【技术】PID 控制器开发笔记之十二：模糊 PID 控制器的实现

原创： 尹家军　木南创智　2018-11-10

在现实控制中，被控系统并非是线性时不变的，往往需要动态调整 PID 的参数，而模糊控制正好能够满足这一需求，所以在接下来的这一节我们将讨论模糊 PID 控制器的相关问题。模糊 PID 控制器是将模糊算法与 PID 控制参数的自整定相结合的一种控制算法。可以说是模糊算法在 PID 参数整定上的应用。

1、模糊算法的原理

模糊算法是一种基于智能推理的算法，虽然称之为模糊算法其实并不模糊，实际上是一种逐步求精的思想。一个模糊控制器主要是由模糊化，模糊推理机和精确化三个功能模块和知识库（包括数据库和规则库）构成的。在此我们近讨论模糊控制的几个主要问题。

1.1、输入量的量化

输入数据都是精确的，要实现模糊算法需要现对其实现量化。所谓量化就是通过量化函数将输入量投射到一定的数字级别，一般都是相对于 0 对称的数字区间。具体投射到怎样的区间根据实际情况而定，因为这会直接影响到计算的精度。

1.2、模糊化

模糊化是模糊算法非常重要的一步，首先确定对应各语言变量的模糊子集，然后根据量化的结果，我们就可以判断该输入所属的集合并计算出对应的隶属度。计算隶属度的方法有很多，最常用的是使用三角形隶属度函数或梯形隶属度函数等来计算获得。

1.3、规则库

规则库是基于控制量的模糊化而的味道的，是实现模糊推理的基础，很大程度上依赖于经验来完成。规则库的表现形式可以有多种，具体实现的形式根据我们实现的方便。

1.4、推理机

推理决策才是模糊控制的核心，它利用知识库中的信息和模糊运算方式，模拟人的推理决策的思想方法，在一定的输入条件下激活相应的控制规则给出适当的模糊控制输出。

1.5、精确化

我们通过模糊推理，得到一系列的模糊表达，需要进行解模糊操作才能得到紧缺的数据。常用的解模糊方法有：

最大隶属度法——计算简单，适用于控制要求不高场合。

重心法——输出更平滑，但计算难度大

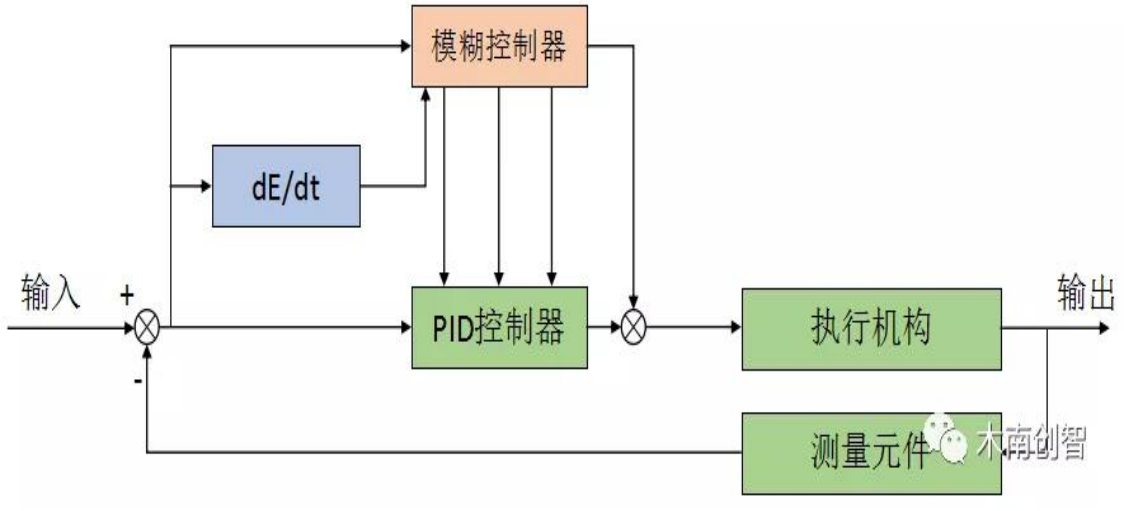
加权平均法———般在工业上应用最广泛

1.6、工程量化

系统控制输出是一个精确的数，但不是可以直接用于对象控制的物理量，所以在最后还要按照我们的需要进行转换。比如对应 PID 的参数则可进行必要的转换和修正在输出给 PID 控制器。

