

十一种滤波算法及程序

1 概述

数字滤波方法有很多种，每种方法有其不同的特点和使用范围。从大的范围可分为3类。

1.1 克服大脉冲干扰的数字滤波法

克服由仪器外部环境偶然因素引起的突变性扰动或仪器内部不稳定引起误码等造成的尖脉冲干扰，是仪器数据处理的第一步。通常采用简单的非线性滤波法。

(一).限幅滤波法 (二).中值滤波法

1.2 抑制小幅度高频噪声的平均滤波法

小幅度高频电子噪声：电子器件热噪声、A/D 量化噪声等。通常采用具有低通特性的线性滤波器：算数平均滤波法、加权平均滤波法、滑动加权平均滤波法一阶滞后滤波法等。

(一). 算数平均 (二). 滑动平均 (三). 加权滑动平均 (四)一阶滞后滤波法

1.3 复合滤波法

在实际应用中，有时既要消除大幅度的脉冲干扰，有要做到数据平滑。因此常把前面介绍的两种以上的方法结合起来使用，形成复合滤波。去极值平均滤波算法：先用中值滤波算法滤除采样值中的脉冲性干扰，然后把剩余的各采样值进行平均滤波。连续采样 N 次，剔除其最大值和最小值，再求余下 $N-2$ 个采样的平均值。显然，这种方法既能抑制随机干扰，又能滤除明显的脉冲干扰。

2 十一种通用滤波算法

2.1 限幅滤波法（又称程序判断滤波法）

A、方法：

根据经验判断，确定两次采样允许的最大偏差值（设为 A）

每次检测到新值时判断：

如果本次值与上次值之差 $\leq A$ ，则本次值有效

如果本次值与上次值之差 $> A$ ，则本次值无效，放弃本次值，用上次值代替本次值

B、优点：

能有效克服因偶然因素引起的脉冲干扰

C、缺点

无法抑制那种周期性的干扰

平滑度差

2.2 中位值滤波法

A、方法：

连续采样 N 次（N 取奇数）

把 N 次采样值按大小排列

取中间值为本次有效值

B、优点：

能有效克服因偶然因素引起的波动干扰

对温度、液位的变化缓慢的被测参数有良好的滤波效果

C、缺点：

对流量、速度等快速变化的参数不宜

2.3 算术平均滤波法

A、方法：

连续取 N 个采样值进行算术平均运算

N 值较大时：信号平滑度较高，但灵敏度较低

N 值较小时：信号平滑度较低，但灵敏度较高

N 值的选取：一般流量， $N=12$ ；压力： $N=4$

B、优点：

适用于对一般具有随机干扰的信号进行滤波

这样信号的特点是有一个平均值，信号在某一数值范围附近上下波动

C、缺点：

对于测量速度较慢或要求数据计算速度较快的实时控制不适用

比较浪费 RAM

2.4 递推平均滤波法（又称滑动平均滤波法）

A、方法：

把连续取 N 个采样值看成一个队列

队列的长度固定为 N

每次采样到一个新数据放入队尾,并扔掉原来队首的一次数据.(先进先出原则)

把队列中的 N 个数据进行算术平均运算,就可获得新的滤波结果

N 值的选取：流量， $N=12$ ；压力： $N=4$ ；液面， $N=4\sim 12$ ；温度， $N=1\sim 4$

B、优点：

对周期性干扰有良好的抑制作用，平滑度高

适用于高频振荡的系统

C、缺点：

灵敏度低

对偶然出现的脉冲性干扰的抑制作用较差

不易消除由于脉冲干扰所引起的采样值偏差

不适用于脉冲干扰比较严重的场合

比较浪费 RAM

2.5 中位值平均滤波法（又称防脉冲干扰平均滤波法）

A、方法：

相当于“中位值滤波法”+“算术平均滤波法”

连续采样 N 个数据，去掉一个最大值和一个最小值

然后计算 $N-2$ 个数据的算术平均值

N 值的选取：3~14

B、优点：

融合了两种滤波法的优点

对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差

C、缺点：

测量速度较慢，和算术平均滤波法一样

比较浪费 RAM

2.6 限幅平均滤波法

A、方法：

相当于“限幅滤波法”+“递推平均滤波法”

