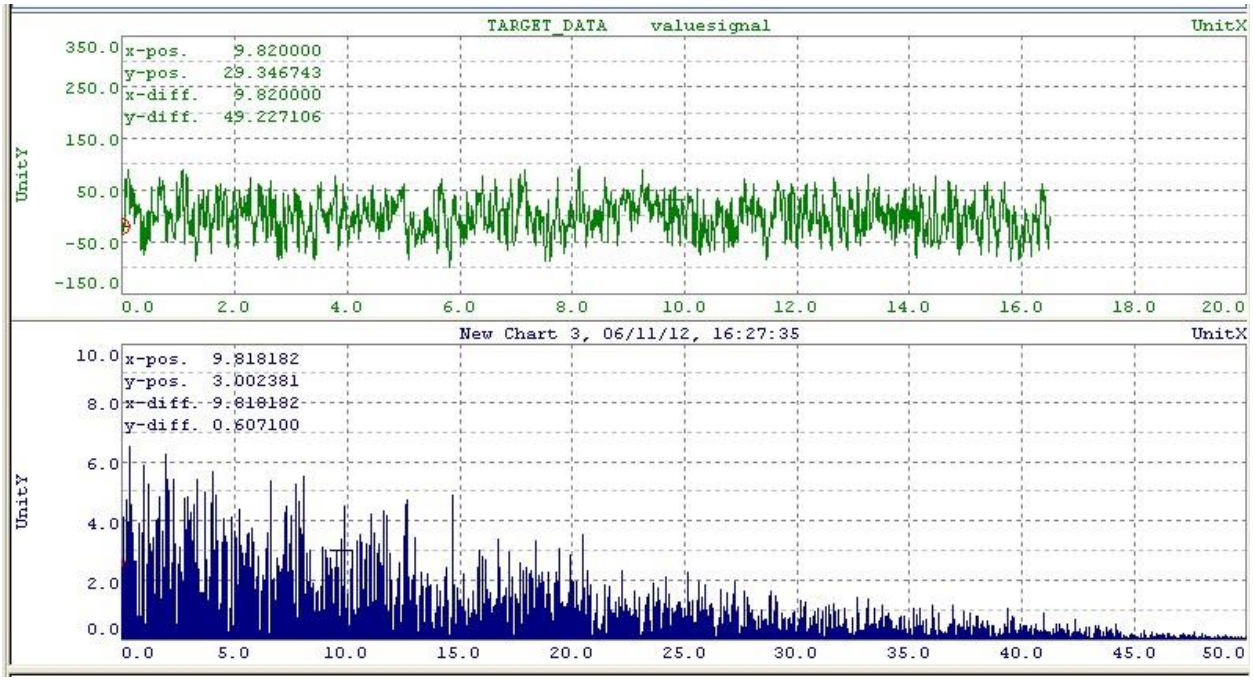
巴特沃斯 10 阶滤波器滤波图像

方法 2：先设计模拟滤波器然后转化为数字滤波器

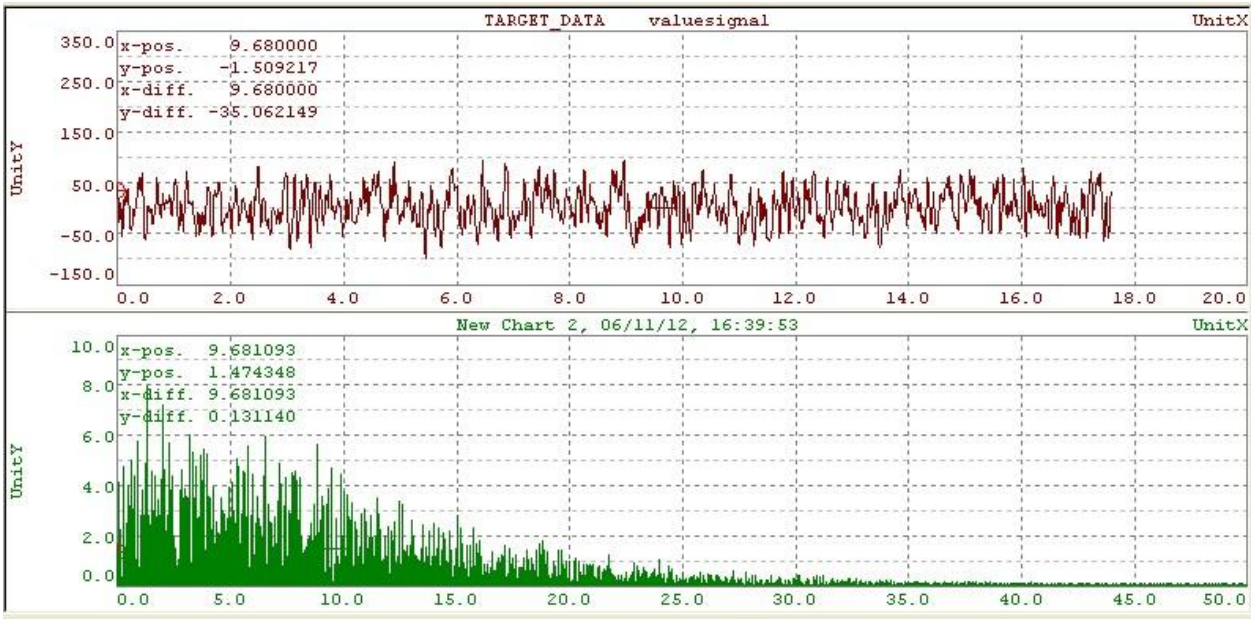
先写出传递函数 H(s)然后利用“双极性变换法”得出 H(z)。此种方法需要由数字指标转化为模拟指标。



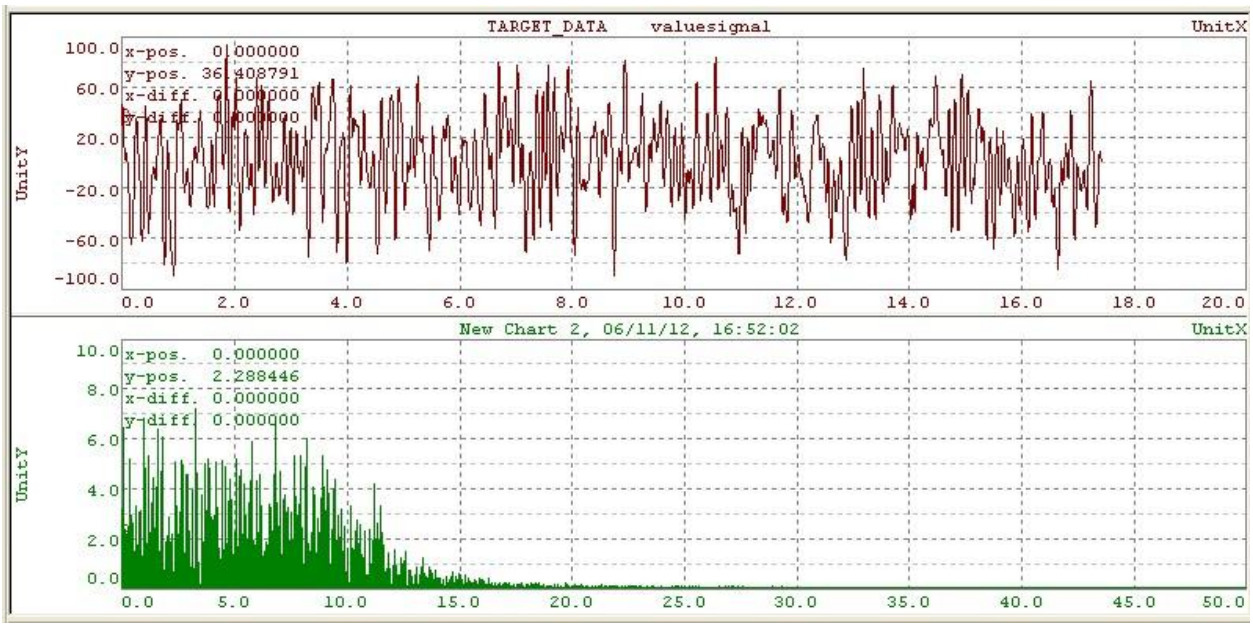
贝加莱 LCRPT1 滤波图像(1.白噪声 2.输出 3.白噪声的 FFT（快速傅里叶变换）4.输出 FFT)



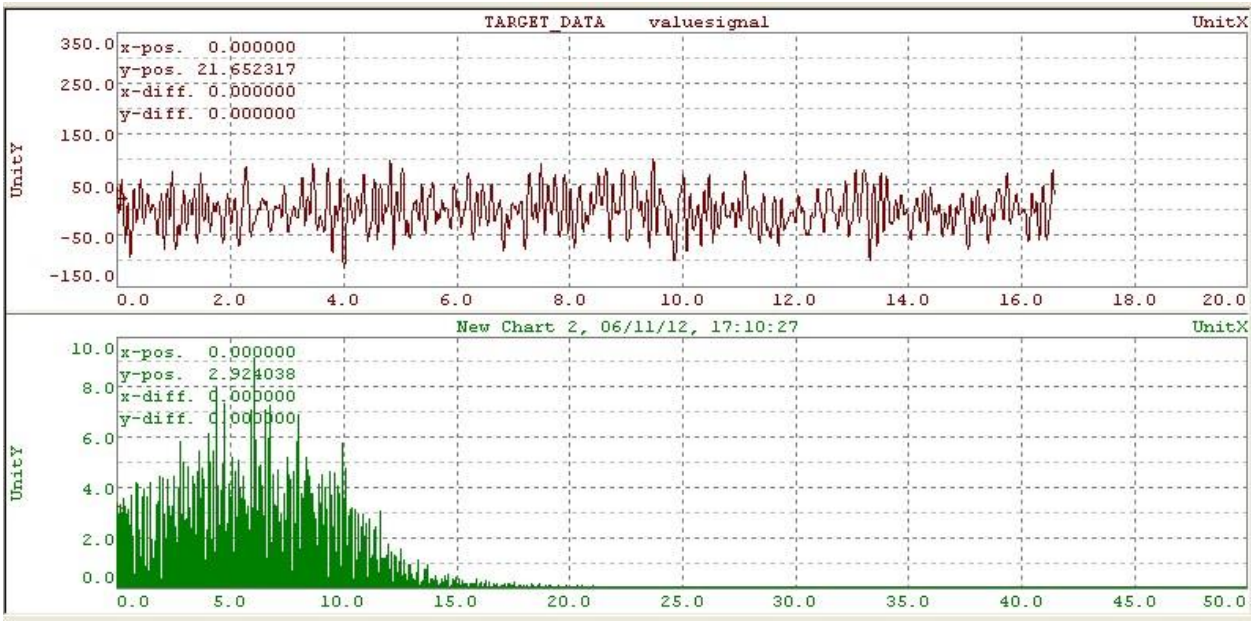
巴特沃斯 1 阶滤波器滤波图像



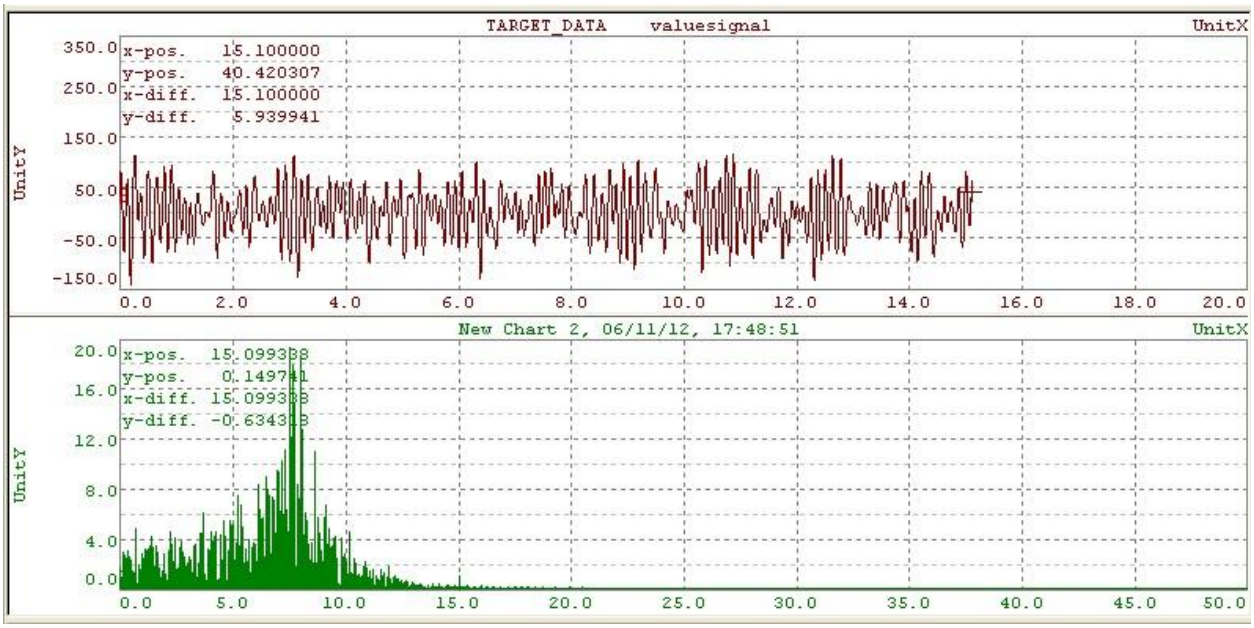
巴特沃斯 2 阶滤波器滤波图像



巴特沃斯 5 阶滤波器滤波图像



巴特沃斯 6 阶滤波器滤波图像

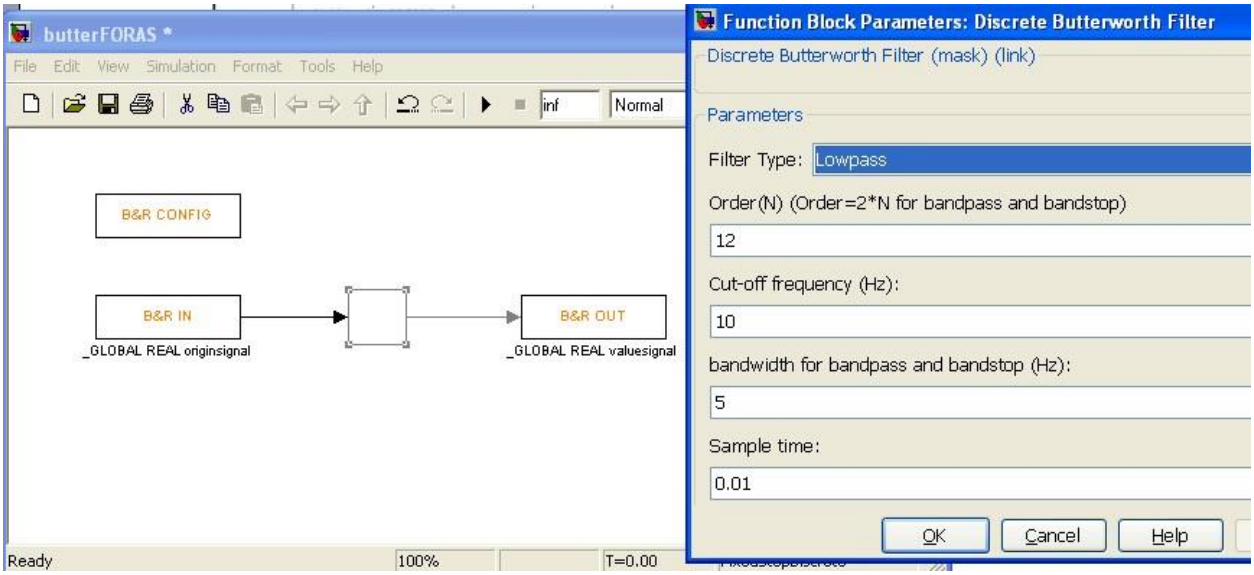


巴特沃斯 8 阶滤波器滤波图像

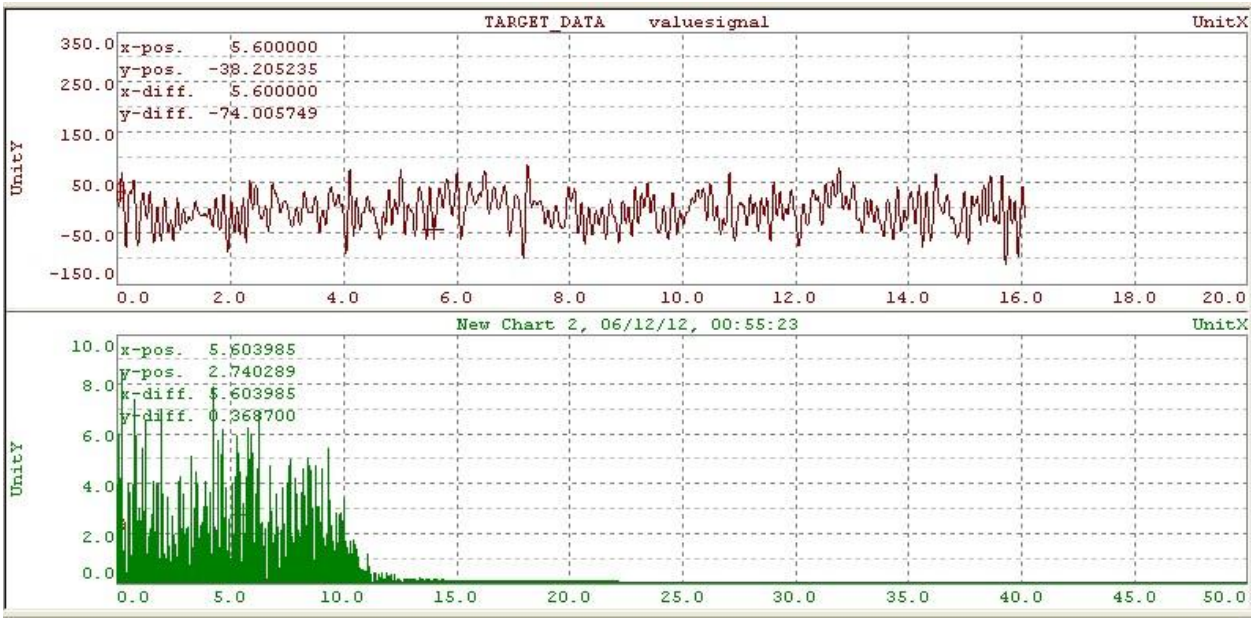
当 n>=9 时，因为系数精度问题，输出会发散。

由上面图像可看出，低阶巴特沃斯滤波器效果不是很理想，但高阶巴特沃斯滤波器明显比 LCRPT1 具有更好的滤波效果。

方法 3：AS 自动生成代码



巴特沃斯滤波器模型



巴特沃斯 12 阶滤波器滤波图像

4．程序实现/应用举例

方法 1：使用 matlab 计算离散传递函数系数

[b,a]=butter(n,2*wc*t,'low') % 0<wn<1

其中，b 为 h(z)分子系，a 为分母系数，n 为阶数，wc 为截止频率，单位 hz，t 为采样时间，单位秒，low 表示设计低通滤波。

离散传递函数表达式如下：

$$h(z) = y/x = \frac{(b_0 * Z^n + b_1 * Z^{n-1} + b_2 * Z^{n-2} + \cdots + b_n)}{(a_0 * Z^n + a_1 * Z^{n-1} + a_2 * Z^{n-2} + \cdots + a_n)}$$

转化为输出离散公式：

$$a_0 y = (b_0 * x + b_1 * x * Z^{-1} + b_2 * x * Z^{-2} + \cdots) - (a_1 * y * Z^{-1} + a_2 * y * Z^{-2})$$

其中Z^{-1}：表示上一次采样。

为了防止小数精度不够造成不稳定，建议不要超过 9 阶。

高阶滤波可能会造成发散（如果极点不再单位圆内），所以计算好系数后应该测试一下。

使用举例：

设计 butterworth 5 阶滤波器，采样时间 0.001s，截止频率 200hz。

观察阶跃响应，观察输入白噪声经过滤波后的 fft 分析。

首先在 matlab 中计算参数，在 matlab 命令窗口中输入以下命令：

n=5

wc=200

t = 0.001

[b,a]=butter(n,2*wc*t,'low') %这里计算 butterworth 分子分母系数，‘low’表示低通，‘high’表示高通，‘stop’表示带阻。

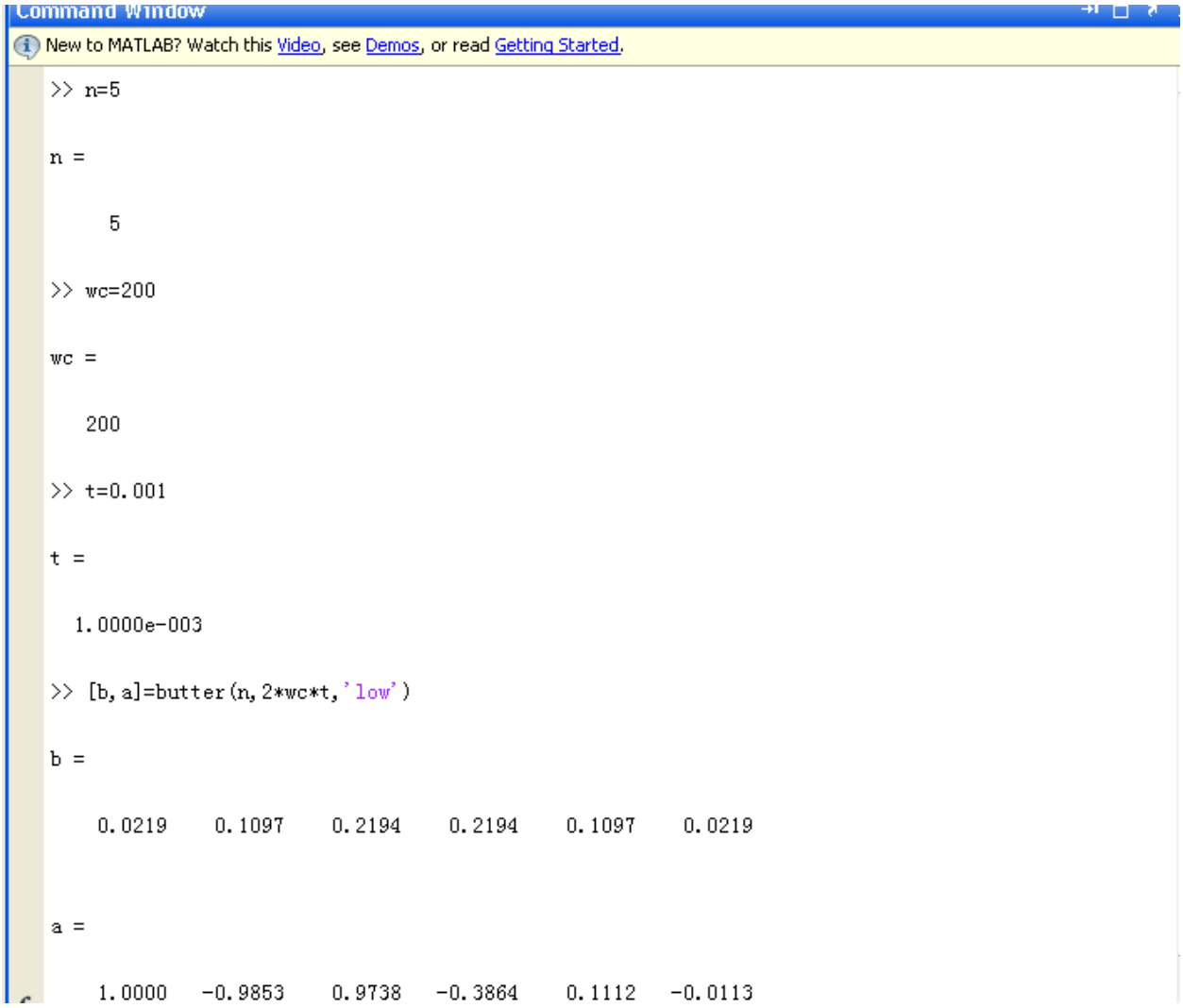
结果如下：

b =

0.0219 0.1097 0.2194 0.2194 0.1097 0.0219

a =

1.0000 -0.9853 0.9738 -0.3864 0.1112 -0.0113



MATLAB 中的命令实现

然后使用上述得到的系数，在 AS 中实现 butterworth 滤波代码，如下：

/*a b init*/

b[0] = 0.0219;

b[1] = 0.1097;

b[2] = 0.2194;

b[3] = 0.2194;

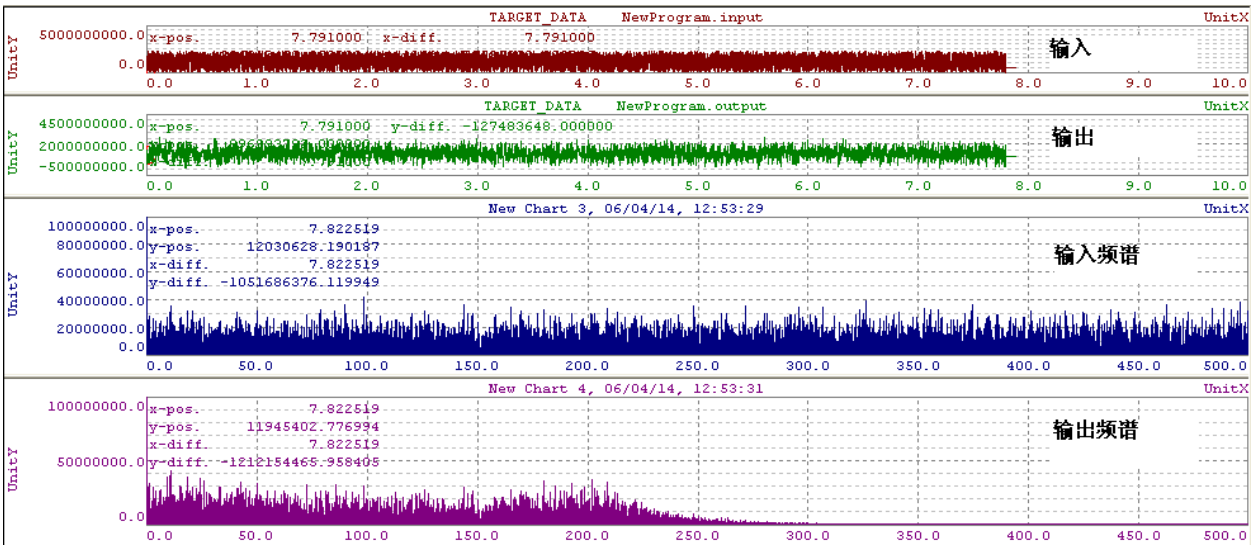
b[4] = 0.1097;

b[5] = 0.0219;