2、模糊 PID 算法的设计

前面简单的描述了模糊算法的基本原理，接下来我们将讨论如何将其应用于 PID 控制当中。所谓模糊 PID 控制是以偏差 e 及偏差的变化 ec 为输入，利用模糊控制规则在线对 PID 参数进行调整，以满足不同的偏差 e 和偏差的增量 ec 对 PID 参数的不同要求。其结构图如下：



2.1、输入值的模糊化

输入值的模糊化就是将用于计算的输入对应到标准化的数值区间，并根据量化结果和模糊化子集得到该输入对子集的隶属度。我们在使用偏差 e 和偏差增量 ec 作为输入实现控制参数调整则需要对 e 和 ec 进行模糊化。

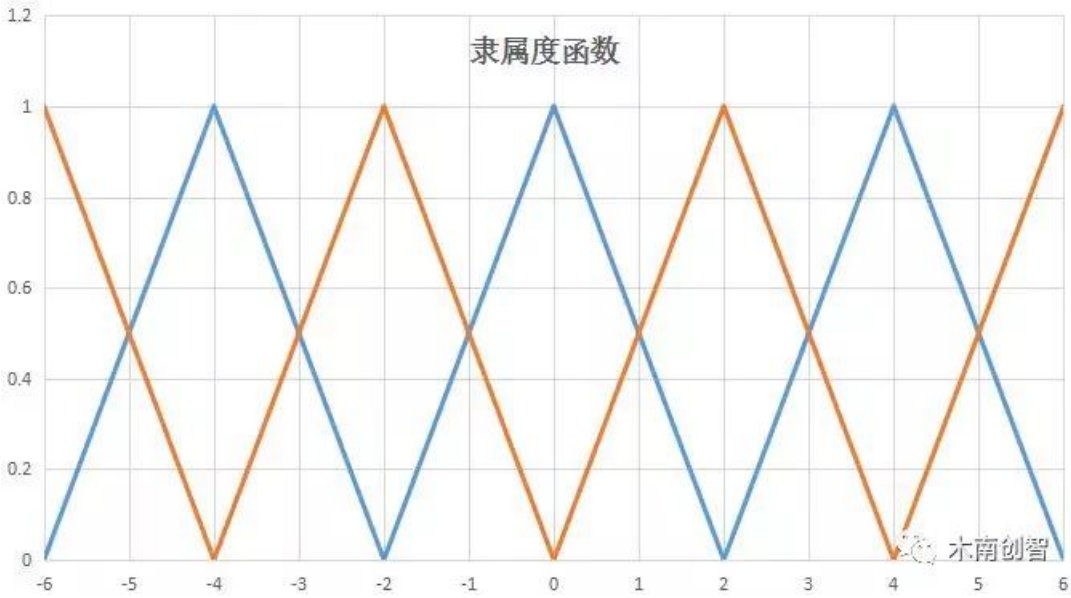
首先，我们确定 e 和 ec 的模糊子集，对于 PID 控制我们选则：负大[NB]、负中[NM]、负小[NS]、零[ZO]、正小[PS]、正中[PM]、正大[PB]等 7 个语言变量就能够有足够精度表达其模糊子集。所以我们定义 e 和 ec 的模糊子集均为{NB，NM，NS，ZO，PS，PM，PB}。