每次采样到的新数据先进行限幅处理，

再送入队列进行递推平均滤波处理

B、优点：

融合了两种滤波法的优点

对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差

C、缺点：

比较浪费 RAM

2.7 一阶滞后滤波法

A、方法：

取 $a=0\sim 1$

本次滤波结果 = $(1-a) * \text{本次采样值} + a * \text{上次滤波结果}$

B、优点：

对周期性干扰具有良好的抑制作用

适用于波动频率较高的场合

C、缺点：

相位滞后，灵敏度低

滞后程度取决于 a 值大小

不能消除滤波频率高于采样频率的 $1/2$ 的干扰信号

2.8 加权递推平均滤波法

A、方法：

是对递推平均滤波法的改进，即不同时刻的数据加以不同的权
通常是，越接近现时刻的数据，权取得越大。

给予新采样值的权系数越大，则灵敏度越高，但信号平滑度越低

B、优点：

适用于有较大纯滞后时间常数的对象

和采样周期较短的系统

C、缺点：

对于纯滞后时间常数较小，采样周期较长，变化缓慢的信号

不能迅速反应系统当前所受干扰的严重程度，滤波效果差

2.9 消抖滤波法

A、方法：

设置一个滤波计数器

将每次采样值与当前有效值比较：

如果采样值 = 当前有效值，则计数器清零

如果采样值 \neq 当前有效值，则计数器 + 1，并判断计数器是否 \geq 上限 N (溢出)

如果计数器溢出，则将本次值替换当前有效值，并清计数器

B、优点：

对于变化缓慢的被测参数有较好的滤波效果，

可避免在临界值附近控制器的反复开/关跳动或显示器上数值抖动

C、缺点：

对于快速变化的参数不宜

如果在计数器溢出的那一次采样到的值恰好是干扰值,则会将干扰值当作有效值导入系统

2.10 限幅消抖滤波法

A、方法：

相当于“限幅滤波法” + “消抖滤波法”

先限幅,后消抖

B、优点：

继承了“限幅”和“消抖”的优点

改进了“消抖滤波法”中的某些缺陷,避免将干扰值导入系统

C、缺点：

对于快速变化的参数不宜

2.11 IIR 数字滤波器

A. 方法：

确定信号带宽， 滤之。

$$Y(n) = a_1 * Y(n-1) + a_2 * Y(n-2) + \dots + a_k * Y(n-k) + b_0 * X(n) + b_1 * X(n-1) + b_2 * X(n-2) + \dots + b_k * X(n-k)$$

B. 优点： 高通，低通，带通，带阻任意。设计简单(用 matlab)

C. 缺点： 运算量大。

```
//-----
```

软件滤波的 C 程序样例

3 10 种软件滤波方法的示例程序

假定从 8 位 AD 中读取数据（如果是更高位的 AD 可定义数据类型为 int),子程序为 get_ad();

3.1 限副滤波

/* A 值可根据实际情况调整

value 为有效值，new_value 为当前采样值

滤波程序返回有效的实际值 */

```
#define A 10
```

```
char value;
```

```
char filter()
```

```
{
```

```
char new_value;
```

```
new_value = get_ad();
```

```
if ( ( new_value - value > A ) || ( value - new_value > A )
```

```
return value;
```

```
return new_value;
```

```
}
```

3.2 中位值滤波法

/* N 值可根据实际情况调整

排序采用冒泡法*/

#define N 11

char filter()

{

char value_buf[N];

char count,i,j,temp;

for (count=0;count<N;count++)

{

value_buf[count] = get_ad();

delay();

}

for (j=0;j<N-1;j++)

{

for (i=0;i<N-j;i++)

{

if (value_buf>value_buf[i+1])

{

temp = value_buf;

value_buf = value_buf[i+1];

value_buf[i+1] = temp;

}

}

}

return value_buf[(N-1)/2];