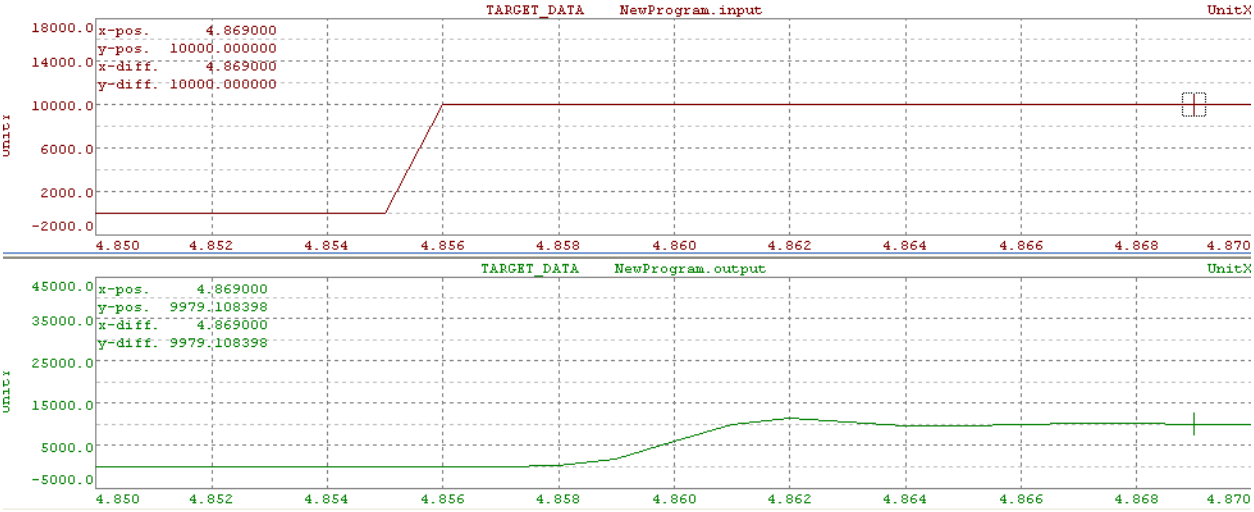
a[0] = 1;

```
a[1] = -0.985;
a[2] = 0.9738;
a[3] = -0.3864;
a[4] = 0.1112;
a[5] = -0.0113;
/* xn,xn-1,...yn,yn-1... */
x[5] = x[4];
x[4] = x[3];
x[3] = x[2];
x[2] = x[1];
x[1] = x[0];
x[0] = input;
/*white noise*/
y[5] = y[4];
y[4] = y[3];
y[3] = y[2];
y[2] = y[1];
y[1] = y[0];
y[0] = (b[0]*x[0]+b[1]*x[1]+b[2]*x[2]+b[3]*x[3]+b[4]*x[4]+b[5]) -
(a[1]*y[1]+a[2]*y[2]+a[3]*y[3]+a[4]*y[4]+a[5]*y[5])/a[0];
output = y[0];
input = rand();
```

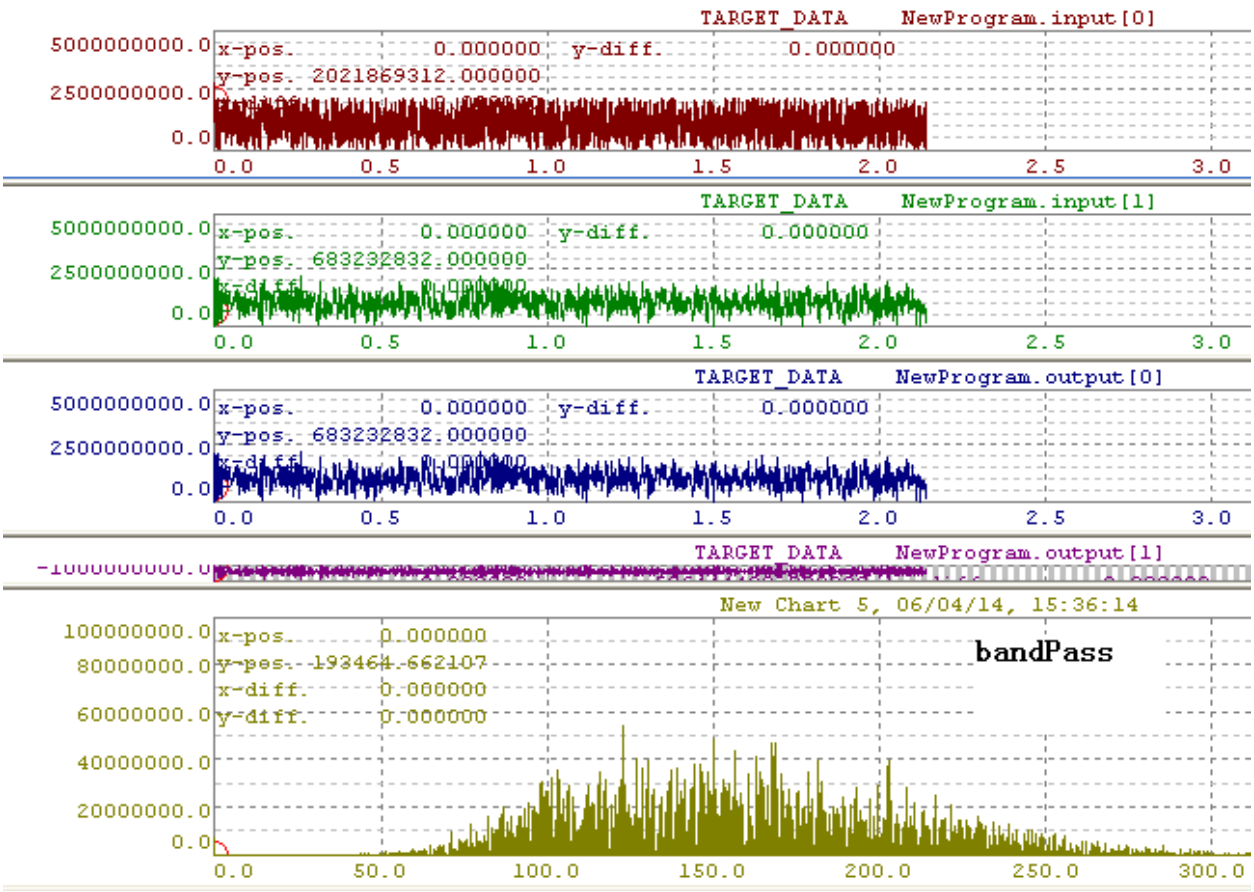
使用 Trace 功能获取输入输出信号，并观察其频谱特性，如下：



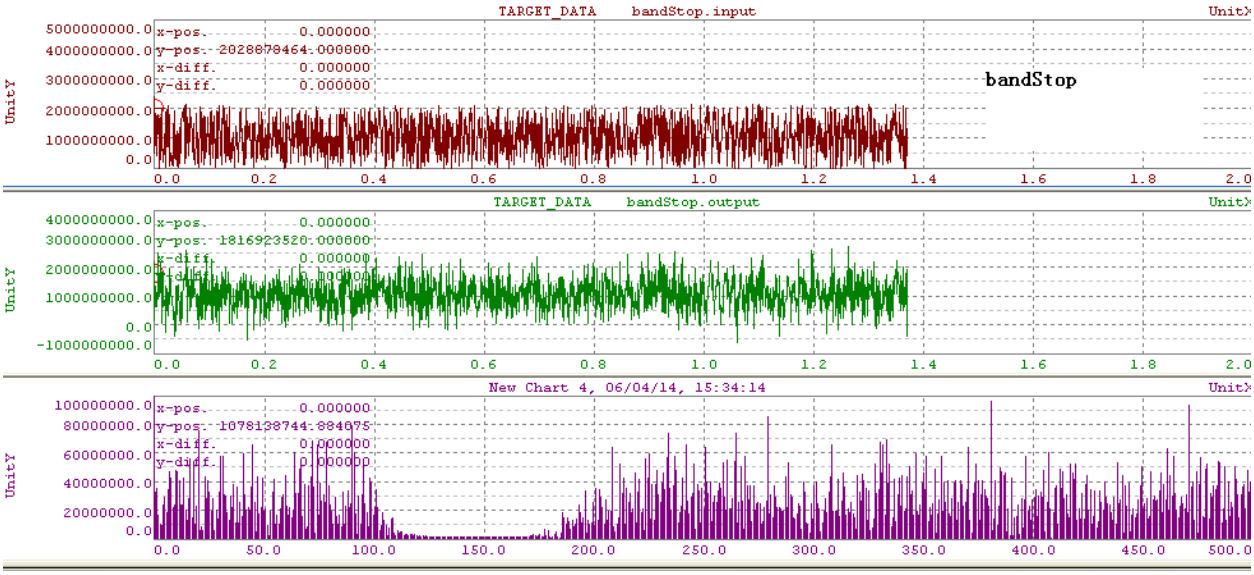
Butterworth 滤波器对于阶跃信号的响应如下，可以看到滤波后的信号有明显的超调：



带通滤波器相当于一个截止频率较高的低通滤波器和一个截止频率较低的高通滤波器串联。原理一样，过程不再介绍，Trace 到的结果如下图所示：



带阻滤波器相当于一个截止频率较低的低通滤波器和一个截止频率较高的高通滤波器串联。可以直接由 matlab 产生系数，但是阶数翻倍。Trace 到的结果如下图：



方法 2：使用 matlab 计算连续域系数，然后离散化，得到离散系数。

调用函数

format long %长精度

[b,a]=butter(5,1256,'s') %调用函数 计算截止频率 200hz 其中 1256= 200*2*pi 单位 rad/s

gs=TF(b,a) %得到连续传递函数

gz=c2d(gs,0.001,'trustin')%得到离散传递函数

bode(gs,gz) %比较连续和离散 bode 图

step(gz) %观察阶越响应

如果使用 besel 滤波那么第一行更换为[b,a] = besself(n,Wo)其中 Wo 单位 rad/s

需要注意 besel 滤波只支持先计算连续域然后离散化。

与总部 MT 库功能比较系数计算略有不同,总部库函数,使用方法 2 进行计算,但是在我们使用的时候建议使用第一种方法,因为第二种方法 matlab 不能得到很多小数点后面的数据，所以可能造成极点超过单位圆，从而发生不稳定现象。所以如果用程序，建议使用第一种方法。

为了使 matlat 有更长精度，可以在命令窗口输入下面命令，并得出 a、b 系数:

Format long %长精度

[b,a] = butter(5,1256,'s') %1256= 200*2*pi 计算结果与 MTFilterLowPass.internal.num,den 完全一致。

然后进行离散化。

gs=TF(b,a)

gz=c2d(gs,0.001) 或者 gz=c2d(gs,0.001,'zoh') 或者 gz=c2d(gs,0.001,'trustin')

不知道总部离散化方法是什么，以上几种 matlab 方法都尝试了，trustin 方式最接近。但是此方法阶数高的时候容易造成不稳定。建议使用 zoh 方法。

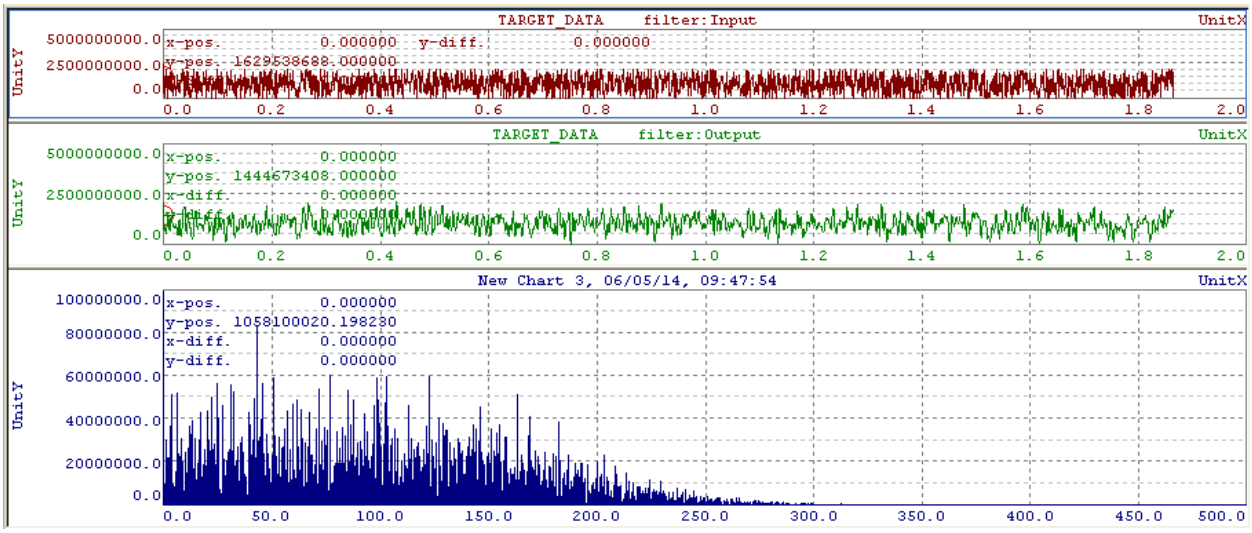
使用 matlab 计算系数

```
>> gz=c2d(gs,0.001,'trustin')

Transfer function:
0.01399 z^5 + 0.06993 z^4 + 0.1399 z^3 + 0.1399 z^2 + 0.06993 z + 0.01399
-------------------------------------------------------------------------
      z^5 - 1.409 z^4 + 1.332 z^3 - 0.6322 z^2 + 0.1775 z - 0.0202

Sampling time: 0.001
```

MTFilterLowPass 采样频率 1000，截止频率 200，输入白噪声，输出 fft 分析图



AS4.0 测试 MTFilterLowPass 代码如下：

变量定义：

```
LowPassFilter : MTFilterLowPass := (0); (*function block MTFilterLowPass*)
Input : REAL := 0.0; (*input signal*)
Output : REAL := 0.0; (*output signal*)
tempSum : LREAL := 0.0;
i : USINT := 0;
```

初始化：

```
/* TODO: Add code here */
/* chosen cycle time: 10ms */
LowPassFilter.CutOffFrequency= 200; /* cut-off frequency [Hz] */
LowPassFilter.Order= 5;             /* 3rd order low-pass filter */
LowPassFilter.Type= mtBUTTERWORTH;/*mtBESSEL;     */  /* bessel filter */
LowPassFilter.Enable= 1;            /* enable function block */
```

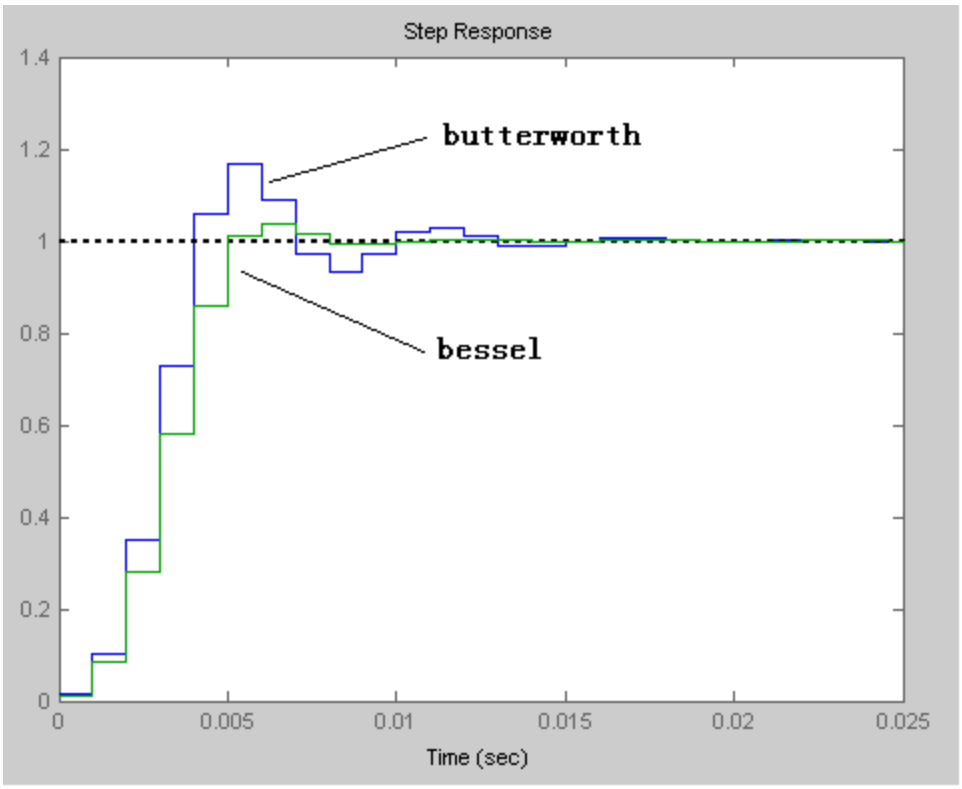
循环体：

```
/* TODO: Add code here */
Input = rand();
LowPassFilter.In= Input;
/* call function block */
MTFilterLowPass(&LowPassFilter);
Output= LowPassFilter.Out;
LowPassFilter.Update= 0;
```

其中离散化后传递函数系数如下图：

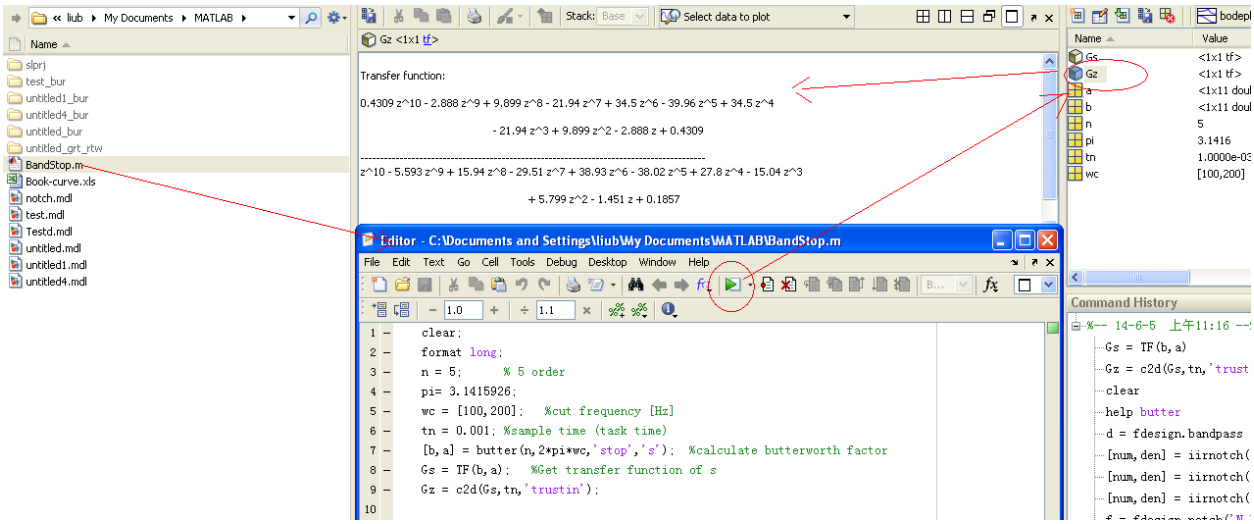
BVector	LREAL[0..5]			
BVector[0]	LREAL			0.0142690759655447
BVector[1]	LREAL			0.067450109971127
BVector[2]	LREAL			0.148707064161447
BVector[3]	LREAL			0.121241512308664
BVector[4]	LREAL			0.0896401605282582
BVector[5]	LREAL			0.0139864946235753
AVector	LREAL[0..5]			
AVector[0]	LREAL			-0.0202038716329268
AVector[1]	LREAL			0.17748286568998
AVector[2]	LREAL			-0.632189706117623
AVector[3]	LREAL			1.33152976698666
AVector[4]	LREAL			-1.40905122697168
AVector[5]	LREAL			1.0

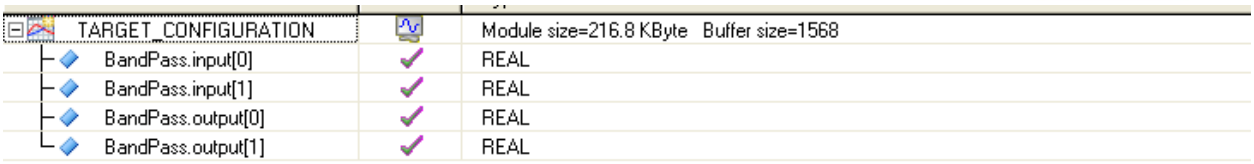
阶跃响应贝塞尔滤波和巴特沃斯滤波比较：

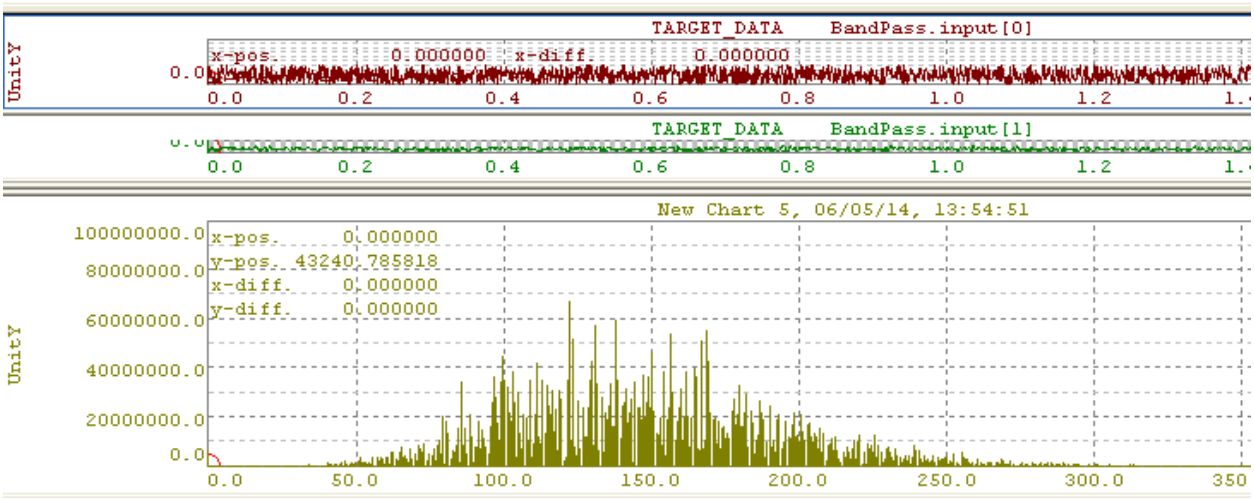


由图可以得出：贝塞尔滤波超调小。

下图为带阻滤波器的 AS 程序的运行结果：







5. 结论

（1）由以上分析可知，高阶滤波器的滤波效果确实比一阶惯性滤波好。但同时阶数越大，其牺牲的相频特性也越大，同时响应滞后也越大。

（2）建议使用方法 1，或者方法 2 的‘zoh’离散化方式。

（3）贝塞尔滤波超调小。

6. 附录

程序名称	说明	下载
BandPass.zip	AS 带通程序	BandPass_AS
BandStop.zip	AS 带阻程序	BandStop_AS
BandPass.m	MATLAB 带通程序	BandPass.m
BandStop.m	MATLAB 带阻程序	BandStop.m

1.4.3. Notch 滤波

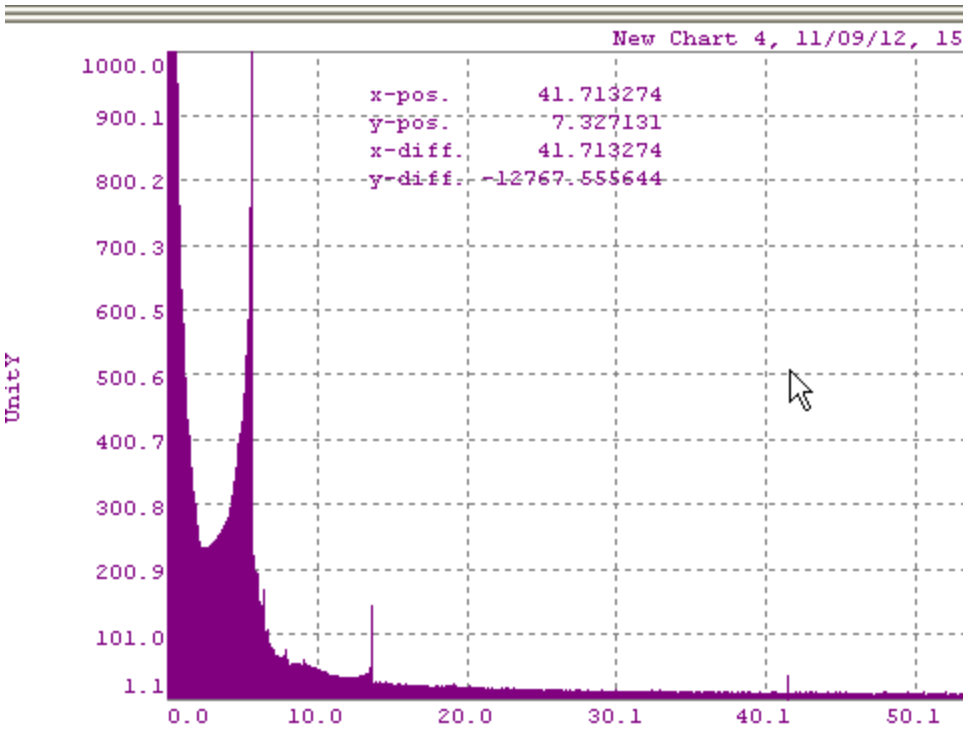
1. 基本信息

编写人	刘柏严	**审核人**	
算法归属	信号处理	**应用对象**	控制信号
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V0.0		详细介绍了实现方法	刘柏严
V0.1	2023.02.08	整理标准化功能块 V4.15 版本原始内容	王宇鑫