确定了模糊子集，我们怎么将 e 和 ec 的具体值和模糊集对应上呢？我们需要引入量化函数。要确定量化函数，我们先引入 e 和 ec 模糊集对应的论域，定义为{-6，-5，-4，-3，-2，-1，0，1，2，3，4，5，6}。对于任何一个物理量测量信号都有一个量程范围，我们记为 Vmax 和 Vmin，和自然在 PID 调节时设定值的范围预期相同，所以偏差 e 的范围就是 Vmin-Vmax 到 Vmax-Vmin 的范围内，而偏差的增量范围则是其两倍。这里我们采用线性方式量化，则其函数关系为：

$$f(e)=\frac{6*e}{V\max-V\min} \qquad f(ec)=\frac{6*ec}{2(V\max-V\min)}$$

利用上述的量化函数就可以将 e 和 ec 量化，我们可以采用如 4 舍 5 入的方式获取确定的模糊子集。但考虑到 e 和 ec 的变化是连续变化的，4 舍 5 入对控制精度可能存在影响，所以我们引入隶属度来实现这一过程。

最后我们确定 e 和 ec 在模糊子集上的隶属度。隶属度是一个介于 0 和 1 之间的值，用以描述对应一个输入属于某一个模糊自己的程度。一般我们描述成隶属度函数，可采用的隶属度函数很多，我们在次采用线性的隶属度函数，或者称为三角隶属度函数，其函数关系如下：



如果我们量化后的结果是 1，那么属于 ZO 的隶属度为 0.5，同样属于 PS 的隶属度也是 0.5。至此，模糊化全部完成。

2.2、建立模糊规则表

前面我们简述了输入的模糊化，但模糊推理才是模糊控制的根本。为了实现模糊推理首先我们要建立模糊推理的规则库或者称知识库，然后建立推理机进行推理。

首先，我们来建立模糊规则库，在这里我们要对 Kp、Ki 和 Kd 三个参数进行调整，所以要建立这 3 个变量的模糊规则库。

（1）、Kp 模糊规则设计

在 PID 控制器中，Kp 值的选取决定于系统的响应速度。增大 Kp 能提高响应速度，减小稳态偏差；但是，Kp 值过大会产生较大的超调，甚至使系统不稳定减小 Kp 可以减小超调，提高稳定性，但 Kp 过小会减慢响应速度，延长调节时间。因此，调节初期应适当取较大的 Kp 值以提高响应速度，而在调节中期，Kp 则取较小值，以使系统具有较小的超调并保证一定的响应速度；而在调节过程后期再将 Kp 值调到较大值来减小静差，提高控制精度。基于上述描述我们定义 Kp 的模糊规则如下：

EC

E / ΔKp	NB负大	NM负中	NS负小	ZO零	PS正小	PM正中	PB正大
NB 负大	PB正大	PB正大	PM正中	PM正中	PS正小	ZO零	ZO零
NM 负中	PB正大	PB正大	PM正中	PS正小	PS正小	ZO零	NS 负小
NS 负小	PM正中	PM正中	PM正中	PS正小	ZO零	NS 负小	NS 负小
ZO 零	PM正中	PM正中	PS正小	ZO零	NS 负小	NM 负中	NM 负中
PS 正小	PS正小	PS正小	ZO零	NS 负小	NS 负小	NM 负中	NM 负中
PM 正中	PS正小	ZO零	NS 负小	NM 负中	NM 负中	NM 负中	NB 负大
PB 正大	ZO零	ZO零	NM 负中	NM 负中	NM 负中	NB 负大	NB 负大

（2）、Ki 模糊规则设计

在系统控制中，积分控制主要是用来消除系统的稳态偏差。由于某些原因(如饱和非线性等)，积分过程有可能在调节过程的初期产生积分饱和，从而引起调节过程的较大超调。因此，在调节过程的初期，为防止积分饱和，其积分作用应当弱一些，甚至可以取零；而在调节中期，为了避免影响稳定性，其积分作用应该比较适中；最后在过程的后期，则应增强积分作用，