}

3.3 算术平均滤波法

```
/*  
*/  
  
#define N 12  
  
char filter()  
{  
    int sum = 0;  
    for ( count=0;count<N;count++)  
    {  
        sum += get_ad();  
        delay();  
    }  
    return (char)(sum/N);  
}
```

3.4 递推平均滤波法（又称滑动平均滤波法）

```
/*  
*/  
  
#define N 12  
  
char value_buf[N];  
char i=0;  
  
char filter()  
{  
    char count;  
    int sum=0;
```

```

value_buf[i++] = get_ad();
if ( i == N ) i = 0;
for ( count=0;count<N,count++)
sum = value_buf[count];
return (char)(sum/N);
}

```

3.5 中位值平均滤波法（又称防脉冲干扰平均滤波法）

```

/*
*/
#define N 12

char filter()
{
char count,i,j;
char value_buf[N];
int sum=0;
for (count=0;count<N;count++)
{
value_buf[count] = get_ad();
delay();
}
for (j=0;j<N-1;j++)
{
for (i=0;i<N-j;i++)
{
if ( value_buf[i]>value_buf[i+1] )
{
temp = value_buf;

```

```

        value_buf = value_buf[i+1];
        value_buf[i+1] = temp;
    }
}
}
for(count=1;count<N-1;count++)
sum += value[count];
return (char)(sum/(N-2));
}

```

3.6 限幅平均滤波法

```

/*
*/

```

略 参考子程序 1、3

3.7 一阶滞后滤波法

/* 为加快程序处理速度假定基数为 100，a=0~100 */

```

#define a 50

```

```

char value;

```

```

char filter()

```

```

{
char new_value;
new_value = get_ad();
return (100-a)*value + a*new_value;
}

```

3.8 加权递推平均滤波法

/* coe 数组为加权系数表，存在程序存储区。*/

```
#define N 12
```

```
char code coe[N] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

```
char code sum_coe = 1+2+3+4+5+6+7+8+9+10+11+12;
```

```
char filter()
```

```
{
```

```
char count;
```

```
char value_buf[N];
```

```
int sum=0;
```

```
for (count=0,count<N;count++)
```

```
{
```

```
value_buf[count] = get_ad();
```

```
delay();
```

```
}
```

```
for (count=0,count<N;count++)
```

```
sum += value_buf[count]*coe[count];
```

```
return (char)(sum/sum_coe);
```

```
}
```

3.9 消抖滤波法

```
#define N 12
```

```

char filter()
{
char count=0;
char new_value;
new_value = get_ad();
while (value !=new_value);
{
count++;
if (count>=N) return new_value;
    delay();
new_value = get_ad();
}
return value;
}

```

3.10 限幅消抖滤波法

```

/*
*/

```

略 参考子程序 1、9

3.11 IIR 滤波例子

```

int BandpassFilter4(int InputAD4)
{
int ReturnValue;
int ii;
RESLO=0;
RESHI=0;

```

```

MACS=*PdelIn;
OP2=1068; //FilterCoeff4[4];
MACS=*(PdelIn+1);
OP2=8; //FilterCoeff4[3];
MACS=*(PdelIn+2);
OP2=-2001; //FilterCoeff4[2];
MACS=*(PdelIn+3);
OP2=8; //FilterCoeff4[1];
MACS=InputAD4;
OP2=1068; //FilterCoeff4[0];
MACS=*PdelOu;
OP2=-7190; //FilterCoeff4[8];
MACS=*(PdelOu+1);
OP2=-1973; //FilterCoeff4[7];
MACS=*(PdelOu+2);
OP2=-19578; //FilterCoeff4[6];
MACS=*(PdelOu+3);
OP2=-3047; //FilterCoeff4[5];
*p=RESLO;
*(p+1)=RESHI;
mytestmul<=<=2;
ReturnValue=*(p+1);
for (ii=0;ii<3;ii++)
{
DelayInput[ii]=DelayInput[ii+1];
DelayOutput[ii]=DelayOutput[ii+1];
}
DelayInput[3]=InputAD4;
DelayOutput[3]=ReturnValue;

```

```
// if (ReturnValue<0)
// {
// ReturnValue=-ReturnValue;
// }
return ReturnValue;
}
```