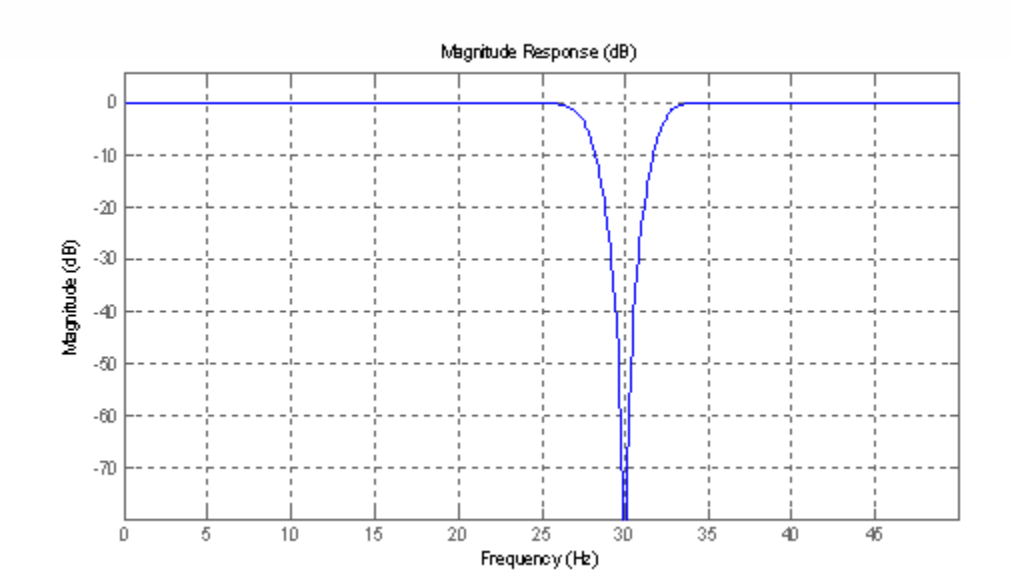
2. 算法简介

Notch 滤波主要在谐振频率较低的传动系统中，用以滤除使系统震动的某一频率信号，可见功能块 MTFilterNotch 介绍 说明文档 。

带有低频谐振频率的系统开环阶跃响应频谱图如下：



Notch filter 的幅值频谱图如下：



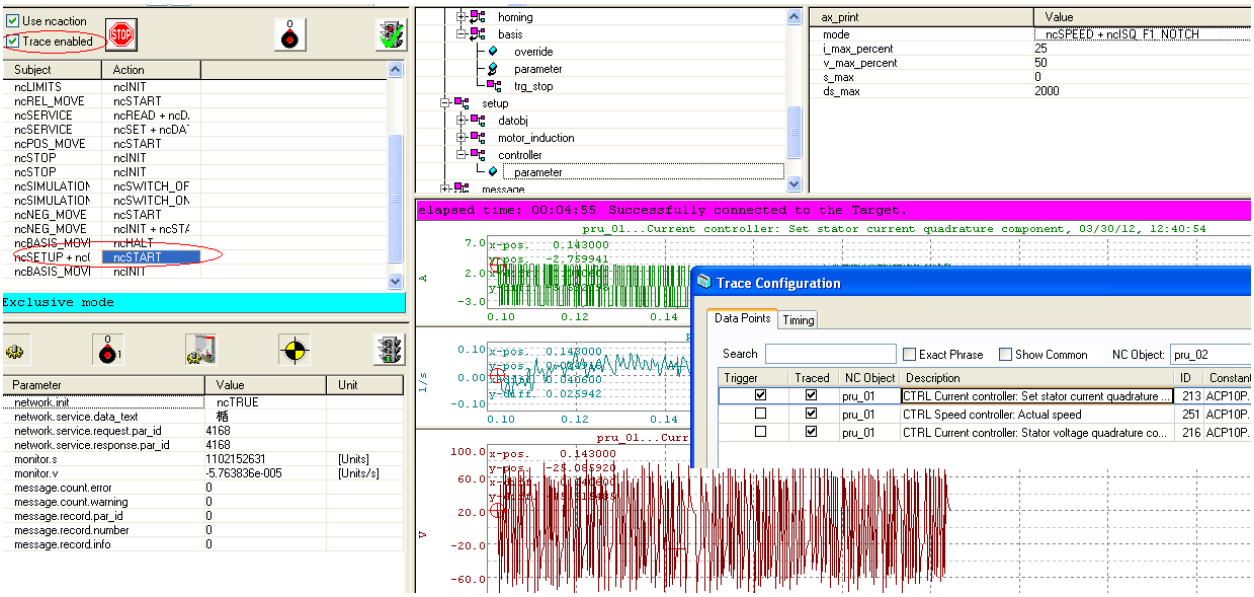
3. 内容

此处介绍一个可用于伺服系统中，分析系统谐振频率的工具。此工具由总部开发，包含 ISQ.fig, ISQ.p 这两个文件，另外需要一个 trace 的配置文件 Speed.mtc。

操作方法如下：

1 打开轴 test

2 导入 Speed.mtc trace 配置文件，



3 trace 结束后，保存为 matlab 格式

4 在 MATLAB 下运行 ISQ.p 分析。

4．工具下载

程序名称	说明	下载
speed.mtc	trace 配置文件	TraceCfg

ISQ.p	ISQ 分析工具	ISQ.p
ISQ.fig	ISQ 配置文件	ISQ.fig

1.4.4. BiQuad 滤波

1. 基本信息

编写人	王宇鑫	**审核人**	
算法归属	信号处理	**应用对象**	控制信号
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V0.0	2023.02.08	整理 AS Help 的内容	王宇鑫

2. 算法简介

BiQuad 滤波器（或补偿滤波器）由两个二次多项式组成其传递函数，一个在分子，一个在分母，见功能块 MTFilterBiQuad 介绍_说明文档 。

$$G(s)=\frac{\frac{s^2}{\omega_n^2}+2\cdot d_n\cdot\frac{s}{\omega_n}+1}{\frac{s^2}{\omega_d^2}+2\cdot d_d\cdot\frac{s}{\omega_d}+1}$$

ω_n ... $2\cdot\pi\cdot CenterFrequencyNum$

ω_d ... $2\cdot\pi\cdot CenterFrequencyDen$

d_n ... $DampingRatioNum$

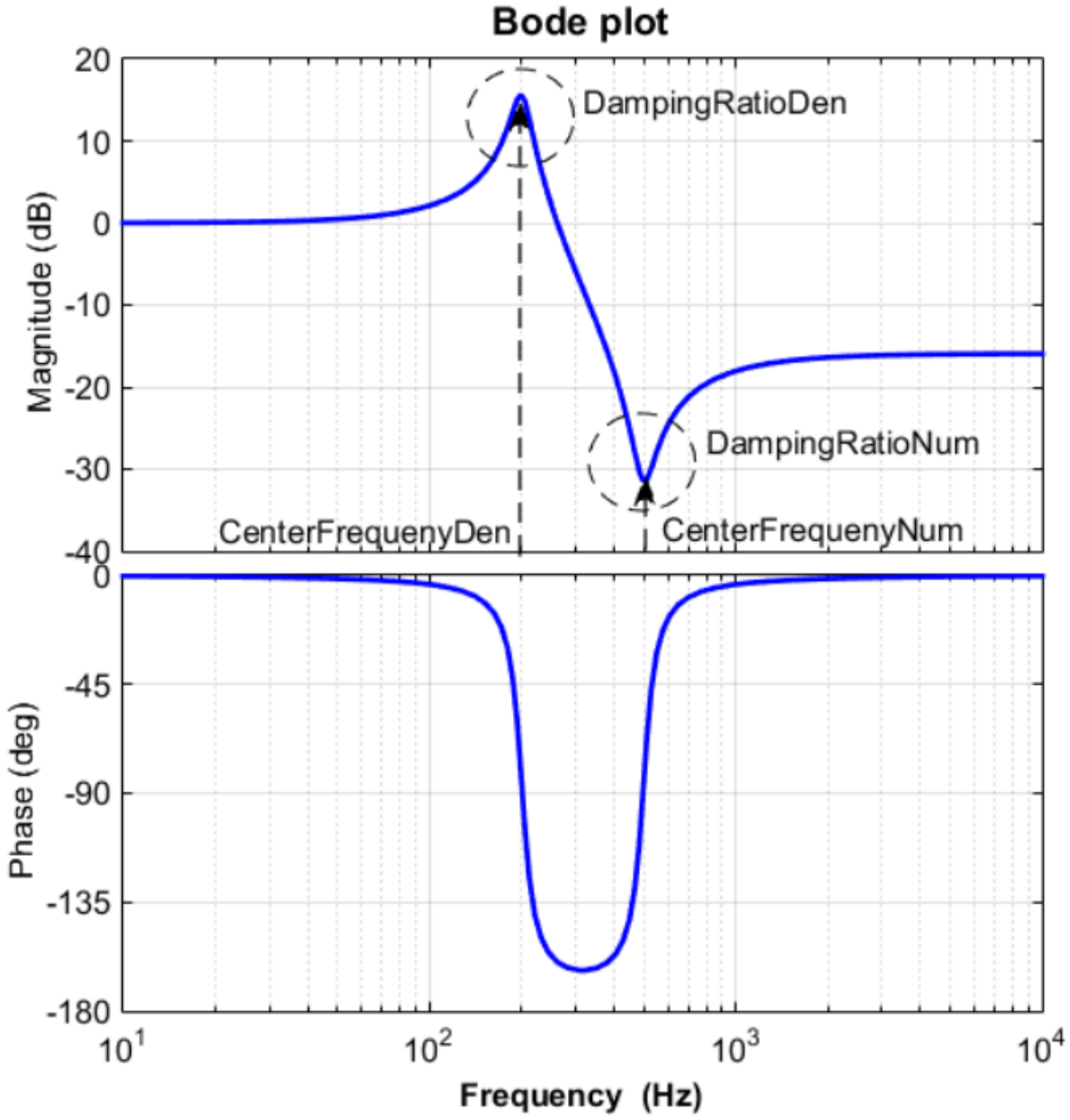
d_d ... $DampingRatioDen$

补偿滤波器通常作为补偿控制器用在受控系统上。从下面的例子可以看出，一个信号频率可以被放大，另一个信号频率可以被阻尼。

滤波器参数为：

- CenterFrequencyDen=200Hz;
- CenterFrequencyNum=500Hz;
- DampingRatioDen=0.07;
- DampingRatioNum=0.07。

的补偿滤波器在 Bode 图中显示如下。振幅峰值将向左、右上或下移，这取决于如何修改滤波器参数。



1.4.5. MTFilter 库中的基于截止频率滤波功能块

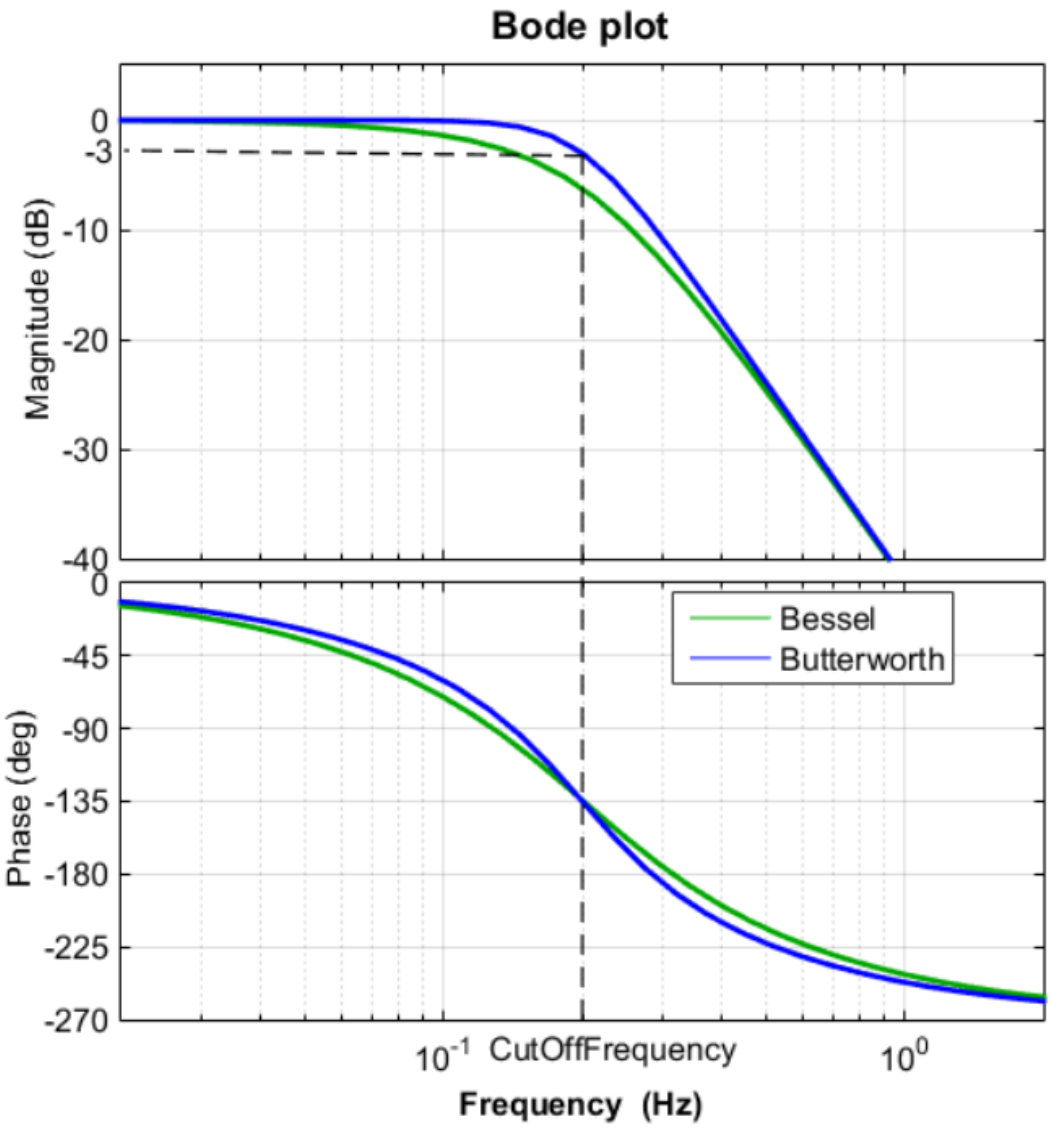
1. 功能块简介

贝加莱 MTFilter 库中，提供了基于截止频率的滤波器功能块供工程师使用，用户通过设置特定的截止频率，进行信号滤波。包括低通、高通、带通、带阻、Notch、BiQuad 几种滤波器。

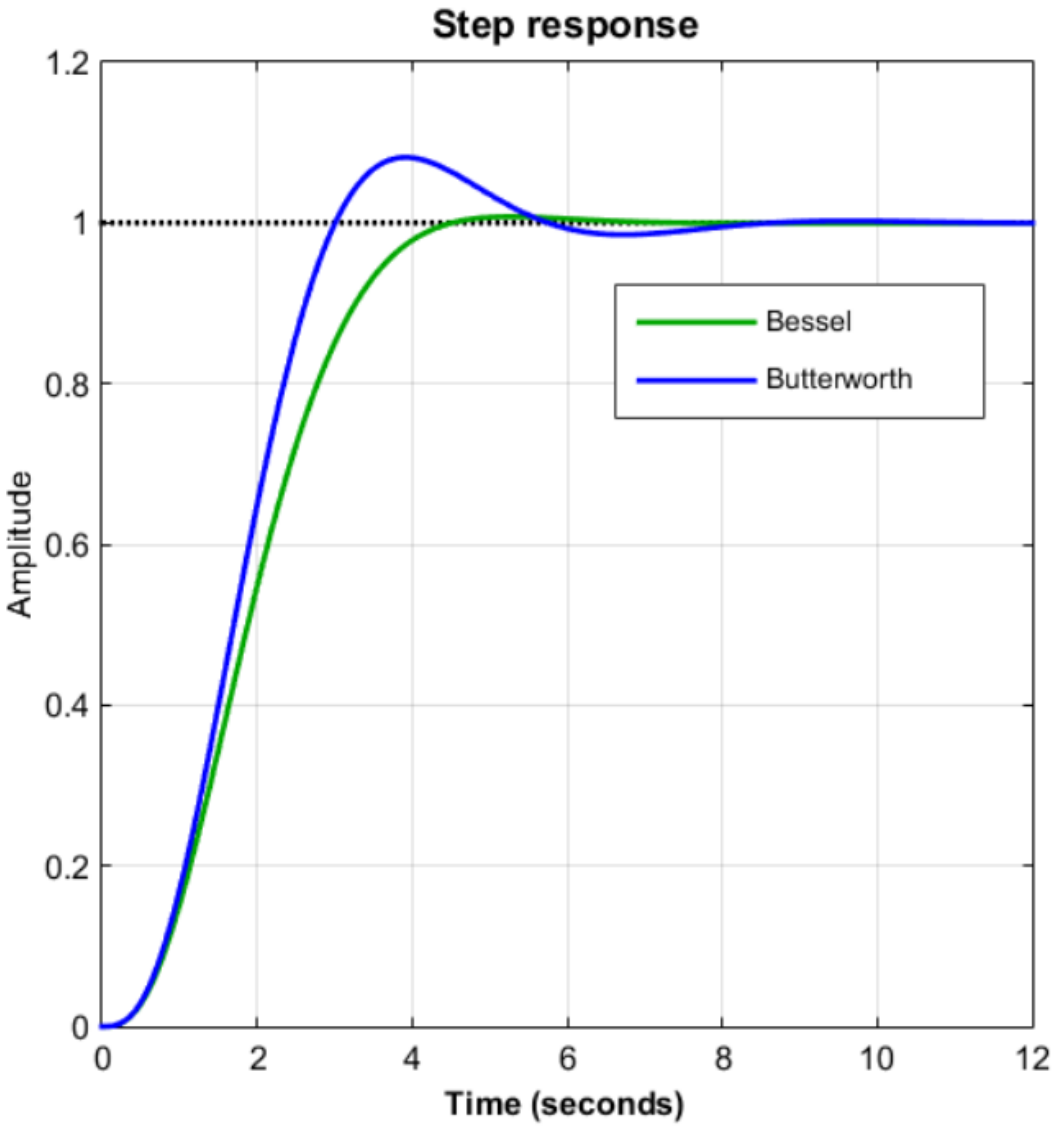
用户可通过配置选择使用贝塞尔（Bessel）或巴特沃斯（Butterworth）实现上述所有滤波器其中的低通、高通、带通、带阻四种滤波器，二者的特性区别为：

- Bessel 滤波器：阶跃响应下超调较小，具有恒定的群延迟；
- Butterworth 滤波器：在其通频带内有一个恒定的振幅响应，在截止频率处的阻尼为-3dB，与滤波器的阶数无关。

下图展示了在截止频率为 0.2Hz 的情况下，Bessel 滤波器和 3 阶 Butterworth 滤波器的频率特性响应。


Bode plot
Magnitude (dB)
0
-3
-10
-20
-30
-40
Phase (deg)
0
-45
-90
-135
-180
-225
-270
Bessel
Butterworth
10^{-1} CutOffFrequency
10^{0}
Frequency (Hz)

下图为二者的阶跃响应：



2. 功能块介绍

- MTFilterLowPass 介绍_说明文档
- MTFilterHighPass 介绍_说明文档
- MTFilterBandPass 介绍_说明文档
- MTFilterBandStop 介绍_说明文档
- MTFilterNotch 介绍_说明文档
- MTFilterBiQuad 介绍_说明文档

1.4.5.1. MTFilterLowPass 介绍_说明文档

1. 功能块功能描述

MTFilterLowPass 为低通滤波器。

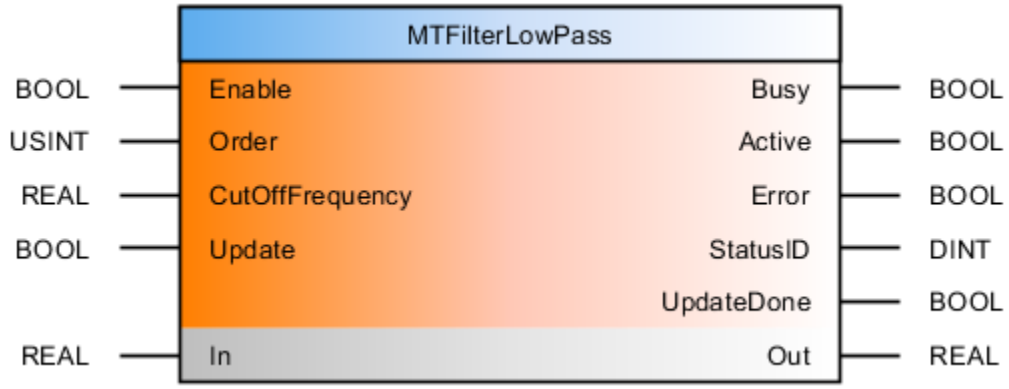
一阶低通滤波器的传递函数可以表示为

$$\frac{Y(s)}{X(s)} = G(s) = \frac{1}{\frac{s}{\omega_C} + 1}$$

输出信号 Y 是由输入信号 X 经过该环节后得到的。

该功能块有两个可配置参数，为截止频率和滤波器阶数。

2. 功能块及接口说明



主要接口说明：

I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	Type	MTFilterTypeEnum	用户可选择选择滤波器的具体实现方式：

			• Type = mtFILTER_BESSEL (default)，选择 Bessel 滤波器； • Type = mtFILTER_BUTTERWORTH，选择 Butterworth 滤波器。
IN	Order	USINT	滤波器阶数： • 允许范围，1 ≤ Order ≤ 5 • 推荐起始值：1 ≤ Order ≤ 1。
IN	CutOffFrequency	REAL	截止频率，单位: [Hz]。 允许设置范围：0 < CutOffFrequency < 采样频率 / 2； 采样频率 = 1/CPU 扫描周期；
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值，滤波前的数值。
OUT	Busy	BOOL	该功能块尚未执行完毕。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值，经过滤波后的值。
OUT	SystemReference	UDINT	系统参考。

3. AS 编程测试

对该功能块进行了相关测试：

AS 版本	AS4.2.3.159	操作系统版本	H4.23	测试库版本	V2.10.0
需要库	MTFilter				
编写人	董俊清	测试人		审核人	
项目名称	MTFilterLowPassTest				
任务说明	对输入信号进行低通滤波。				

（1）变量定义

```
VAR
LowPassFilterHigh : MTFilterLowPass := (0); (*function block MTFilterLowPass*)
LowPassFilterLow : MTFilterLowPass := (0); (*function block MTFilterLowPass*)
InputHighFreq : REAL := 0.0; (*input signal*)
OutputHighFreq : REAL := 0.0; (*output signal*)
InputLowFreq : REAL := 0.0; (*input signal*)
OutputLowFreq : REAL := 0.0; (*output signal*)
t : REAL := 0.0;
RTinfo : RTInfo := (0);
cycT : REAL := 0.0; (*task cycle*)
END_VAR
```

（2）初始化程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
void _INIT ProgramLowPass_INIT(void)
```

```
{
/* input freqency is lower than cut off frequency */
LowPassFilterLow.CutOffFrequency= 0.2; /* center frequency [Hz] */
LowPassFilterLow.Order= 2; /* 2nd order band-pass filter */
//LowPassFilterLow.Type= mtBESSEL; /* bessel filter */
LowPassFilterLow.Enable= 1; /* enable function block */
/* input freqency is higher than center frequency */
LowPassFilterHigh.CutOffFrequency= 0.2; /* center frequency [Hz] */
LowPassFilterHigh.Order= 2; /* 2nd order band-pass filter */
//LowPassFilterHigh.Type= mtBESSEL; /* bessel filter */
LowPassFilterHigh.Enable= 1; /* enable function block */
/* chosen cycle time: */
RTinfo.enable = 1;
RTInfo(&RTinfo);
cycT=RTinfo.cycle_time/1000000.0;
}
```

（3）循环程序

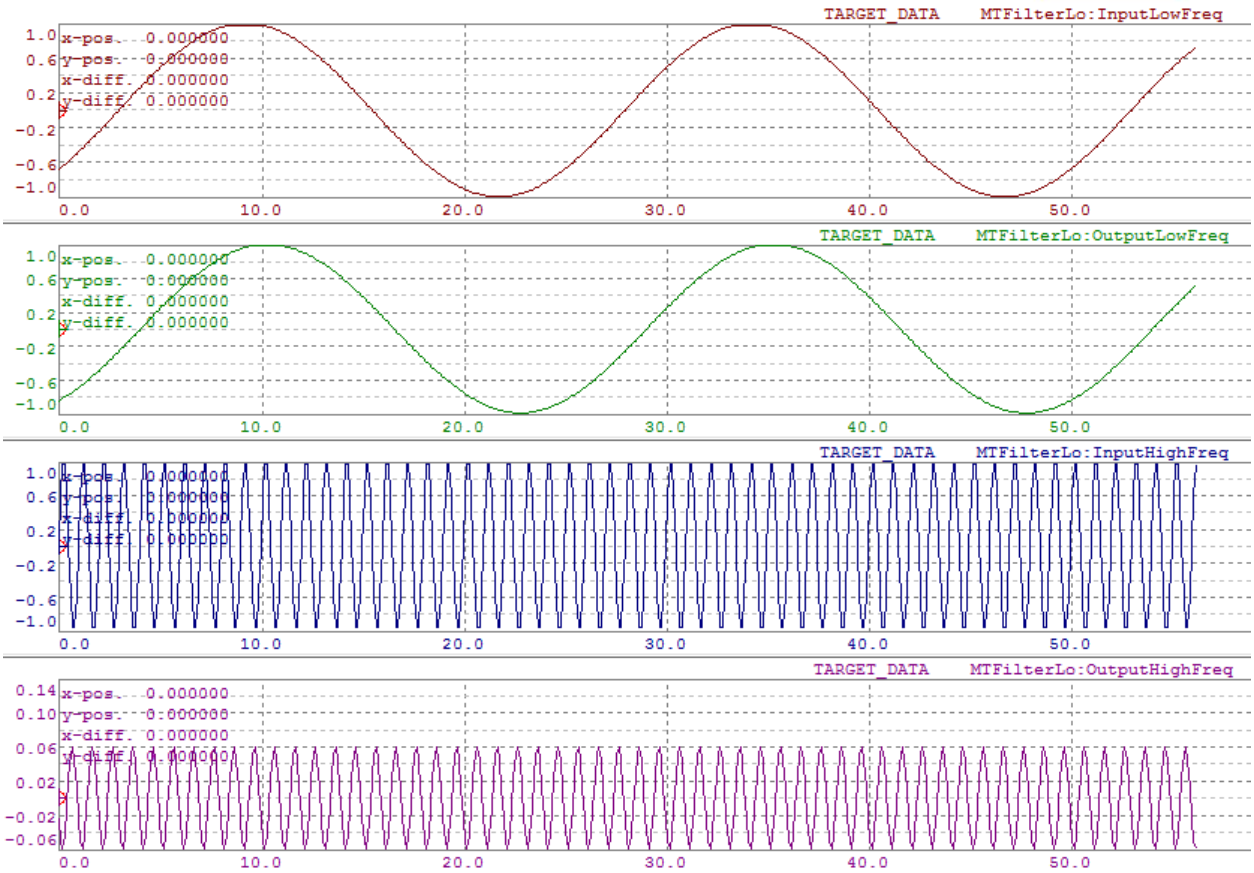
```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
#include "math.h"
#define PI 3.14159265
void _CYCLIC ProgramLowPass_CYCLIC(void)
{
t += cycT;
```

```
InputLowFreq = sin(0.04*2*PI*t);
LowPassFilterLow.In= InputLowFreq;
/* call function block */
MTFilterLowPass(&LowPassFilterLow);
OutputLowFreq= LowPassFilterLow.Out;
InputHighFreq = sin(1*2*PI*t);
LowPassFilterHigh.In= InputHighFreq;
/* call function block */
MTFilterLowPass(&LowPassFilterHigh);
OutputHighFreq= LowPassFilterHigh.Out;
}
```

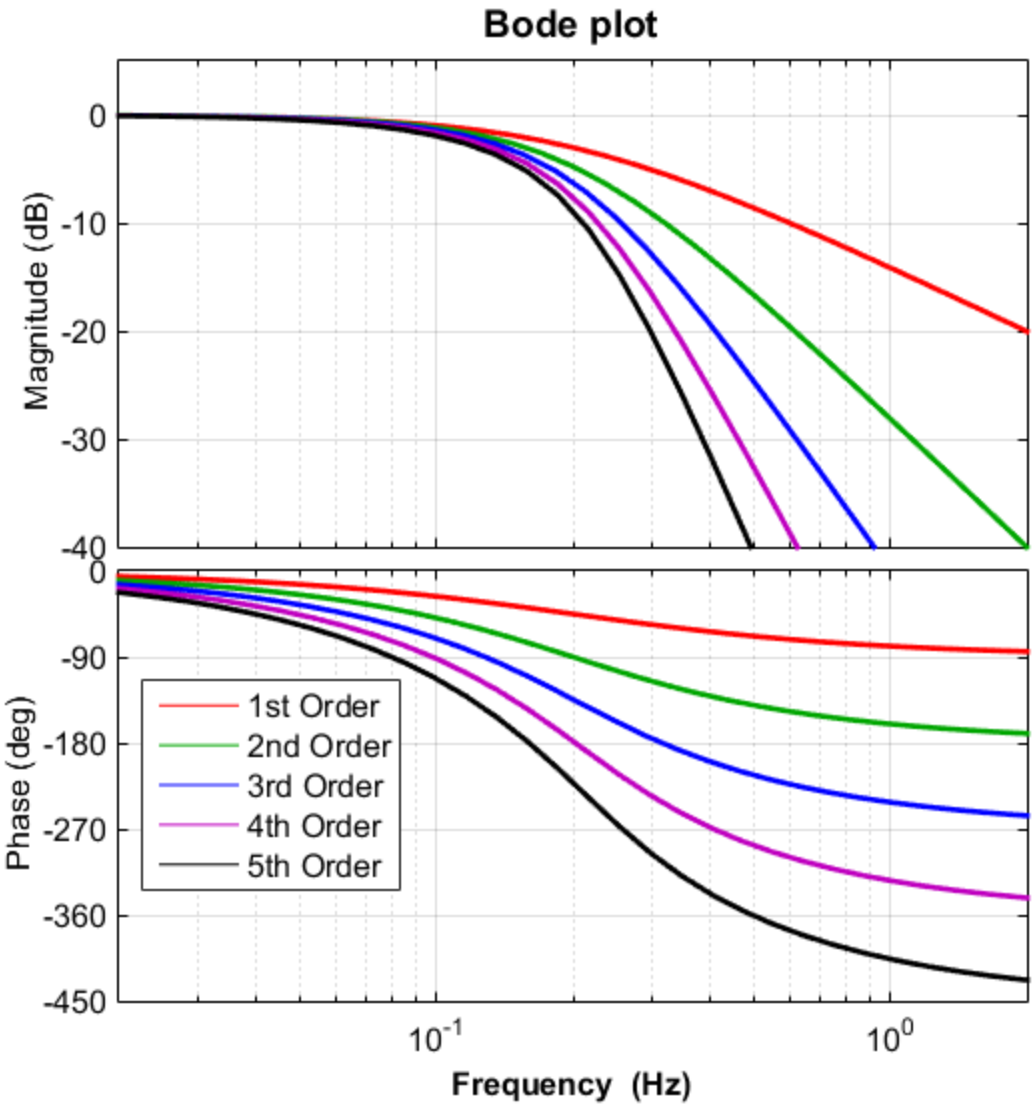
4. 测试结果分析

说明： 任务周期 0.1s

设置同样的低通滤波器，测试在不同频率同样幅值的输入正弦信号下，输出信号的差异



- 输入信号频率大于截止频率时，其输出响应信号的幅值明显衰减，相位滞后输入信号。体现了低通滤波器对高频输入信号的过滤作用。
- 输入信号频率小于截止频率时，输出响应信号幅值保持不变，相位也基本保持不变。而且频率比转折频率越小，幅值和相位的变化更加微小。体现了低通滤波器的特性。
- 时域的响应与频域的伯德图有对应关系。



5. 说明

•

1.4.5.2. MTFilterHighPass 介绍_说明文档

1. 功能块功能描述

MTFilterHighPass 为高通滤波器。

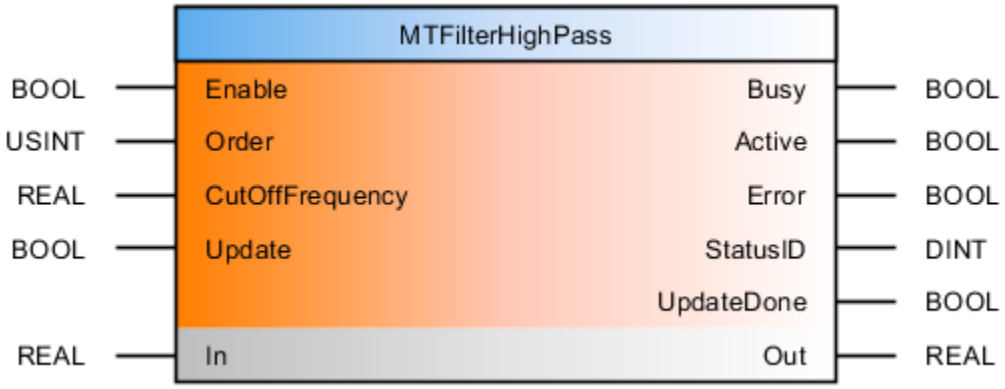
一阶高通滤波器的传递函数可以表示为：

$$\frac{Y(s)}{X(s)} = G(s) = \frac{\frac{s}{\omega_C}}{\frac{s}{\omega_C} + 1}$$

输出信号 Y 是由输入信号 X 经过该环节后得到的。

该功能块有两个可配置参数，为截止频率和滤波器阶数。

2. 功能块及接口说明



主要接口说明：

I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	Type	MTFilterTypeEnum	用户可选择选择滤波器的具体实现方式： • Type = mtFILTER_BESSEL (default)，选择 Bessel 滤波器； • Type = mtFILTER_BUTTERWORTH，选择 Butterworth 滤波器。
IN	Order	USINT	滤波器阶数： • 允许范围，1 ≤ Order ≤ 5 • 推荐起始值：1 ≤ Order ≤ 1。

IN	CutOffFrequency	REAL	截止频率，单位: [Hz]。 允许设置范围：0 < CutOffFrequency < 采样频率 / 2; 采样频率 = 1/CPU 扫描周期;
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值，滤波前的数值。
OUT	Busy	BOOL	该功能块尚未执行完毕。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值，经过滤波后的值。
OUT	SystemReference	UDINT	系统参考。

3．AS 编程测试

对该功能块进行了相关测试：

AS 版本	AS4.2.3.159	操作系统版本	H4.23	测试库版本	V2.10.0
需要库	MTFilter				
编写人	董俊清	测试人		审核人	
项目名称	MTFilterHighPassTest				

任务说明	对输入信号进行高通滤波。				

（1）变量定义：

```
VAR
HighPassFilterHigh : MTFilterHighPass := (0); (*function block MTFilterHighPass*)
HighPassFilterLow : MTFilterHighPass := (0); (*function block MTFilterHighPass*)
InputHighFreq : REAL := 0.0; (*input signal*)
OutputHighFreq : REAL := 0.0; (*output signal*)
InputLowFreq : REAL := 0.0; (*input signal*)
OutputLowFreq : REAL := 0.0; (*output signal*)
t : REAL := 0.0;
RTinfo : RTInfo := (0);
cycT : REAL := 0.0; (*task cycle*)
END_VAR
```

（2）初始化程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
void _INIT ProgramBandPass_INIT(void)
{
/* input freqency is lower than cut off frequency */
HighPassFilterLow.CutOffFrequency= 0.2; /* center frequency [Hz] */
HighPassFilterLow.Order= 2; /* 2nd order band-pass filter */
//HighPassFilterLow.Type= mtBESSEL; /* bessel filter */
HighPassFilterLow.Enable= 1; /* enable function block */
```

```
/* input freqency is higher than center frequency */
HighPassFilterHigh.CutOffFrequency= 0.2; /* center frequency [Hz] */
HighPassFilterHigh.Order= 2; /* 2nd order band-pass filter */
//HighPassFilterLow.Type= mtBESSEL; /* bessel filter */
HighPassFilterHigh.Enable= 1; /* enable function block */
/* chosen cycle time: */
RTinfo.enable = 1;
RTInfo(&RTinfo);
cycT=RTinfo.cycle_time/1000000.0;
}
```

（3）循环程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
#include "math.h"
#define PI 3.14159265
void _CYCLIC ProgramHighPass_CYCLIC(void)
{
t += cycT;
InputLowFreq = sin(0.04*2*PI*t);
HighPassFilterLow.In= InputLowFreq;
/* call function block */
MTFilterHighPass(&HighPassFilterLow);
OutputLowFreq= HighPassFilterLow.Out;
InputHighFreq = sin(1*2*PI*t);
```

HighPassFilterHigh.In= InputHighFreq;

/* call function block */

MTFilterHighPass(&HighPassFilterHigh);

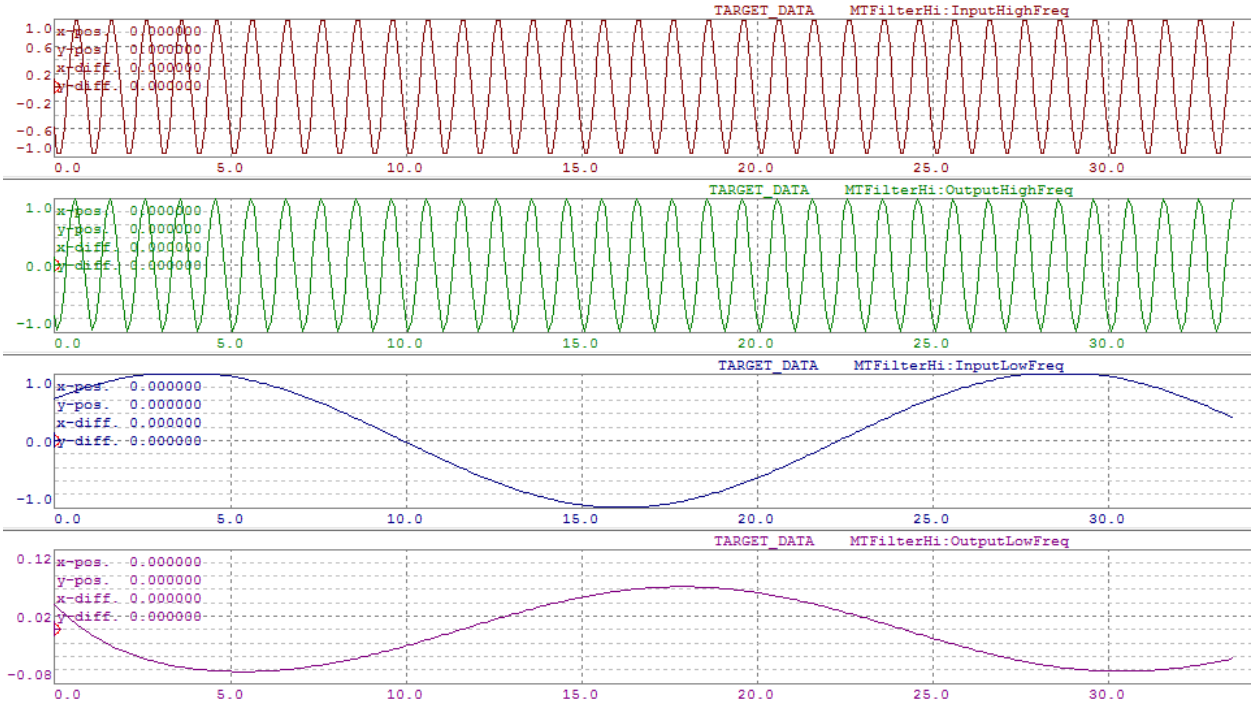
OutputHighFreq= HighPassFilterHigh.Out;

}

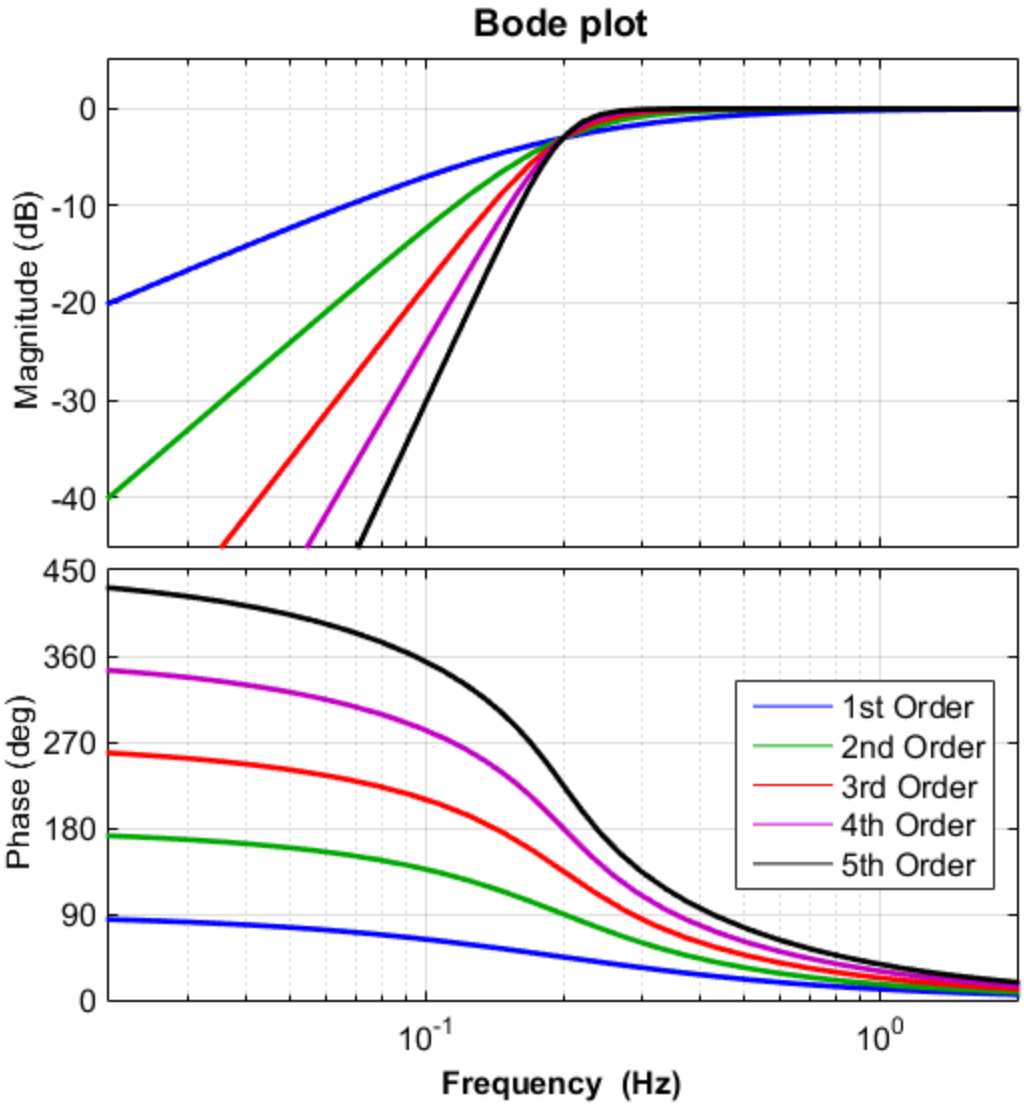
4. 测试结果分析

说明： 任务周期 0.1s

设置同样的高通滤波器，测试在不同频率同样幅值的输入正弦信号下，输出信号的差异



- 输入信号频率小于转折频率时，其输出响应信号的幅值明显衰减，相位超前输入信号。体现了高通滤波器对低频输入信号的过滤作用。
- 输入信号频率大于转折频率时，输出响应信号幅值保持不变，相位也基本保持不变。而且频率比转折频率越大，幅值和相位的变化更加微小。体现了高通滤波器的特性。
- 时域的响应与频域的伯德图有对应关系。



5. 说明

- 滤波器类型的枚举定义(MTFilterTypeEnumFiltertype)，编译时报错。

1.4.5.3. MTFilterBandPass 介绍_说明文档

1. 功能块功能描述

MTFilterBandPass 为带通滤波器。

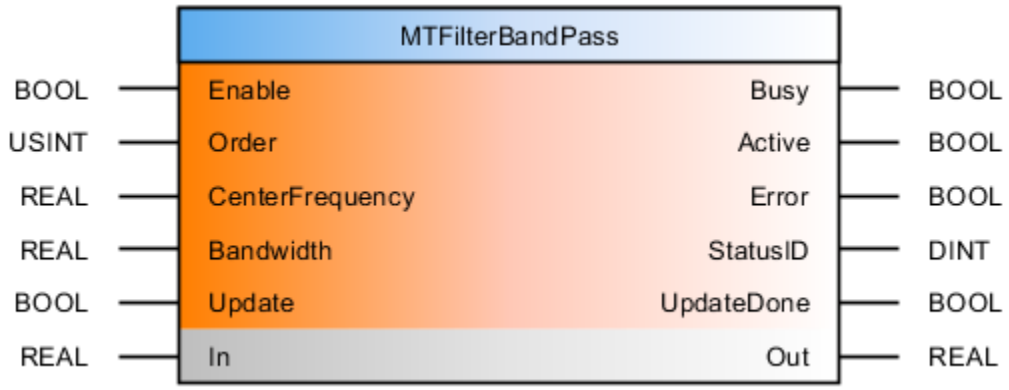
一阶带通滤波器的传递函数可以表示为

$$G(s)=\frac{\frac{\omega_B}{\omega_c^2}\cdot s}{\frac{s^2}{\omega_c^2}+\frac{\omega_B}{\omega_c^2}\cdot s+1}$$

输出信号 Y 是由输入信号 X 经过该环节后得到的。

该功能块有三个可配置参数，分别为中心频率，带宽和滤波器阶数。

2. 功能块及接口说明



主要接口说明：

I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	Type	MTFilterTypeEnum	用户可选择选择滤波器的具体实现方式： • Type = mtFILTER_BESSEL (default)，选择 Bessel 滤波器； • Type = mtFILTER_BUTTERWORTH，选择 Butterworth 滤波器。
IN	Order	USINT	滤波器阶数： • 允许范围，1 ≤ Order ≤ 2； • 推荐起始值：1 ≤ Order ≤ 1。

IN	CenterFrequency	REAL	中心频率，单位: [Hz]。 允许设置范围：0 < CenterFrequency < 采样频率 / 2; 采样频率 = 1/CPU 扫描周期;
IN	Bandwidth	REAL	带宽，单位：[Hz]; 允许设置范围：0 < Bandwidth < 2 * 中心频率;
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值，滤波前的数值。
OUT	Busy	BOOL	该功能块尚未执行完毕。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值，经过滤波后的值。
OUT	SystemReference	UDINT	系统参考。

3. AS 编程测试

对该功能块进行了相关测试：

AS 版本	AS4.2.3.159	操作系统版本	H4.23	测试库版本	V2.10.0
需要库	MTFilter				
编写人	董俊清	测试人		审核人	

项目名称	MTFilterBandPassTest				
任务说明	对输入信号进行带通滤波。				

（1）变量定义

```
VAR
BandPassFilterHigh : MTFilterBandPass := (0); (*function block MTFilterBandPass*)
BandPassFilterMedium : MTFilterBandPass := (0); (*function block MTFilterBandPass*)
BandPassFilterLow : MTFilterBandPass := (0); (*function block MTFilterBandPass*)
InputHighFreq : REAL := 0.0; (*input signal*)
OutputHighFreq : REAL := 0.0; (*output signal*)
InputMediumFreq : REAL := 0.0; (*input signal*)
OutputMediumFreq : REAL := 0.0; (*output signal*)
InputLowFreq : REAL := 0.0; (*input signal*)
OutputLowFreq : REAL := 0.0; (*output signal*)
t : REAL := 0.0;
RTinfo : RTInfo := (0);
cycT : REAL := 0.0; (*task cycle*)
END_VAR
```

（2）初始化程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
void _INIT ProgramBandPass_INIT(void)
```

```
{
/* input freqency is center frequency */
BandPassFilterMedium.CenterFrequency= 0.2; /* center frequency [Hz] */
BandPassFilterMedium.Order= 2; /* 2nd order band-pass filter */
BandPassFilterMedium.Bandwidth= 0.3; /* bandwidth [Hz] */
BandPassFilterMedium.Enable= 1; /* enable function block */
/* input freqency is lower than center frequency */
BandPassFilterLow.CenterFrequency= 0.2; /* center frequency [Hz] */
BandPassFilterLow.Order= 2; /* 2nd order band-pass filter */
BandPassFilterLow.Bandwidth= 0.3; /* bandwidth [Hz] */
BandPassFilterLow.Enable= 1; /* enable function block */
/* input freqency is higher than center frequency */
BandPassFilterHigh.CenterFrequency= 0.2; /* center frequency [Hz] */
BandPassFilterHigh.Order= 2; /* 2nd order band-pass filter */
BandPassFilterHigh.Bandwidth= 0.3; /* bandwidth [Hz] */
BandPassFilterHigh.Enable= 1; /* enable function block */
/* chosen cycle time: */
RTinfo.enable = 1;
RTInfo(&RTinfo);
cycT=RTinfo.cycle_time/1000000.0;
}
```

（3） 循环程序

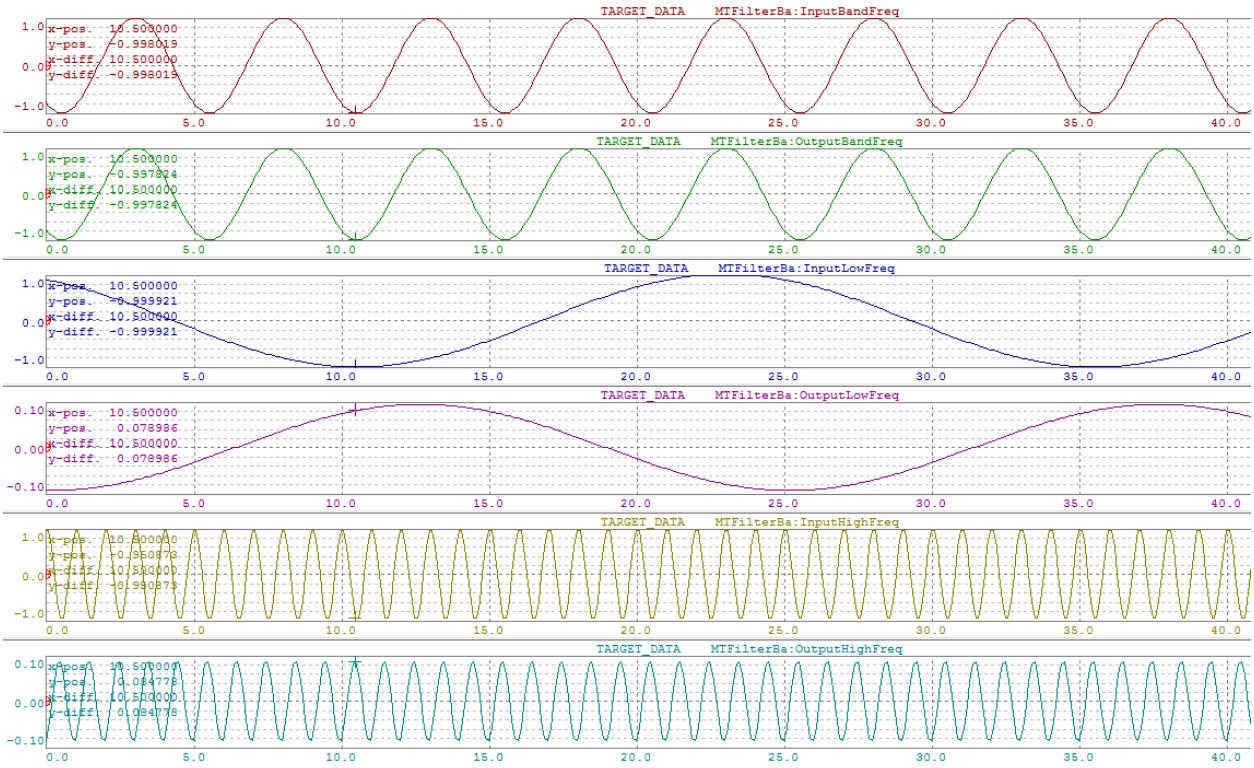
```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
#include "math.h"
```

```
#define PI 3.14159265
void _CYCLIC ProgramBandPass_CYCLIC(void)
{
t += cycT;
InputMediumFreq = sin(0.2*2*PI*t);
BandPassFilterMedium.In= InputMediumFreq;
/* call function block */
MTFilterBandPass(&BandPassFilterMedium);
OutputMediumFreq= BandPassFilterMedium.Out;
InputLowFreq = sin(0.04*2*PI*t);
BandPassFilterLow.In= InputLowFreq;
/* call function block */
MTFilterBandPass(&BandPassFilterLow);
OutputLowFreq= BandPassFilterLow.Out;
InputHighFreq = sin(1*2*PI*t);
BandPassFilterHigh.In= InputHighFreq;
/* call function block */
MTFilterBandPass(&BandPassFilterHigh);
OutputHighFreq= BandPassFilterHigh.Out;
}
```

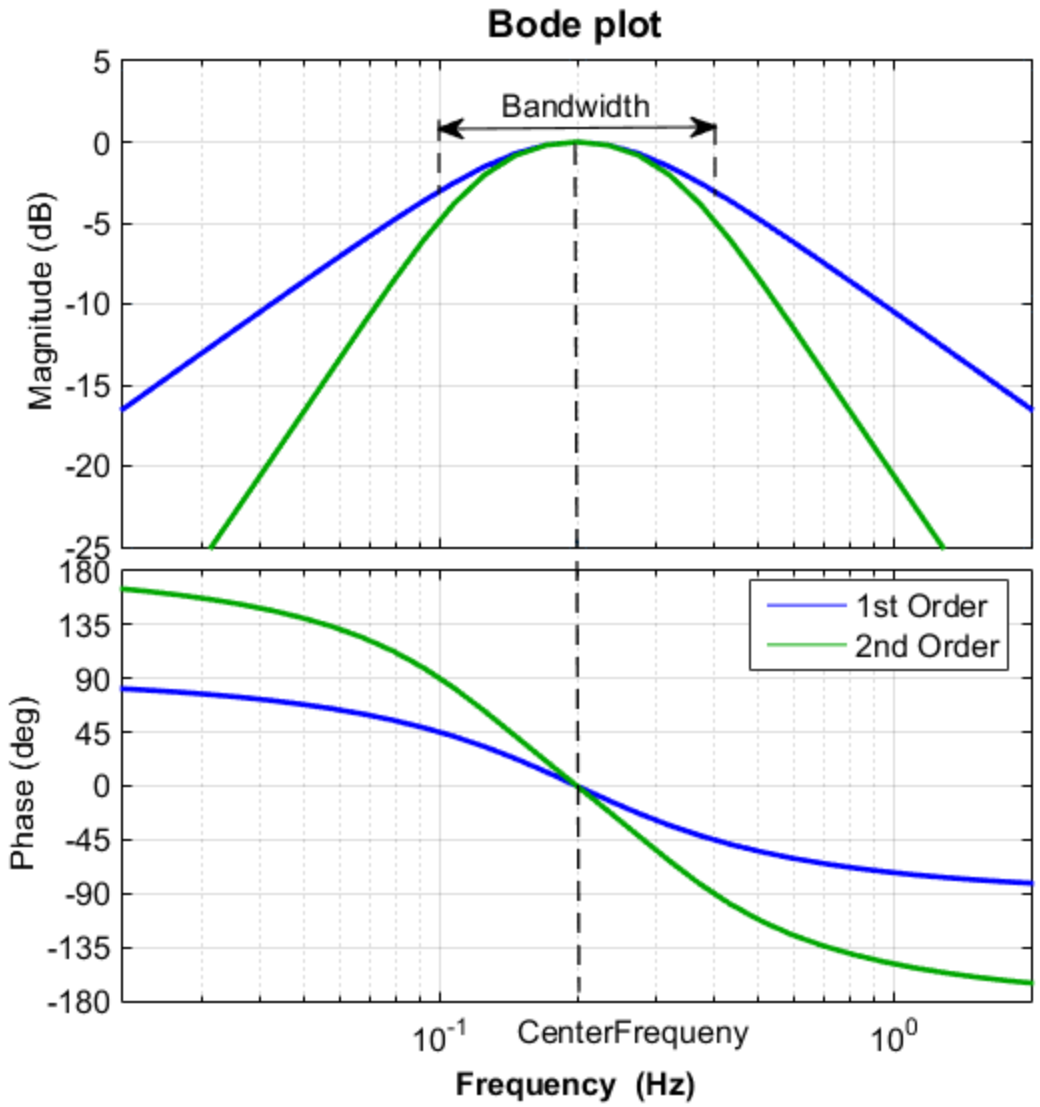
4. 测试结果分析

说明： 任务周期 0.1s

设置同样的带通滤波器，测试在不同频率同样幅值的输入正弦信号下，输出信号的差异



- 输入信号频率与带通频率一样时，其输出响应信号与输入信号幅值相同，相位也基本不变。
- 输入信号频率小于带通频率，输出响应信号幅值明显衰减，相位产生超前。输入信号频率大于带通频率，输出响应信号幅值明显衰减，相位产生滞后。幅值衰减的比例和相位变化的量可以从带通滤波器的伯德图上得到印证。



5. 说明

- 滤波器类型的枚举定义(MTFilterTypeEnumFiltertype)，编译时报错。

1.4.5.4. MTFilterBandStop 介绍_说明文档

1. 功能块功能描述

MTFilterBandStop 为带阻滤波器。

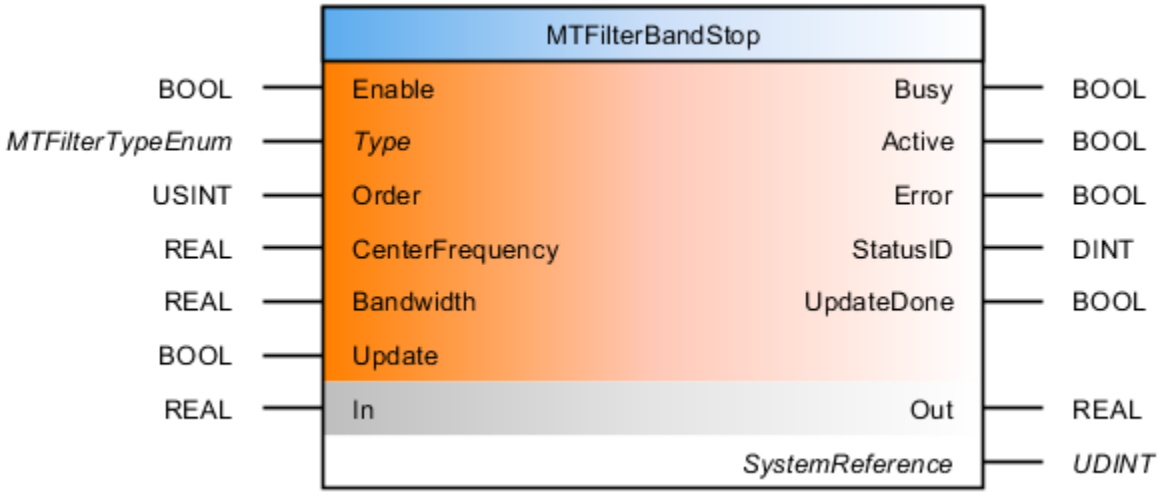
一阶带阻滤波器的传递函数可以表示为

$$G(s)=\frac{\frac{s^2}{\omega_c^2}+1}{\frac{s^2}{\omega_c^2}+\frac{\omega_B}{\omega_c^2}\cdot s+1}$$

输出信号 Y 是由输入信号 X 经过该环节后得到的。

该功能块有三个可配置参数，分别为中心频率，带宽和滤波器阶数。

2. 功能块及接口说明



主要接口说明：

I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	Type	MTFilterTypeEnum	用户可选择选择滤波器的具体实现方式： • Type = mtFILTER_BESSEL (default)，选择 Bessel 滤波器； • Type = mtFILTER_BUTTERWORTH，选择 Butterworth 滤波器。
IN	Order	USINT	滤波器阶数： • 允许范围，1 ≤ Order ≤ 2；

			• 推荐起始值：1 ≤ Order ≤ 1。
IN	CenterFrequency	REAL	中心频率，单位: [Hz]。 允许设置范围：0 < CenterFrequency < 采样频率 / 2; 采样频率 = 1/CPU 扫描周期;
IN	Bandwidth	REAL	带宽，单位：[Hz]; 允许设置范围：0 < Bandwidth < 2 * 中心频率;
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值，滤波前的数值。
OUT	Busy	BOOL	该功能块尚未执行完毕。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值，经过滤波后的值。
OUT	SystemReference	UDINT	系统参考。

3. AS 编程测试

对该功能块进行了相关测试：

AS 版本	AS4.2.3.159	操作系统版本	H4.23	测试库版本	V2.10.0
需要库	MTFilter				

编写人	董俊清	测试人		审核人	
项目名称	MTFilterBandStopTest				
任务说明	对输入信号进行带通滤波。				

（1）变量定义

```
VAR
BandStopFilterHigh : MTFilterBandStop := (0); (*function block MTFilterBandStop*)
BandStopFilterMedium : MTFilterBandStop := (0); (*function block MTFilterBandStop*)
BandStopFilterLow : MTFilterBandStop := (0); (*function block MTFilterBandStop*)
InputHighFreq : REAL := 0.0; (*input signal*)
OutputHighFreq : REAL := 0.0; (*output signal*)
InputMediumFreq : REAL := 0.0; (*input signal*)
OutputMediumFreq : REAL := 0.0; (*output signal*)
InputLowFreq : REAL := 0.0; (*input signal*)
OutputLowFreq : REAL := 0.0; (*output signal*)
t : REAL := 0.0;
RTinfo : RTInfo := (0);
cycT : REAL := 0.0; (*task cycle*)
END_VAR
```

（2）初始化程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
```

```
void _INIT ProgramBandStop_INIT(void)
{
/* input freqency is center frequency */
BandStopFilterMedium.CenterFrequency= 0.2; /* center frequency [Hz] */
BandStopFilterMedium.Order= 2; /* 2nd order band-stop filter*/
BandStopFilterMedium.Bandwidth= 0.3; /* bandwidth [Hz] */
BandStopFilterMedium.Enable= 1; /* enable function block */
/* input freqency is lower than center frequency */
BandStopFilterLow.CenterFrequency= 0.2; /* center frequency [Hz] */
BandStopFilterLow.Order= 2; /* 2nd order band-stop filter */
BandStopFilterLow.Bandwidth= 0.3; /* bandwidth [Hz] */
BandStopFilterLow.Enable= 1; /* enable function block */
/* input freqency is higher than center frequency */
BandStopFilterHigh.CenterFrequency= 0.2; /* center frequency [Hz] */
BandStopFilterHigh.Order= 2; /* 2nd order band-stop filter*/
BandStopFilterHigh.Bandwidth= 0.3; /* bandwidth [Hz] */
BandStopFilterHigh.Enable= 1; /* enable function block */
/* chosen cycle time: */
RTinfo.enable = 1;
RTInfo(&RTinfo);
cycT=RTinfo.cycle_time/1000000.0;
}
```

（3）循环程序

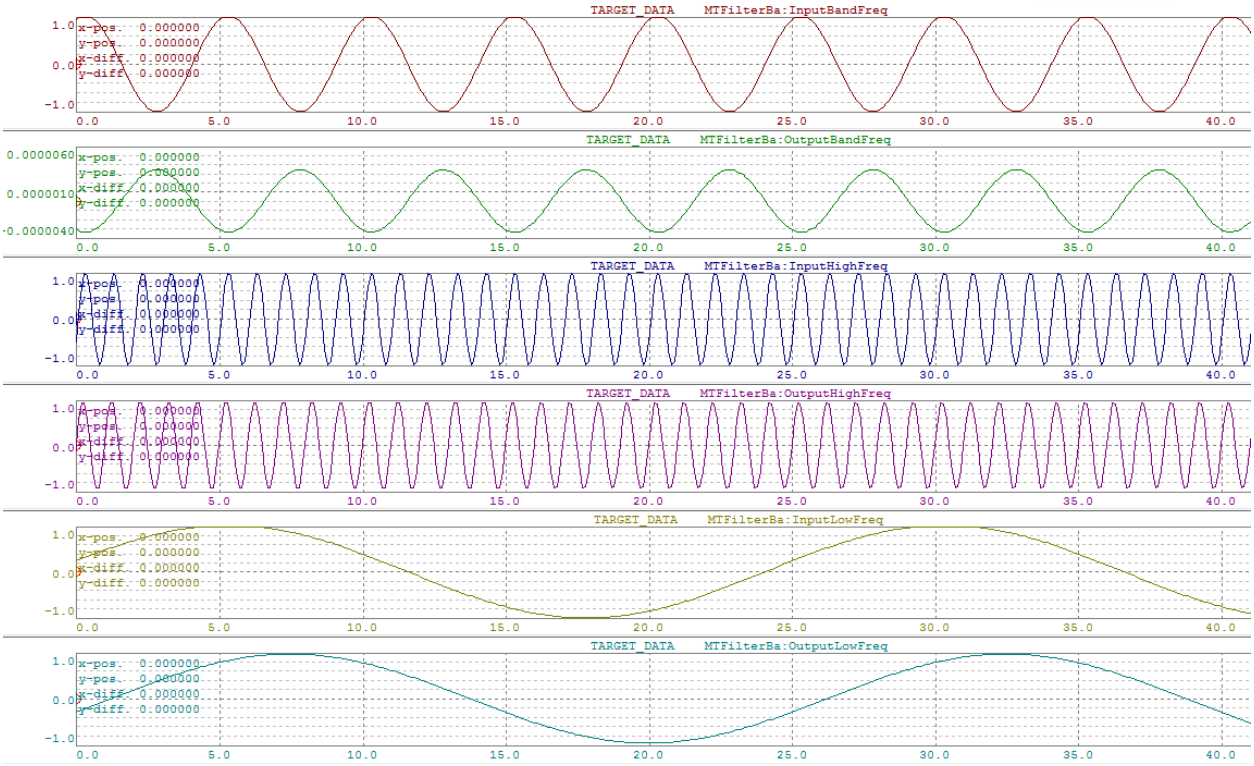
```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
```

```
#include "math.h"
#define PI 3.14159265
void _CYCLIC ProgramBandStop_CYCLIC(void)
{
t += cycT;
InputMediumFreq = sin(0.2*2*PI*t);
BandStopFilterMedium.In= InputMediumFreq;
/* call function block */
MTFilterBandStop(&BandStopFilterMedium);
OutputMediumFreq= BandStopFilterMedium.Out;
InputLowFreq = sin(0.04*2*PI*t);
BandStopFilterLow.In= InputLowFreq;
/* call function block */
MTFilterBandStop(&BandStopFilterLow);
OutputLowFreq= BandStopFilterLow.Out;
InputHighFreq = sin(1*2*PI*t);
BandStopFilterHigh.In= InputHighFreq;
/* call function block */
MTFilterBandStop(&BandStopFilterHigh);
OutputHighFreq= BandStopFilterHigh.Out;
}
```

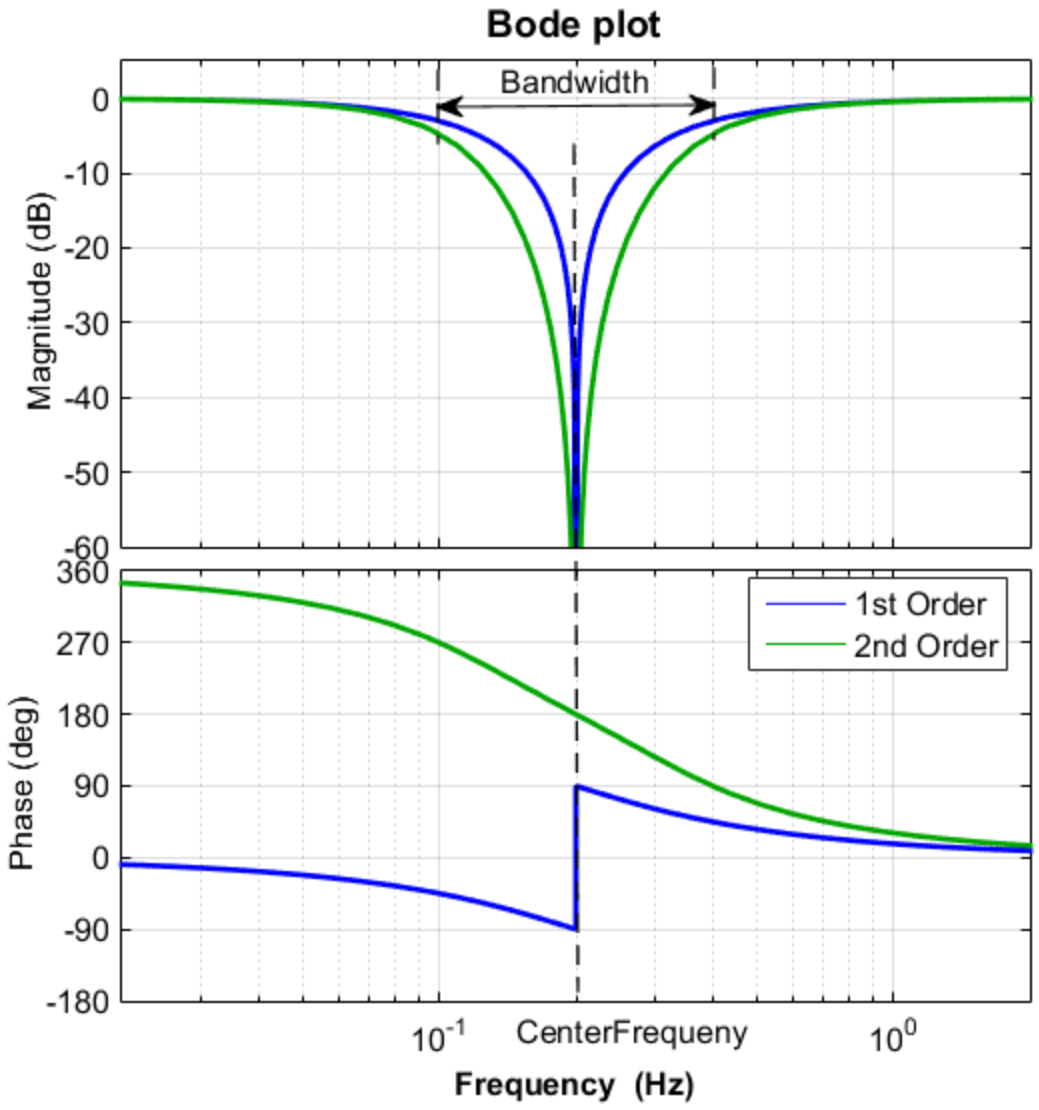
4. 测试结果分析

说明： 任务周期 0.1s

设置同样的带阻滤波器，测试在不同频率同样幅值的输入正弦信号下，输出信号的差异



- 输入信号频率与带阻频率一样时，其输出响应信号幅值明显衰减，相位基本反向。
- 输入信号频率小于带阻频率或者输入信号频率大于带阻频率，输出响应信号的幅值和相位变化不明显，而且离带阻中心频率越远，变化越不明显。这可以从带阻滤波器的伯德图得到印证。



5. 说明

•

1.4.5.5. MTFilterNotch 介绍_说明文档

1. 功能块功能描述

MTFilterNotch 滤波器的传递函数可以表示为

$$G(s)=\frac{\frac{s^2}{\omega_c^2}+1}{\frac{s^2}{\omega_c^2}+2\cdot D\cdot\frac{s}{\omega_c}+1}$$

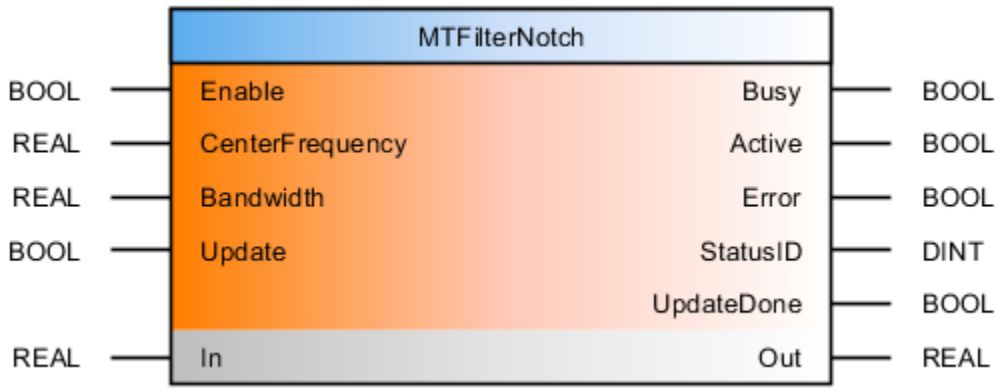
$$D=\frac{1}{2}\cdot\left(\frac{f_c+\frac{B}{2}}{f_c}-\frac{f_c}{f_c+\frac{B}{2}}\right)$$

其中$\omega_c = 2\pi f_c$,

fc 为中心频率，B 为带宽，D 为阻尼。

该功能块有两个可配置参数，分别为中心频率和带宽。

2. 功能块及接口说明



主要接口说明：

I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	CenterFrequency	REAL	中心频率，单位: [Hz]。 允许设置范围：0 < CenterFrequency < 采样频率 / 2; 采样频率 = 1/CPU 扫描周期;
IN	Bandwidth	REAL	带宽，单位：[Hz];

			允许设置范围：0 < Bandwidth < 2 * 中心频率；
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值，滤波前的数值。
OUT	Busy	BOOL	该功能块尚未执行完毕。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值，经过滤波后的值。
OUT	SystemReference	UDINT	系统参考。

3. AS 编程测试

对该功能块进行了相关测试：

AS 版本	AS4.2.3.159	操作系统版本	H4.23	测试库版本	V2.10.0
需要库	MTFilter				
编写人	董俊清	测试人		审核人	
项目名称	MTFilterNotchTest				
任务说明	对输入信号进行 Notch 滤波。				

（1）变量定义

VAR

NotchFilterLow : MTFilterNotch := (0); (*function block MTFilterNotch*)

NotchFilterMedium : MTFilterNotch := (0); (*function block MTFilterNotch*)

NotchFilterHigh : MTFilterNotch := (0); (*function block MTFilterNotch*)

cycT : REAL := 0.0; (*task cycle*)

t : REAL := 0.0;

OutputLowFreq : REAL := 0.0; (*output signal*)

InputLowFreq : REAL := 0.0; (*input signal*)

OutputMediumFreq : REAL := 0.0; (*output signal*)

InputMediumFreq : REAL := 0.0; (*input signal*)

OutputHighFreq : REAL := 0.0; (*output signal*)

InputHighFreq : REAL := 0.0; (*input signal*)

RTinfo : RTInfo := (0);

END_VAR

（2）初始化程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
void _INIT ProgramNotch_INIT(void)
{
/* input freqency is center frequency */
NotchFilterMedium.CenterFrequency= 0.2; /* center frequency [Hz] */
NotchFilterMedium.Bandwidth= 0.3; /* bandwidth [Hz] */
NotchFilterMedium.Enable= 1; /* enable function block */
/* input freqency is lower than center frequency */
```

```
NotchFilterLow.CenterFrequency= 0.2; /* center frequency [Hz] */
NotchFilterLow.Bandwidth= 0.3; /* bandwidth [Hz] */
NotchFilterLow.Enable= 1; /* enable function block */
/* input freqency is higher than center frequency */
NotchFilterHigh.CenterFrequency= 0.2; /* center frequency [Hz] */
NotchFilterHigh.Bandwidth= 0.3; /* bandwidth [Hz] */
NotchFilterHigh.Enable= 1; /* enable function block */
/* chosen cycle time: */
RTinfo.enable = 1;
RTInfo(&RTinfo);
cycT=RTinfo.cycle_time/1000000.0;
}
```

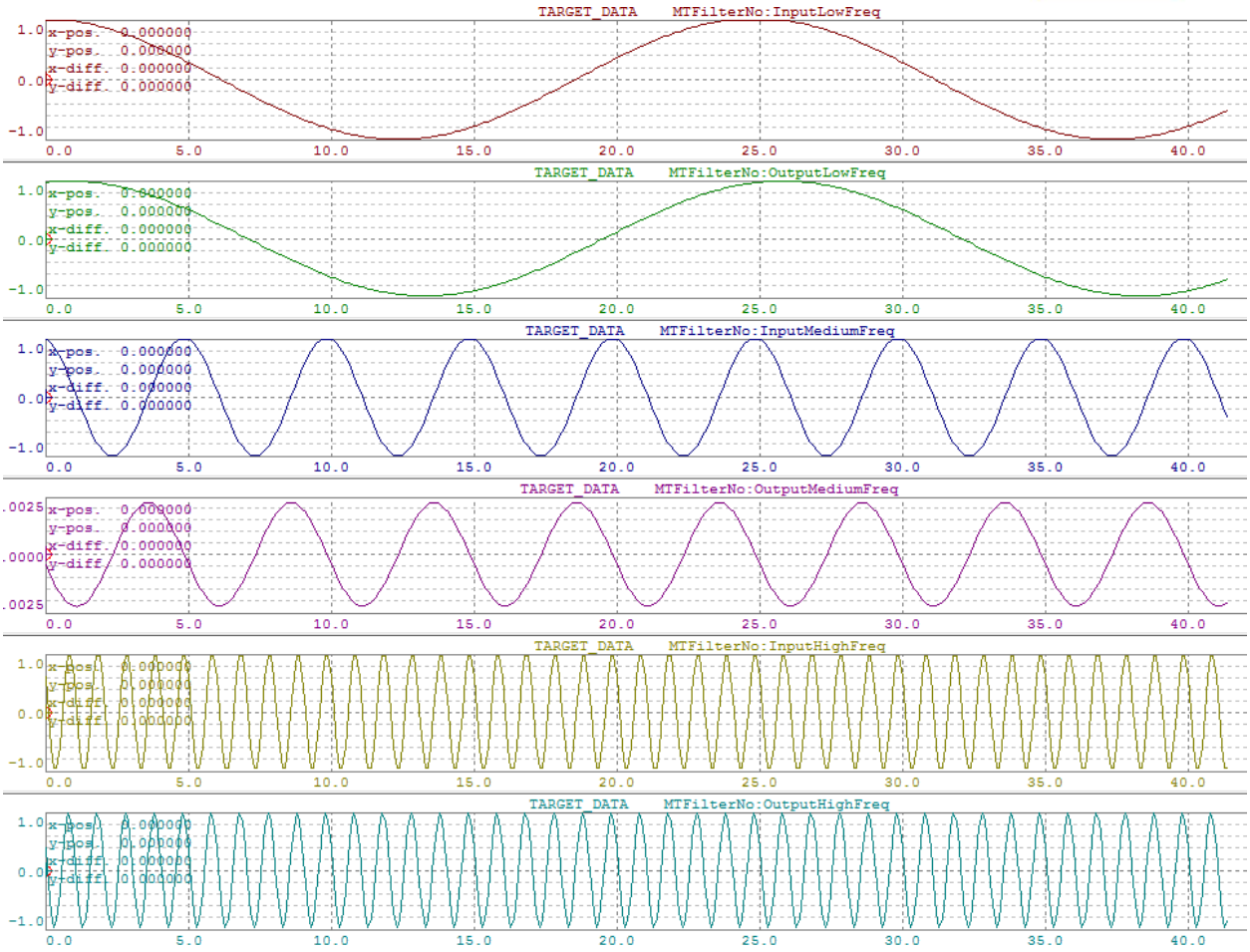
（3） 循环程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
#include "math.h"
#define PI 3.14159265
void _CYCLIC ProgramNotch_CYCLIC(void)
{
t += cycT;
InputMediumFreq = sin(0.2*2*PI*t);
NotchFilterMedium.In= InputMediumFreq;
/* call function block */
MTFilterNotch(&NotchFilterMedium);
```

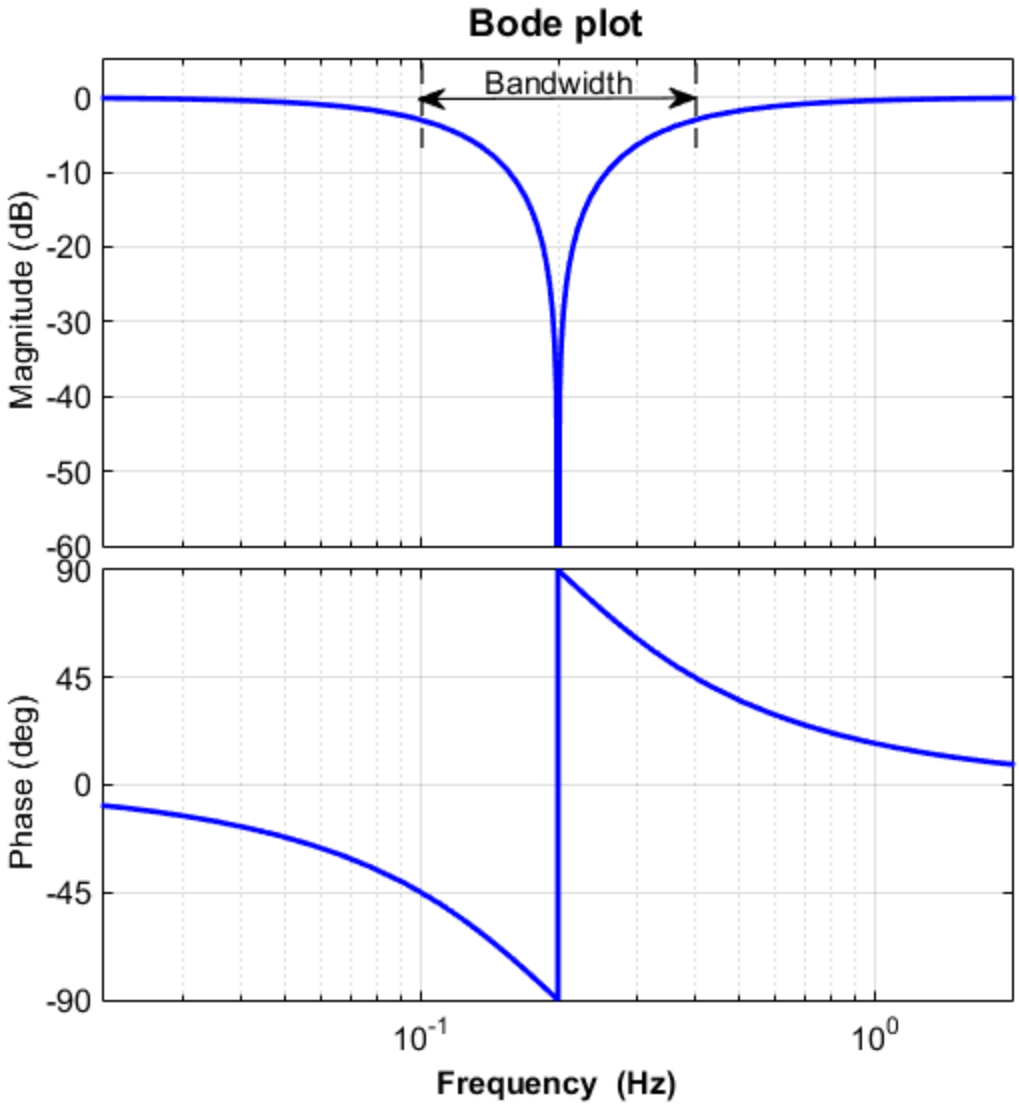
```
OutputMediumFreq= NotchFilterMedium.Out;
InputLowFreq = sin(0.04*2*PI*t);
NotchFilterLow.In= InputLowFreq;
/* call function block */
MTFilterNotch(&NotchFilterLow);
OutputLowFreq= NotchFilterLow.Out;
InputHighFreq = sin(1*2*PI*t);
NotchFilterHigh.In= InputHighFreq;
/* call function block */
MTFilterNotch(&NotchFilterHigh);
OutputHighFreq= NotchFilterHigh.Out;
}
```

4. 测试结果分析

说明： 任务周期 0.1s



- 当输入信号的频率为中心频率为，输出信号幅值被大幅衰减。
- 当输入信号的频率大于或者小于中心频率时，输出信号的幅值基本保持不变，相位也基本保持不变。而且输入信号的频率越远离中心频率，输出信号幅值和相位变化的量越小。
- 时域的响应和频率的传函有对应关系。



5. 说明

•

1.4.5.6. MTFilterBiQuad 介绍_说明文档

1. 功能块功能描述

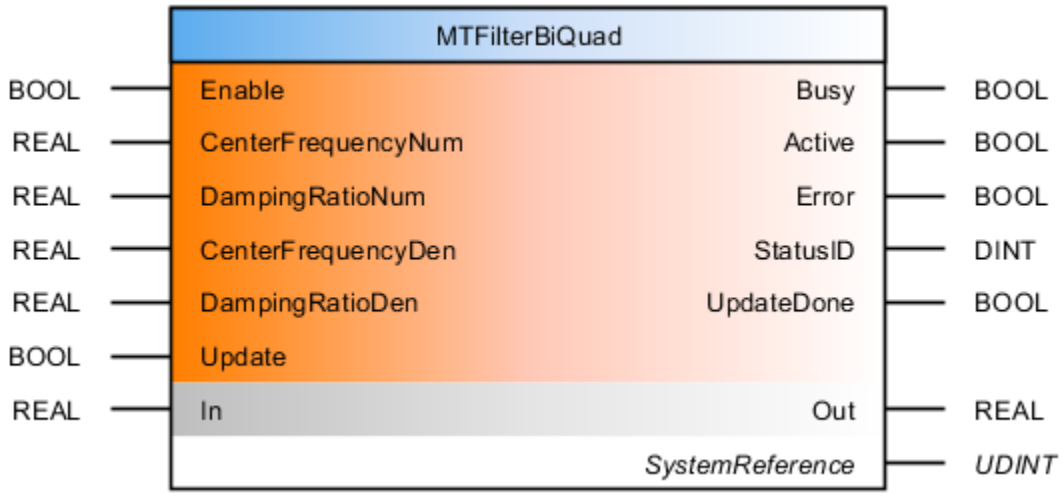
BiQuad 滤波器的传递函数可以表示为

$$\frac{Y(s)}{X(s)} = G(s) = \frac{\frac{s^2}{\omega_n^2} + 2 * d_n * \frac{s}{\omega_n} + 1}{\frac{s^2}{\omega_d^2} + 2 * d_d * \frac{s}{\omega_d} + 1}$$

输出信号 Y 是由输入信号 X 经过该环节后得到的。

该功能块有四个可配置参数，分别为传递函数分子的中心频率、分子的阻尼比、分母的中心频率、分母的阻尼比。

2. 功能块及接口说明



主要接口说明：

I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	CenterFrequencyNum	REAL	中波段频率（分母），单位: [Hz]。 允许设置范围：0 < CenterFrequencyNum < 采样频率 / 2; 采样频率 = 1/CPU 扫描周期;

IN	DampingRatioNum	REAL	阻尼（分母）； 允许设置范围：0 ≤ DampingRatioNum ≤ 1
IN	CenterFrequencyDen	REAL	中心频率（分子），单位: [Hz]。 允许设置范围：0 < CenterFrequencyDen < 采样频率 / 2; 采样频率 = 1/CPU 扫描周期;
IN	DampingRatioDen	REAL	阻尼（分子）； 允许设置范围：0 ≤ DampingRatioNum ≤ 1
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	In	REAL	输入值，滤波前的数值。
OUT	Busy	BOOL	该功能块尚未执行完毕。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	Out	REAL	输出值，经过滤波后的值。
OUT	SystemReference	UDINT	系统参考。

3. AS 编程测试

对该功能块进行了相关测试：

AS 版本	AS4.2.3.159	操作系统版本	H4.23	测试库版本	V2.10.0
需要库	MTFilter				

编写人	董俊清	测试人		审核人	
项目名称	MTFilterBiQuadTest				
任务说明	对输入信号进行 BQ 滤波。				

（1）变量定义

```
VAR
BiQuadFilterNum : MTFilterBiQuad := (0); (*function block MTFilterBiQuad*)
BiQuadFilterDen : MTFilterBiQuad := (0); (*function block MTFilterBiQuad*)
InputNumFreq : REAL := 0.0; (*input signal*)
OutputNumFreq : REAL := 0.0; (*output signal*)
InputDenFreq : REAL := 0.0; (*input signal*)
OutputDenFreq : REAL := 0.0; (*output signal*)
t : REAL := 0.0;
RTinfo : RTInfo := (0);
cycT : REAL := 0.0; (*task cycle*)
END_VAR
```

（2）初始化程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
void _INIT ProgramBandPass_INIT(void)
{
/* BQ Filter */
BiQuadFilterNum.CenterFrequencyNum = 5;
```

```
BiQuadFilterNum.CenterFrequencyDen = 2;
BiQuadFilterNum.DampingRatioNum = 0.07;
BiQuadFilterNum.DampingRatioDen = 0.07;
BiQuadFilterNum.Enable= 1; /* enable function block */
/* BQ Filter */
BiQuadFilterDen.CenterFrequencyNum = 5;
BiQuadFilterDen.CenterFrequencyDen = 2;
BiQuadFilterDen.DampingRatioNum = 0.07;
BiQuadFilterDen.DampingRatioDen = 0.07;
BiQuadFilterDen.Enable= 1; /* enable function block */
/* chosen cycle time: */
RTinfo.enable = 1;
RTInfo(&RTinfo);
cycT=RTinfo.cycle_time/1000000.0;
}
```

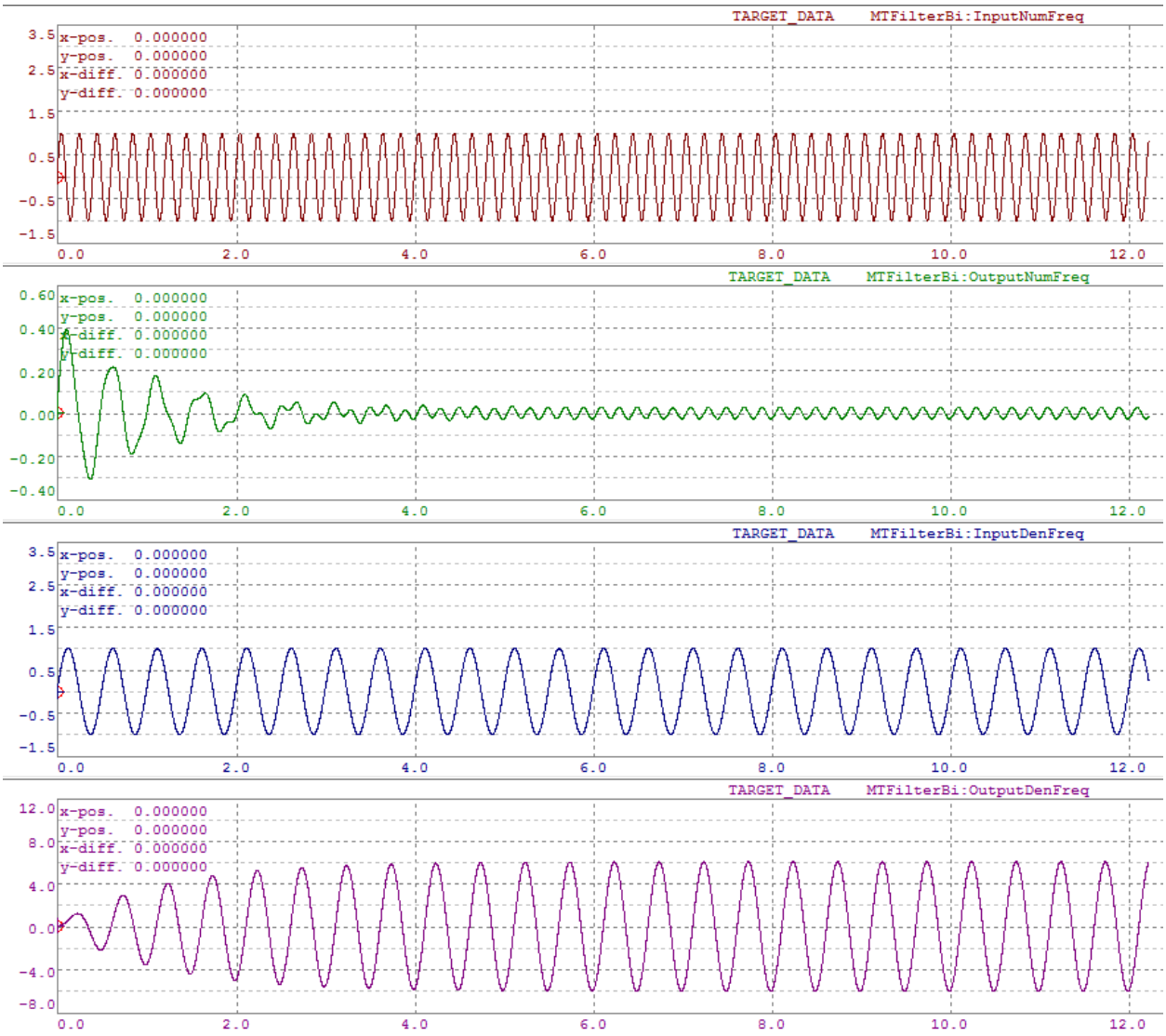
（3）循环程序

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
#include "math.h"
#define PI 3.14159265
void _CYCLIC ProgramBandPass_CYCLIC(void)
{
t += cycT;
InputNumFreq = sin(5*2*PI*t);
```

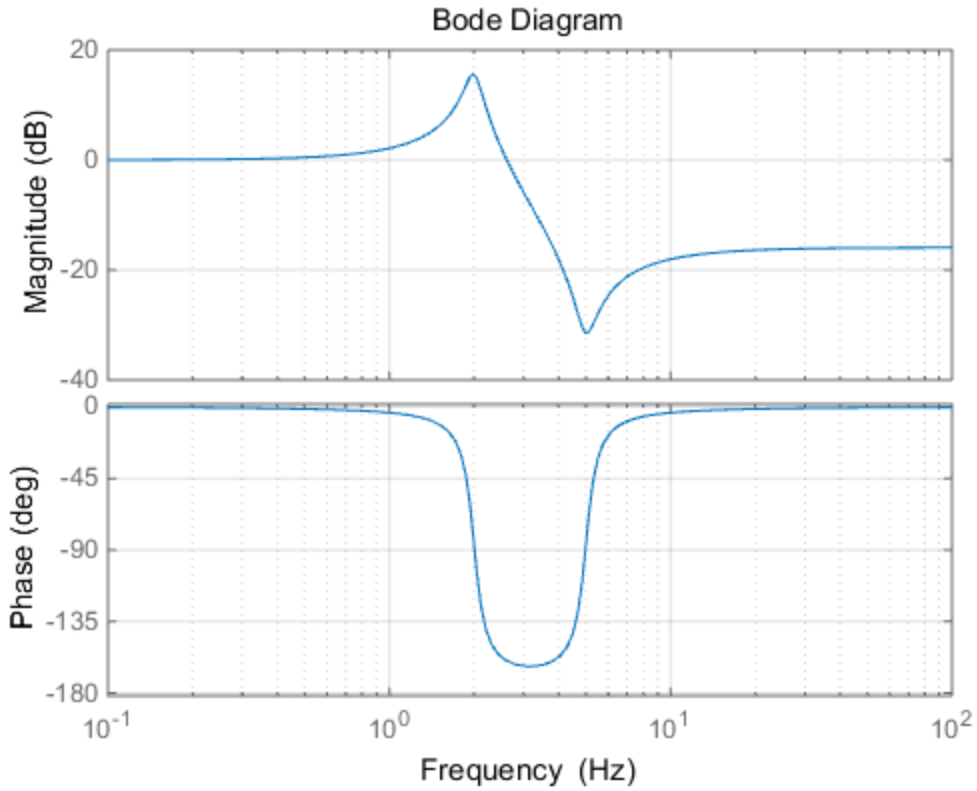
```
BiQuadFilterNum.In= InputNumFreq;
/* call function block */
MTFilterBiQuad(&BiQuadFilterNum);
OutputNumFreq= BiQuadFilterNum.Out;
InputDenFreq = sin(2*2*PI*t);
BiQuadFilterDen.In= InputDenFreq;
/* call function block */
MTFilterBiQuad(&BiQuadFilterDen);
OutputDenFreq= BiQuadFilterDen.Out;
}
```

4. 测试结果分析

说明： 任务周期 1ms



- 当输入信号的频率为 BQ 滤波器共振峰值对应的频率时，输出信号到达稳态时信号幅值会放大，放大倍数与共振峰值的高度有对应关系。输出信号的动态过程将表现为振荡发散的形式，发散速度的快慢与共振峰值对应的阻尼比有对应关系。阻尼比约大，发散越快。
- 当输入信号的频率为 BQ 滤波器反共振峰值对应的频率时，输出信号到达稳态时信号幅值会缩小，缩小倍数与反共振峰值的高度有对应关系。输出信号的动态过程将表现为振荡衰减的形式，衰减速度的快慢与反共振峰值对应的阻尼比有对应关系。阻尼比越大，衰减越快。
- 时域的响应和频率的传函有对应关系。



5. 说明

•

1.5. FIR 数字低通滤波器

1. 基本信息

编写人	汪其锐	**审核人**	刘柏严 添加注释
算法归属	信号处理	**应用对象**	传感器模拟量输入
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V1.00		创建	汪其锐

V1.01		整理修改	刘柏严
V0.0	2023.02.08	整理标准化功能块 V4.15 版本原始内容	王宇鑫

2. 算法简介

FIR 滤波器也称为有限冲击响应滤波器，它用当前和过去输入样值的加权和来形成它的输出, 如下述前馈差分方程所描述的。 FIR 滤波器又称为移动均值滤波器, 因为任何时间点的输出均依赖于包含有最新的 M 个输入样值的一个窗。由于它的响应只依赖于有限个输入, FIR 滤波器对一个离散事件冲激有一个有限长非零响应, 即一个 M 阶 FIR 滤波器对一个冲激的响应在 M 个时钟周期之后为零。

我们常常使用的滑动均值滤波就属于其中一种。

3. 内容

3.1 FIR 滤波器与 IIR 滤波器比较

IIR(Infinite Impulse Response)数字滤波器：又称为“无限冲激响应数字滤波器”，或“递归滤波器”，巴特沃斯即为一种 IIR 滤波器。IIR 滤波器传递函数一般为：

$$H(z)=\frac{b(1)+b(2)z^{-1}+\ldots+b(n+1)z^{-n}}{1+a(2)z^{-1}+\ldots+a(n+1)z^{-n}}$$

有分母存在即需要使用过去的 y,有分子存在表示需要使用过去的 x

FIR(Finite Impulse Response)数字滤波器：又称为“有限冲激响应数字滤波器”。其传递函数一般为：

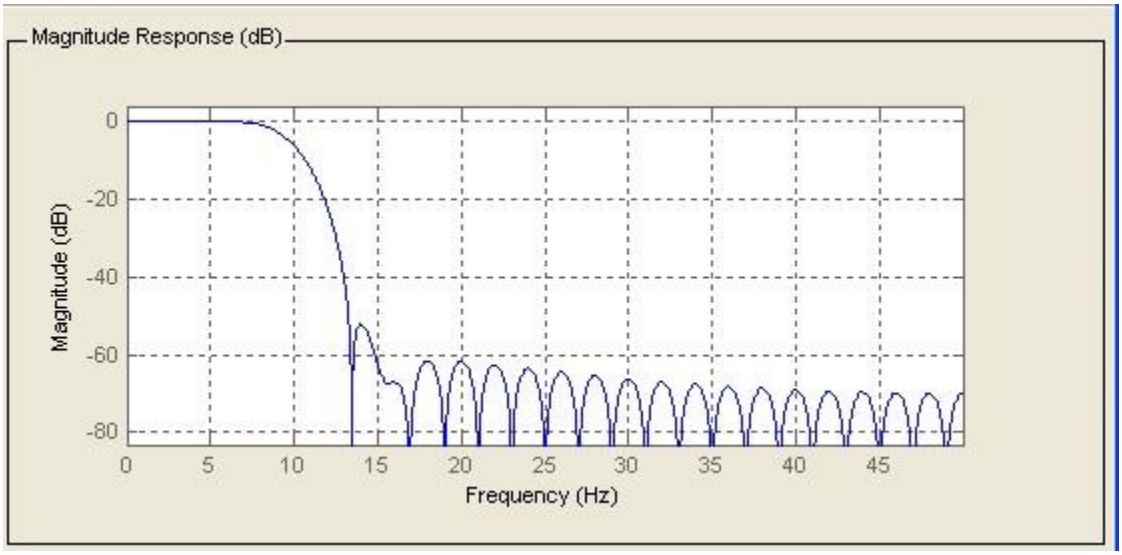
$$H(z)=b(1)+b(2)z^{-1}+\ldots+b(n+1)z^{-n}$$

所以由此看出：IIR 滤波器有回路反馈，传递函数有极点，属于递归型结构。极点必须在单位圆内，否则系统不稳定。这种结构的有限字长影响大，在运算过程中的运算误差有时会引起振荡（即发散），存在系统不稳定问题。

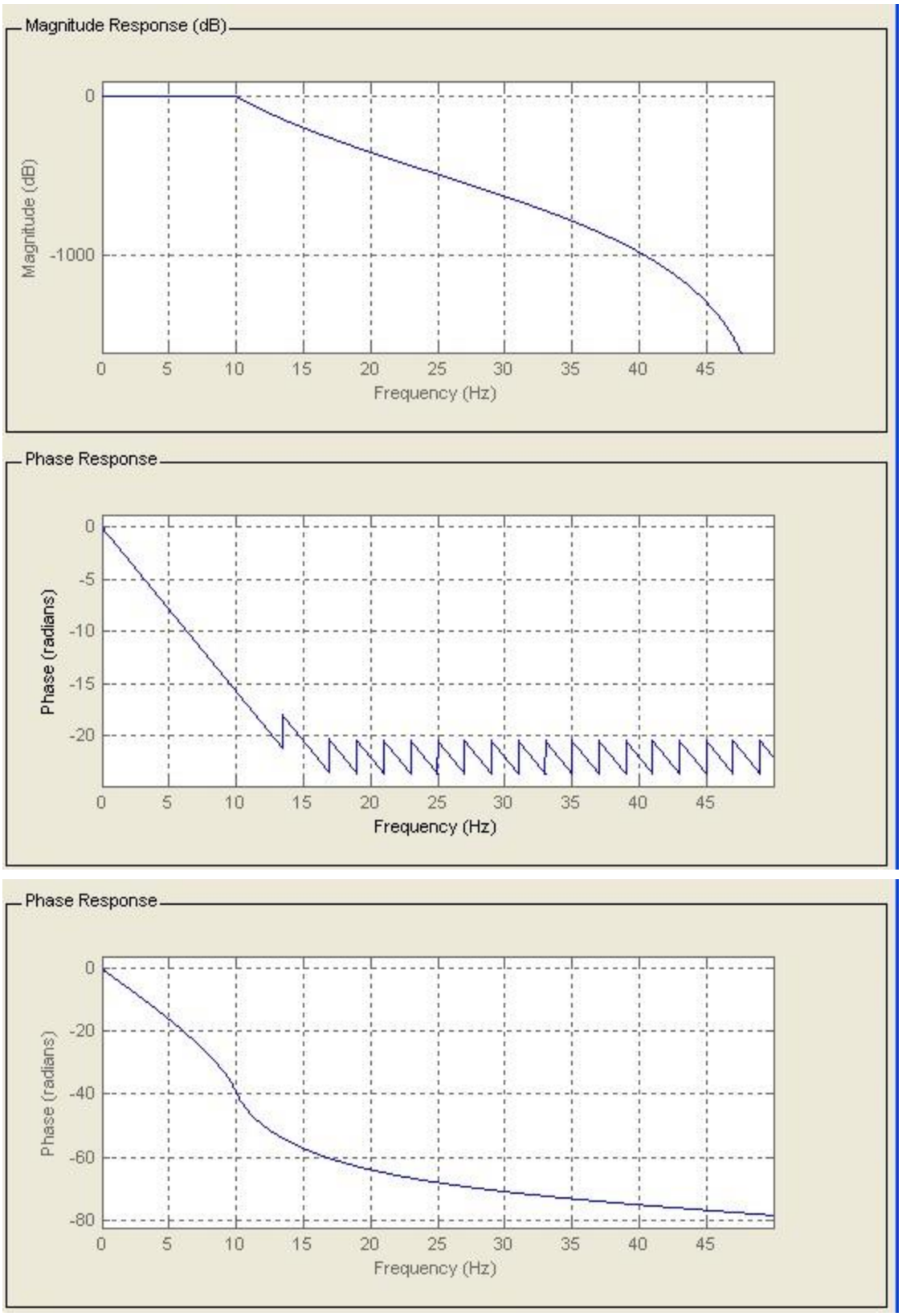
FIR 滤波器无反馈，传递函数无极点(除原点)，属于非递归型结构。所以在实际有限精度运算中不存在稳定性问题，运算误差小。所有系数相加为 1，不会发散。

IIR 滤波器系统函数的极点可以位于单位圆内的任意地方，因此可以用较少阶数获得很高的选择性，但其高效是由相位的非线性为代价，选择性越好的 IIR 滤波器其相位特性越

差。而 FIR 滤波器可以得到严格的线性相位，不过 FIR 滤波器除了原点处的极点外没有可控制的极点，所以要获得与 IIR 滤波器相同的设计指标，FIR 滤波器阶数可能是 IIR 滤波器阶数的 5~10 倍。



FIR 滤波器 50 阶幅频、相频响应图（线性相位）



Butterworth 滤波器 50 阶幅频、相频响应图（非线性相位）

由此可得出：

IIR 滤波器优点：设计方便简单，幅频响应好。

缺点：非线性相位，存在不稳定问题。

FIR 滤波器优点：线性相位，不存在稳定性问题。

缺点：低阶幅频响应不好。

相位问题：任何物理可实现系统都存在延迟，假设一个角频率 w 的正弦信号，通过滤波器时间为 t，则这个信号的输出相位落后原来信号 w*t(卷积)的相位。在实际系统中，一输入信号可以分解为多个正弦信号的叠加，线性相位可以保证这些正弦信号具有相同的延迟，即各频率间相位相对关系没变，从而不失真。非线性相位可能会使各频率延迟不同，造成各频率之间重叠，从而导致信号失真。

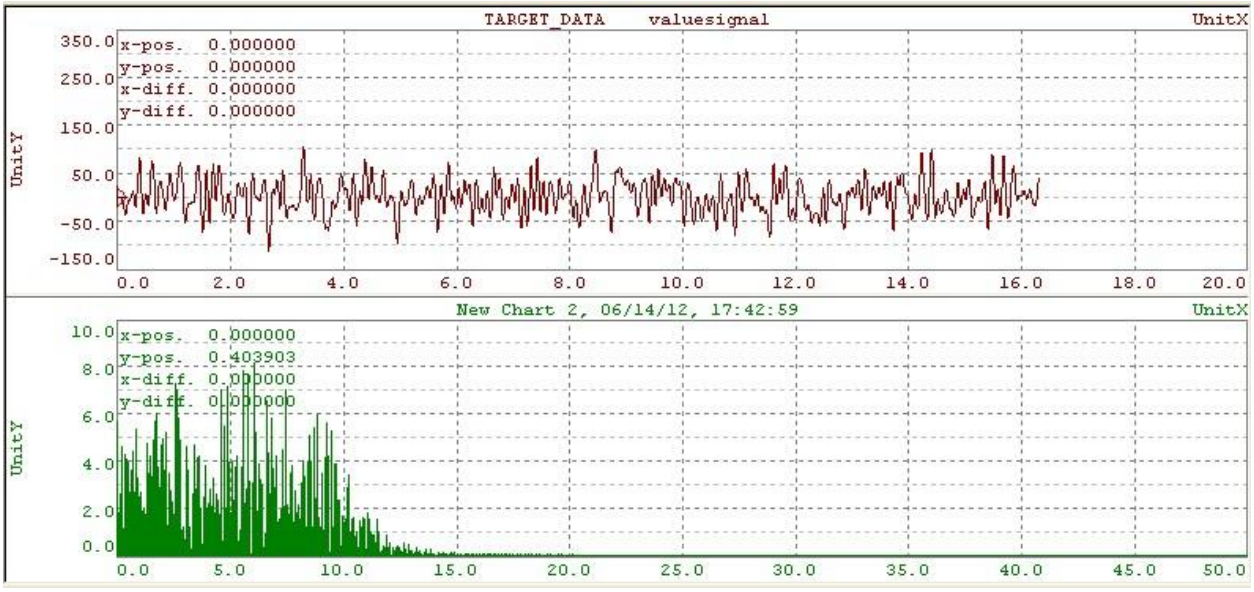
3.2 设计 FIR 滤波器

方法 1：由 matlab 函数得出 h(z)

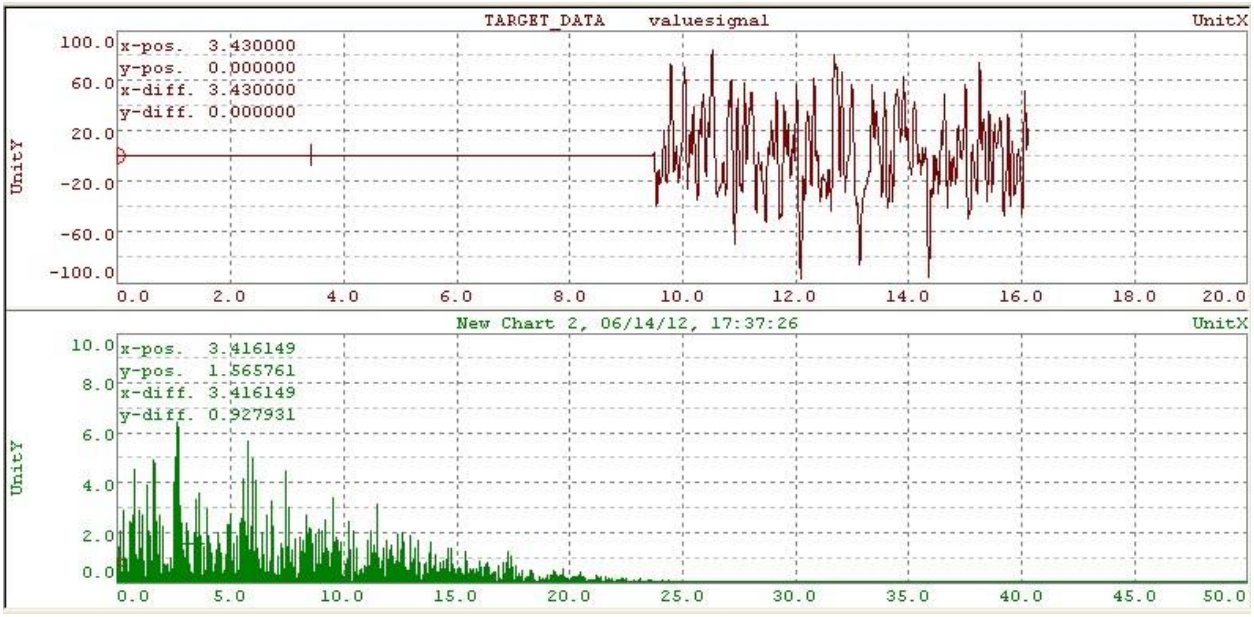
b=fir1(n,2*t*wc,'low') %b 为 h(z)系数，n:滤波器阶数，t:采样时间，wc:截止频率(0<2*t*wc<1)。

若采用白噪声信号作为输入，截止频率为 10hz，采样时间为 10ms，滤波效果如下：

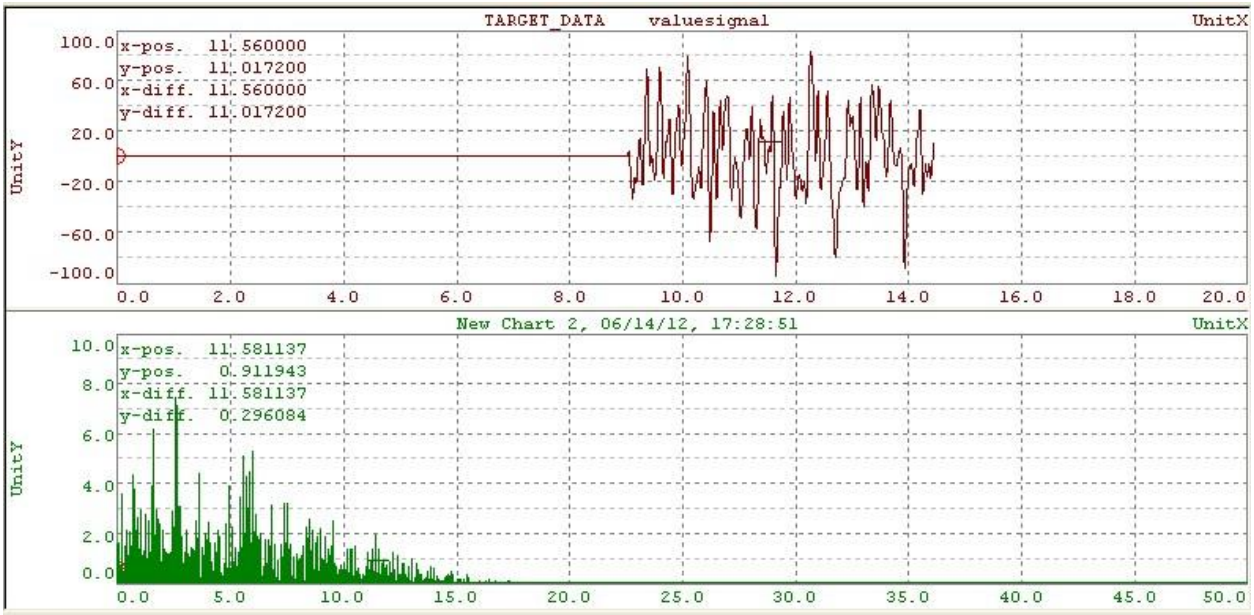
以下图中上图为白噪声信号，全频率。下图为滤波后傅里叶图，横坐标频率（Hz），纵坐标幅值。图中可见高频被过滤。



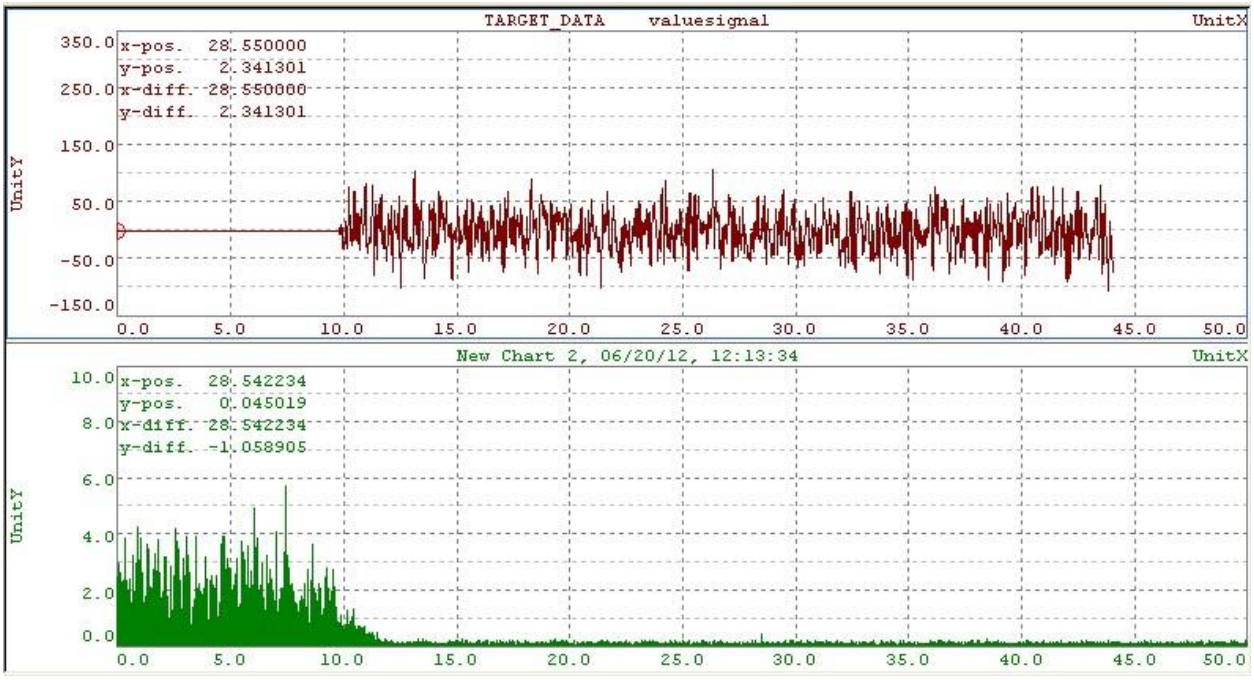
巴特沃斯 10 阶数字滤波器滤波效果



FIR 数字滤波器 10 阶滤波效果



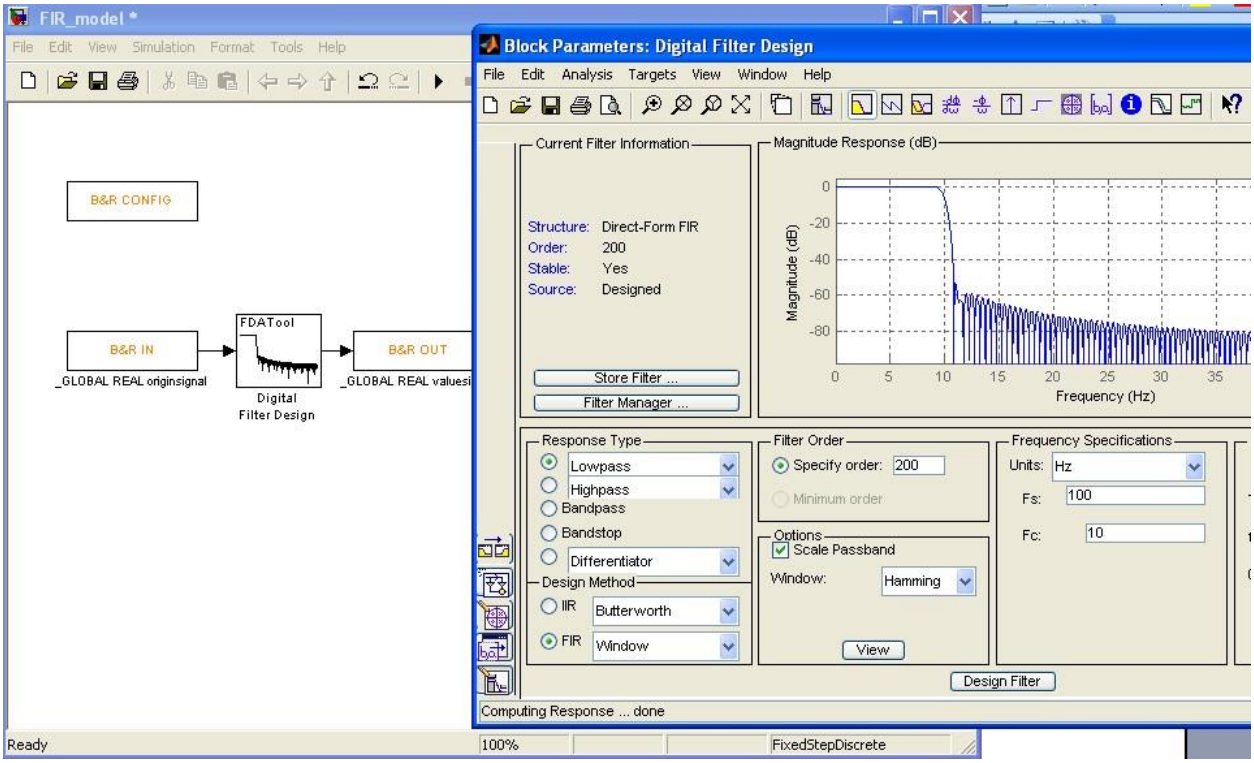
FIR 数字滤波器 20 阶滤波效果



FIR 数字滤波器 50 阶滤波效果

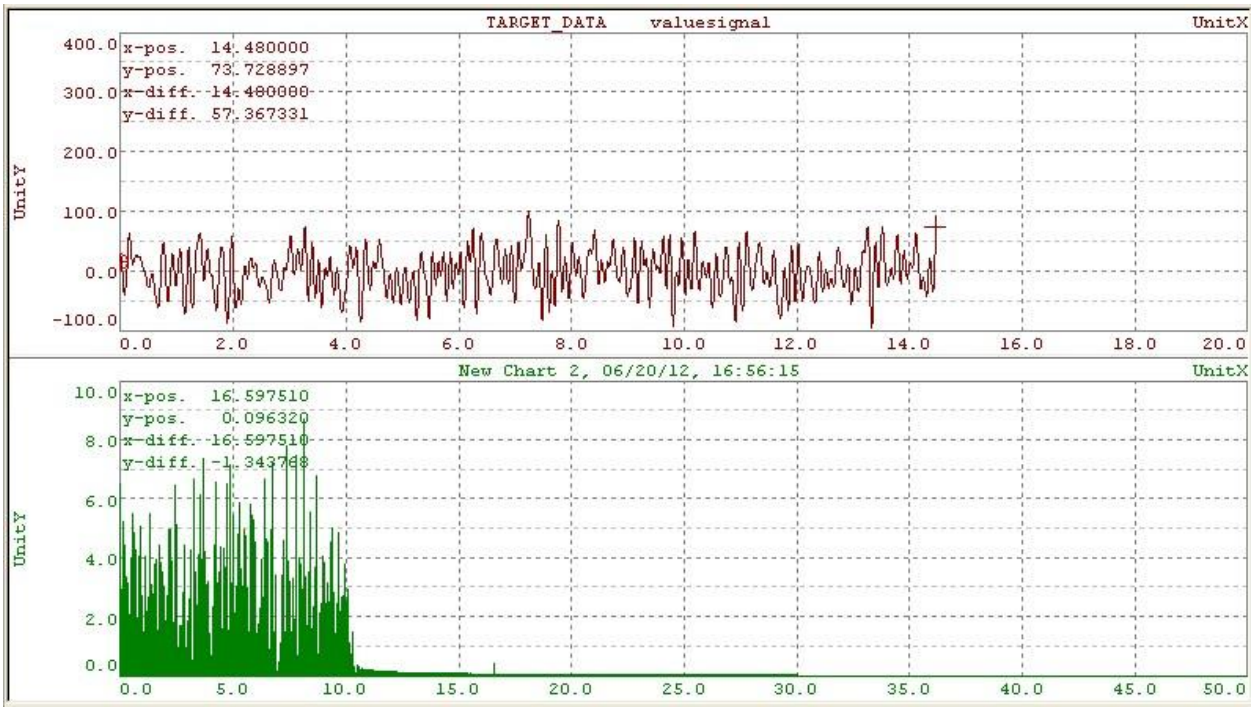
由上图可看出，FIR 滤波器 50 阶效果与 IIR 滤波器 10 阶效果基本相同，低阶 FIR 滤波器效果不理想。FIR 滤波器具有很长的延迟时间!

方法 2：AS 自动生成代码



用 matlab 里面的 FDATool 设计滤波器简单实用，可以设置 IIR 和 FIR 滤波器，并且可以直接查看其幅频相应、相频响应、零极点分布、分子分母系数等。

下图就是 200 阶 FIR 滤波器滤波效果：



FIR 数字滤波器 200 阶滤波效果

可以看出，200 阶 FIR 滤波器效果非常好，但是用编程的话比较麻烦。

4．程序实现/应用举例

MATLAB 公式：b=fir1(n,2*t*wc,'low') %b 为 h(z)系数，n:滤波器阶数=10，t:采样时间=0.01，wc:截止频率(0<2*t*wc<1) 10Hz

在 MATLAB 中输入：b=fir1(10,2*0.01*10, 'low')得到

0 0.0093 0.0476 0.1224 0.2022 0.2370 0.2022 0.1224 0.0476 0.0093

编程

Y= 0.0093*xOld[0]+ 0.0476*xOld[1]+ 0.1224*xOld[2] + 0.2022*xOld[3] + …

所有系数相加等于 1，所以这种方法相当于加权滑动均值滤波。

5．结论及附录

如果对相位要求不高的话，可以选用 IIR 滤波器，如语音处理；对相位敏感的则要选用 FIR 滤波器，如图像数据信号。

部分程序代码如下：

```
clear all;
t=0.01; %采样时间
wc=10; %截止频率
n=50; %阶数
%**************生成各种窗函数*******************************
w_rect=rectwin(n+1)'; % 矩形窗 n+1:滤波器长度
w_hann=hann(n+1)'; % 汉宁窗
w_hamm=hamming(n+1)'; % 海明窗
w_blac=blackman(n+1)'; % 布拉克曼窗
%w_kais=kaiser(n+1,beta)'; % 凯塞窗
%*********************************************************
%b=fir1(n,wn,'ftype',window) % 默认海明窗
b=fir1(n,2*t*wc,'low')
```

1.6. 现代滤波技术

1. 基本信息

编写人	王宇鑫	**审核人**	穆珊珊
算法归属	信号处理	**应用对象**	传感器
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V0.1	2023.02.07	整理标准化功能块 V4.15 版本原始内容	王宇鑫

2. 算法简介

经典滤波器假定信号中有用成分和需要去除的成分各自占有不同的频带，当信号通过滤波器可以将有用成分保留而去除其他成分，但是当信号和噪声的频谱相互重叠时，经典滤波器无法将两者区分。

那么，现代滤波器应运而生，现代滤波器理论研究的主要内容则是从含有噪声的数据记录（又称为时间序列）中估计出信号的某些特征或信号本身。一旦信号被估计出，那么估计出的信号将比原信号的信噪比更高。现代滤波器将信号和噪声都视为随机信号，利用他们的统计特征（例如自相关函数、功率谱等）导出一套最佳的估值算法，然后利用硬件或软件实现。

说白了就是利用统计理论来处理滤波问题。维纳（Wiener）滤波器、卡尔曼（Kalman）滤波器、自适应滤波器都属于现代滤波器。

3. 内容

1.6.1. 状态观测器

1. 基本信息

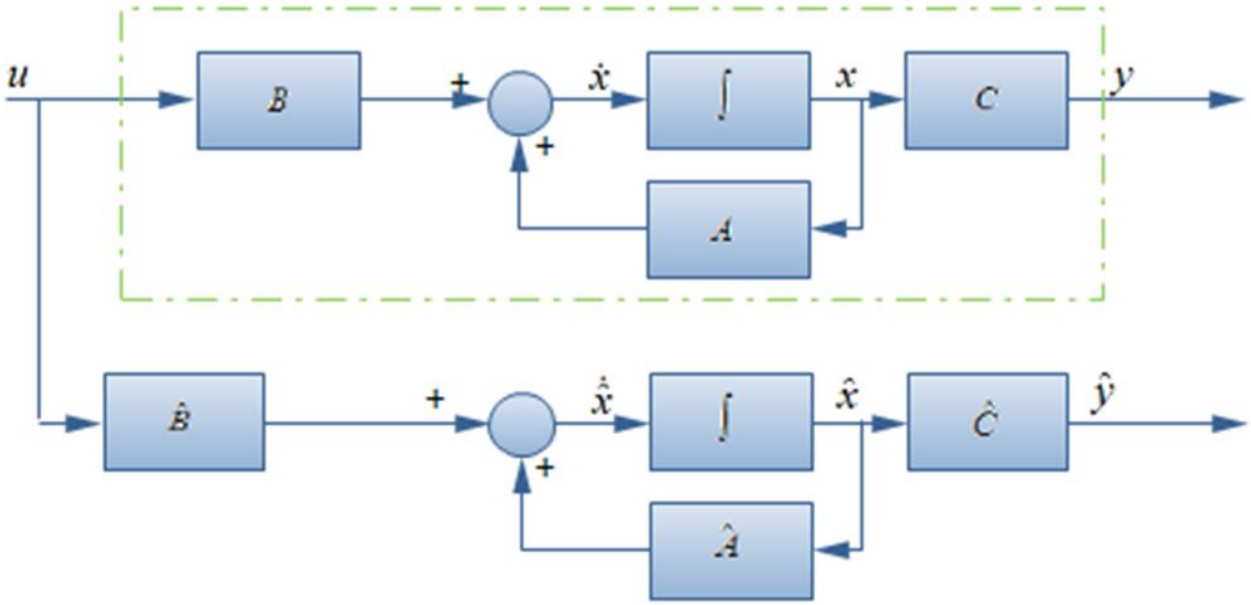
编写人	刘柏严，王宇鑫	**审核人**	
算法归属	信号处理	**应用对象**	
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V0.1	2023.02.08	整理已有相关资料	王宇鑫

2. 算法简介

根据系统的外部变量（输入变量和输出变量）的实测值得出状态变量估计值的一类动态系统，也称为状态重构器。通过系统建模，结合输入输出，使观测出的状态准确快速的逼近真实状态，从而达到抑制扰动和滤波的作用。

3. 内容

利用状态空间法，构造真实系统$\dot{x} = Ax + Bu, y = Cx$，模拟系统$\hat{x}' = A\hat{x} + Bu, \hat{y} = C\hat{x}$，其输出状态为$\hat{x}$



开环：

如果能满足$\lim(x - \hat{x}) = 0$，则$\hat{x}$可以作为系统的最优状态估计。

设$e = x - \hat{x}$，则有$\dot{e} = \dot{x} - \dot{\hat{x}} = A(x - \hat{x}) = Ae$。

如果 A 具有负实部，则状态偏差将收敛于 0，即状态观测值逼近了实际状态，A 矩阵决定了状态收敛的速度。

闭环：

利用输出的估计误差作为反馈，$\Delta y = y - \hat{y}$，

此时状态方程变为：$\dot{x} = Ax + Bu - H(y - \hat{y})$，

状态偏差：$\dot{e} = (A - HC)e$

A-HC 矩阵决定了状态收敛的速度，通过选择 H 矩阵，就可以对状态偏差的收敛速度进行控制。

如何选择反馈矩阵 H：

根据需要的状态收敛的速度，选择期望的极点，进行极点配置，来选择 H 矩阵。

对于一阶系统，期望的极点（特征多项式）为 s+a，极点配置方程： |sI-(A-HC)|=s+a , 获得 H 矩阵。

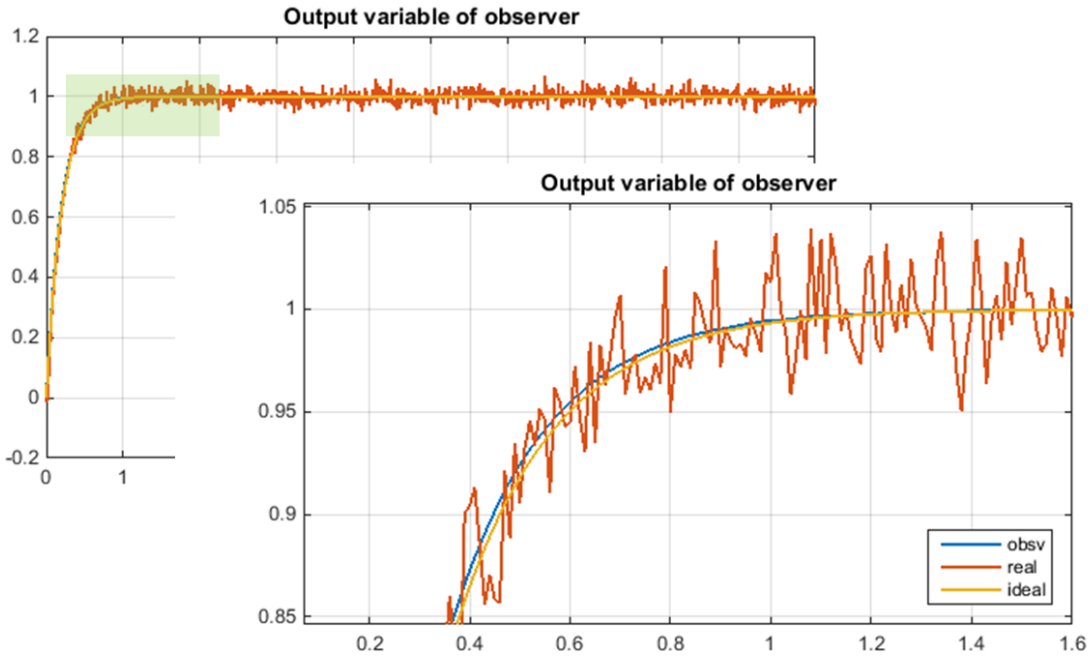
对于二阶系统，期望的极点（特征多项式）为$s_2 + 2\xi\omega s + \omega^2$，极点配置方程： |sI-(A-HC)|=$s_2 + 2\xi\omega s + \omega^2$, 获得 H 矩阵。

系统极点距离虚轴越远，系统响应越快。

4. Matlab 仿真测试

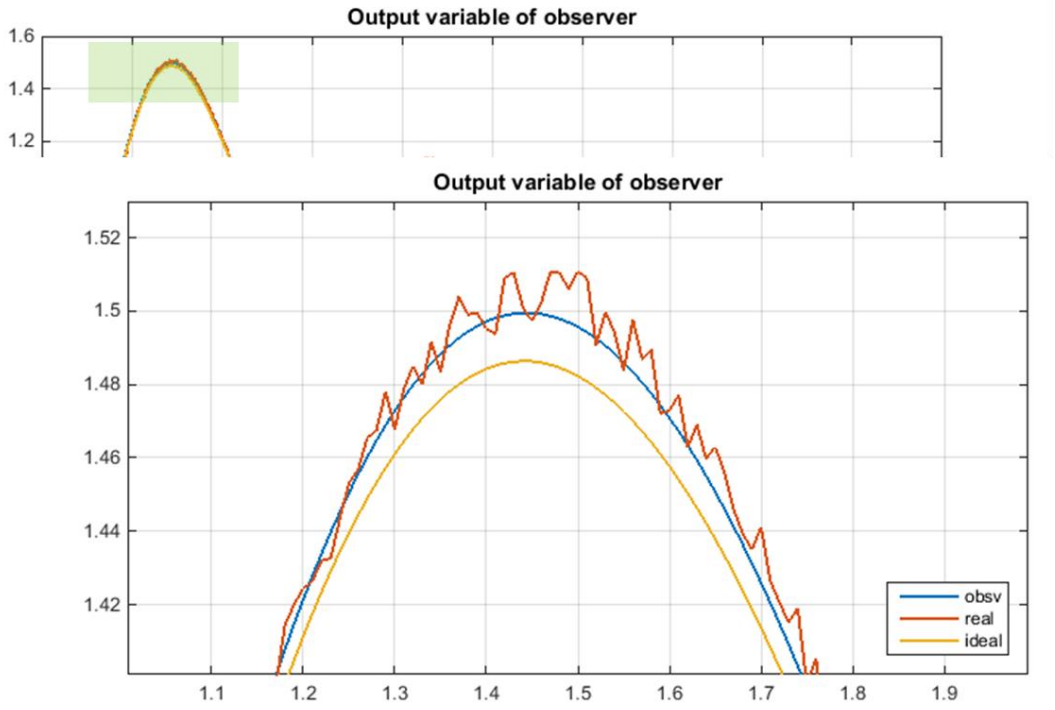
一阶系统：

- 仿真对象： G(s)=1/(Ts+1), T = 0.2.
- 测量噪声： p(noise) ~ N(0,0.02)
- 反馈增益： H = -0.2



二阶系统：

- 仿真对象： G(s)=1/(Ts^2+0.2s+1), T = 0.2.
- 测量噪声： p(noise) ~ N(0,0.02)
- 反馈增益： H = [0.8 0.2]



5. AS 仿真及程序下载

- 仿真对象： G(s)=1/(Ts+1), T = 0.2.
- 测量噪声： p(noise) ~ N(0,0.2)

在 AS 中调用功能块练习状态观测器的使用 ObLibrary

使用 myModelFirstOrder，myObserverFirstOrder。

1.6.2. Kalman 滤波

1. 基本信息

编写人	刘柏严，王宇鑫	**审核人**	
算法归属	信号处理	**应用对象**	
其他说明			
版本信息	**修改日期**	**修改内容**	**修改人**
V0.1	2023.02.08	整理已有相关资料	王宇鑫

2. 算法简介

卡尔曼滤波（Kalman filtering）是一种利用线性系统状态方程，通过系统输入输出观测数据，对系统状态进行最优估计的算法。由于观测数据中包括系统中的噪声和干扰的影响，所以最优估计也可看作是滤波过程。

数据滤波是去除噪声还原真实数据的一种数据处理技术，Kalman 滤波在测量方差已知的情况下能够从一系列存在测量噪声的数据中，估计动态系统的状态。由于它便于计算机编程实现，并能够对现场采集的数据进行实时的更新和处理，Kalman 滤波是目前应用最为广泛的滤波方法，在通信，导航，制导与控制等多领域得到了较好的应用。

贝加莱在 MTFilter 库中提供了 Kalman 滤波器的功能块，见 MTFilterKalman 介绍_说明文档 。

3. 内容

用 Kalman 滤波器，需要首先建立对象的离散状态空间模型，并包含噪声项：

$$x_k = Ax_{k-1} + Bu_{k-1} + \omega_{k-1}$$

$$z_k = Hx_k + v_k$$

ω_k和v_k分别表示过程激励噪声和观测噪声。它们是相互独立，服从高斯分布的白噪声：

$$P(\omega) \sim N(0,Q)$$

$$P(v) \sim N(0,R)$$

定义根据理想模型计算出来的状态量x'_k为第 k 步的先验估计；

定义根据测量值修正过的状态量$\hat{x}_k$ 为第 k 步的后验估计。

根据测量值对状态量的修正可以表示为：

$$\hat{x}_k = x'_k + K(z_k - Hx'_k)$$

分别定义先验估计误差和后验估计误差为：

$$e'_k = x_k - x'_k$$

$$e_k = x_k - \hat{x}_k$$

分别定义先验误差协方差和后验误差协方差为：

$$P'_k = E[e'_k(e'_k)^T]$$

$$P_k = E[e_k e_k^T]$$

Kalman 估计的准则是使得后验估计协方差为最小，即P_k取得极小值，此时的 Kalman 增益 K 即为状态修正系数。

Kalman 滤波器迭代过程

- 向前推算状态变量：$x_k{}' = A\hat{x}_{k-1} + Bu_{k-1}$
- 向前推算误差协方差：$P_k' = AP_{k-1}A^T + Q$
- 计算 Kalman 增益：$K_k = P_k'H^T(HP_K'H^T + R)^{-1}$
- 由观测量更新估计值：$\hat{x}_k = x_k' + K_k(z_k - Hx_k')$
- 更新误差协方差：$P_k = (I - K_kH)P_K'$

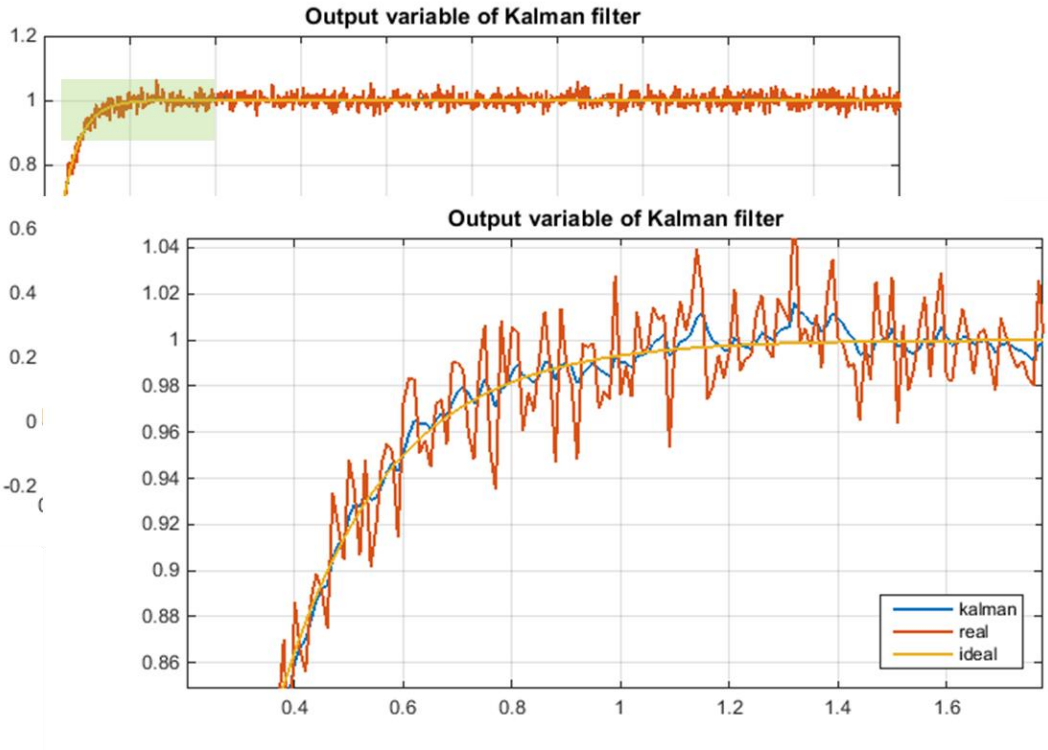
Kalman 滤波器的初始参数： 初始状态，初始协方差。

Kalman 滤波器可调参数： 过程误差的方差、测量误差的方差、误差协方差初值。

4. Matlab 仿真测试

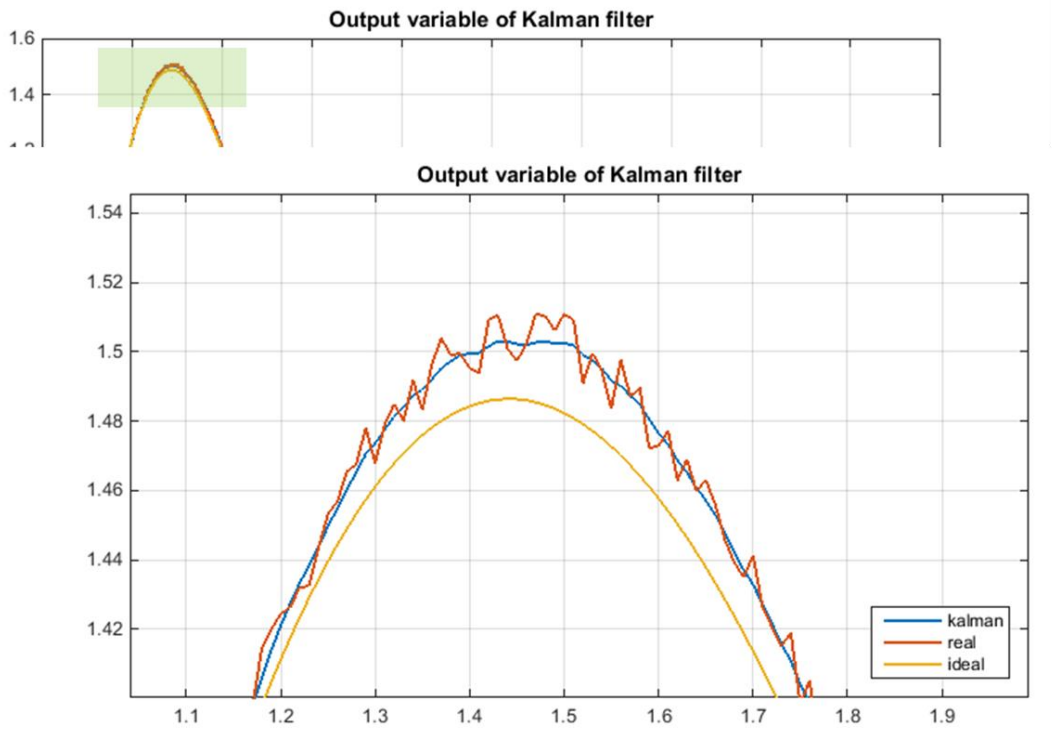
一阶系统：

- 仿真对象： G(s)=1/(Ts+1), T = 0.2.
- 测量噪声： p(measurement_noise) ~ N(0,0.02)
- 过程噪声： p(process_noise) ~ N(0,0.0002)



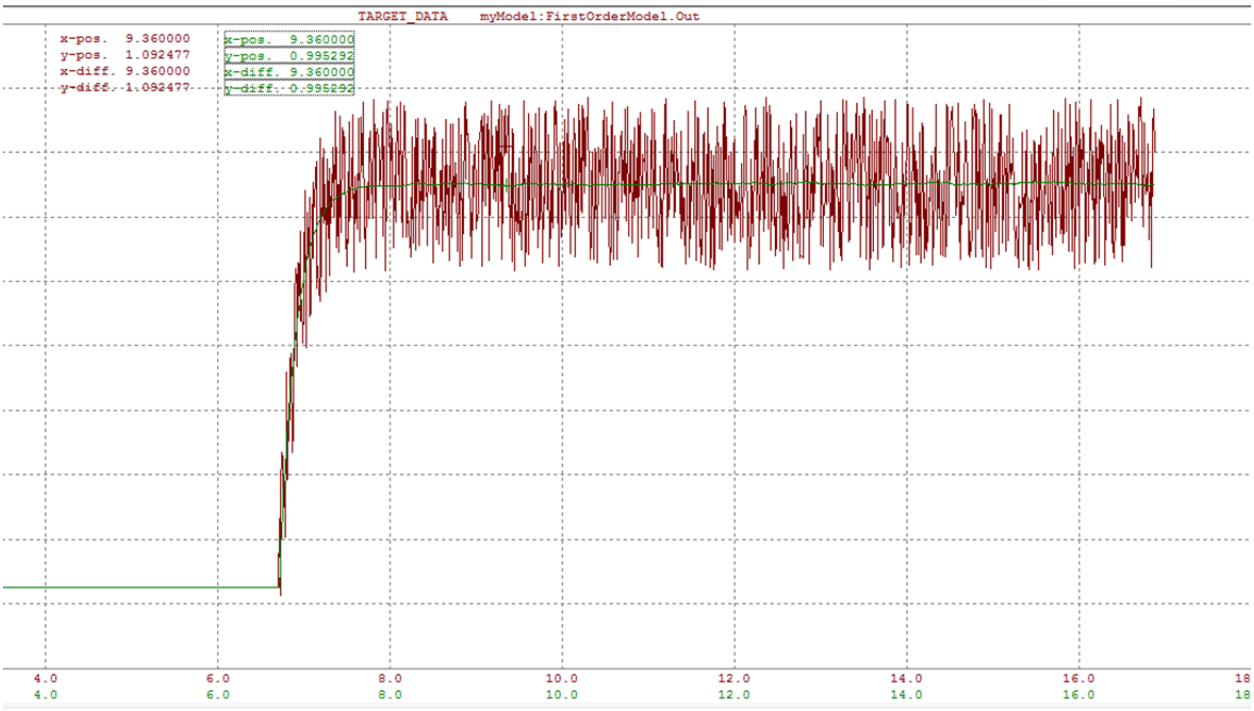
二阶系统：

- 仿真对象： G(s)=1/(Ts^2+0.2s+1), T = 0.2.
- 测量噪声： p(measurement_noise) ~ N(0,0.02)
- 过程噪声： p(process_noise) ~ N(0,0.0002)



5. AS 仿真及程序下载

- 仿真对象： G(s)=1/(Ts+1), T = 0.2.
- 过程噪声： p(process_noise) ~ N(0,0.2)
- 测量噪声： p(measurement_noise) ~ N(0,0.8)



在 AS 中调用功能块练习状态观测器的使用 ObLibrary

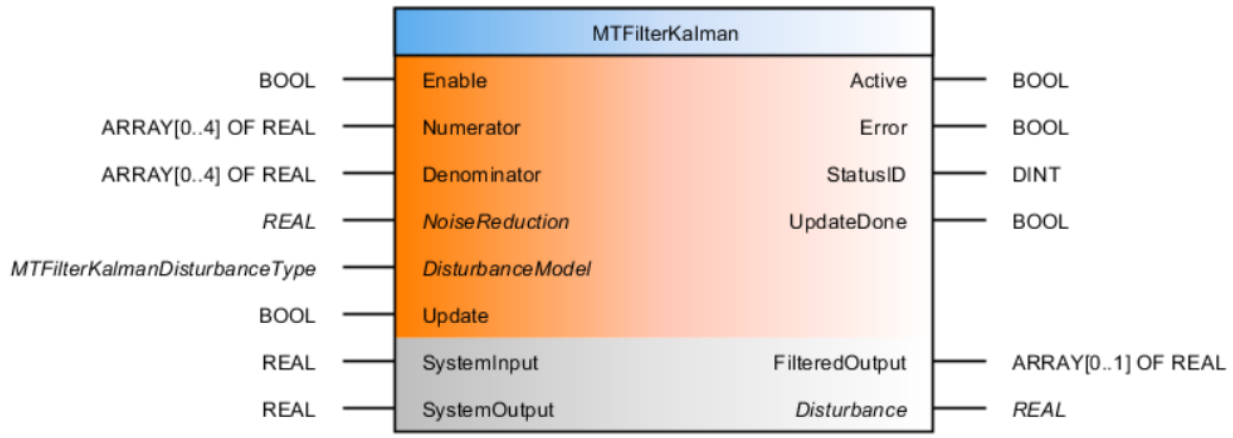
使用 myModelFirstOrder，myKalmanFilterFirstOrder。

1.6.2.1. MTFilterKalman 介绍_说明文档

1. 功能块功能描述

MTFilterKalman 为 Kalman 滤波器的实现功能块，用户需要配置模型对象的传递函数以及噪声抑制系数。

2. 功能块及接口说明



主要接口说明：

I/O	参数名	数据类型	说明
IN	Enable	BOOL	启用/禁用该功能块。
IN	Numerator	ARRAY[0..4] OF REAL	模型传递函数的分母： b0, b1, ..., bn-1, bn
IN	Denominator	ARRAY[0..4] OF REAL	模型传递函数的分子： a0, a1, ..., an-1, an
IN	*NoiseReduction*	REAL	噪音抑制系数。 允许的取值范围：-10.0 ≤ NoiseReduction ≤ 10.0，默认值：2.0。 这个值可以用来影响过滤。这个值越高，应用于指定模型的权重就越高。这将导致对测量噪声更好的过滤。 模型的不准确性在这里也有较高的权重应用。因此，可能出现与测量信号的偏差。
IN	*DisturbanceModel*	MTFilterKalmanDisturbanceType	为扰动模型设置参数，防止外界干扰导致模型不准: • 0，无扰动；

			• 1，固定扰动； • 2，周期性振动，需要同时设定频率。
IN	Update	BOOL	在上升沿更新上面列出的参数。
IN	SystemIn	REAL	系统输入值，控制变量。
IN	SystemOut	REAL	系统观测值，测量信号。
OUT	Active	BOOL	该功能块处于激活状态。
OUT	Error	BOOL	发生了一个错误。
OUT	StatusID	DINT	状态信息。
OUT	UpdateDone	BOOL	更新完成。
OUT	FilteredOut	ARRAY[0..1] OF REAL	• FilteredOutput[0]：滤波后的测量值输出； • FilteredOutput[1]：滤波后输出信号的一阶微分。
OUT	Disturbance	REAL	扰动的估计值。 这个值取决于选择的干扰模型（DisturbanceModel.Mode）。 • mtFILTER_NO_DISTURBANCE：Disturbance = 0； • mtFILTER_CONST_DISTURBANCE：恒定扰动； • mtFILTER_OSCILLATING_DISTURBANCE：振荡扰动值。

3. AS 编程测试

4. 测试结果分析

5. 说明

-

1.7. 样例程序

TBD

Table of Figures

No table of figures entries found.

If your help project contains images with captions, these images will be listed here as figures. (When you customize this template, you may want to delete this explanatory text.)

Keyword Index

No index entries found.

If your help project contains keyword, these keywords will be listed here with page references. (When you customize this template, you may want to delete this explanatory text.)

Some final words here... This is another freely designed page that goes to the end of the published document.