以减小调节静差。依据以上分析，我们制定的 Ki 模糊规则如下：

EC

E / ΔKi	NB负大	NM负中	NS负小	ZO零	PS正小	PM正中	PB正大
NB 负大	NB 负大	NB 负大	NM 负中	NM 负中	NS 负小	ZO零	ZO零
NM 负中	NB 负大	NB 负大	NM 负中	NS 负小	NS 负小	ZO零	ZO零
NS 负小	NB 负大	NM 负中	NS 负小	NS 负小	ZO零	PS正小	PS正小
ZO 零	NM 负中	NM 负中	NS 负小	ZO零	PS正小	PM正中	PM正中
PS 正小	NM 负中	NS 负小	ZO零	PS正小	PS正小	PM正中	PB正大
PM 正中	ZO零	ZO零	PS正小	PS正小	PM正中	PB正大	PB正大
PB 正大	ZO零	ZO零	PS正小	PM正中	PM正中	PB正大	PB正大

（3）、Kd 模糊规则设计

微分环节的调整主要是针对大惯性过程引入的，微分环节系数的作用在于改变系统的动态特性。系统的微分环节系数能反映信号变化的趋势，并能在偏差信号变化太大之前，在系统中引入一个有效的早期修正信号，从而加快响应速度，减少调整时间，消除振荡．最终改变系统的动态性能。因此，Kd 值的选取对调节动态特性影响很大。Kd 值过大，调节过程制动就会超前，致使调节时间过长；Kd 值过小，调节过程制动就会落后，从而导致超调增加。根据实际过程经验，在调节初期，应加大微分作用，这样可得到较小甚至避免超调；而在中期，由于调节特性对 Kd 值的变化比较敏感，因此，Kd 值应适当小一些并应保持固定不变；然后在调节后期，Kd 值应减小，以减小被控过程的制动作用，进而补偿在调节过程初期由于 Kd 值较大所造成的调节过程的时间延长。依据以上分析，我们制定 Kd 的模糊规则如下：

EC

E / ΔKd	NB负大	NM负中	NS负小	ZO零	PS正小	PM正中	PB正大
NB 负大	PS正小	NS 负小	NB 负大	NB 负大	NB 负大	NM 负中	PS正小
NM 负中	PS正小	NS 负小	NB 负大	NM 负中	NM 负中	NS 负小	ZO零
NS 负小	ZO零	NS 负小	NM 负中	NM 负中	NS 负小	NS 负小	ZO零
ZO 零	ZO零	NS 负小	NS 负小	NS 负小	NS 负小	NS 负小	ZO零
PS 正小	ZO零	ZO零	ZO零	ZO零	ZO零	ZO零	ZO零
PM 正中	PB正大	NS 负小	PS正小	PS正小	PS正小	PS正小	PB正大
PB 正大	PB正大	PM正中	PM正中	PM正中	PS正小	PS正小	PB正大

接下来，根据偏差 E 和偏差增量 EC 模糊化的结果以及规则库推理出ΔKp、ΔKi、ΔKd 对应的模糊子集。由于前面我们设计的是采用隶属度函数来定义输入输出量在模糊子集的隶属度，所以推理出来的ΔKp、ΔKi、ΔKd 的模糊子集通常是一个由模糊变量组成的矩阵。而输入量 E 和 EC 则是一个由模糊变量组成的向量。

最后，我们需要明确不同的模糊变量所对应的量化数据。这个量化数据与物理量的对应则根据具体的不同对象是完全不一样的。

2.3、解模糊处理

对于求得的目标对象，我们还需要将其你模糊处理以使其与具体的物理量相对应。在模糊 PID 调解中，我们需要的是 Kp，Ki 和 Kd，所以我们需要根据模糊推理的结果得到我们想要的 Kp，Ki 和 Kd 值。

我们前面设计了三角隶属度函数，并采用相同的量化目标即论域{-6，6}，所以在某一时刻，输入输出所处的模糊变量的隶属度是相同的，基于这一基础，我们采用重心法计算各输出量的量化值。其公式如下：

$$Vo = \frac{\sum_{i=0}^{n} M_i * F_i}{\sum_{i=0}^{n} M_i}$$

其中M为隶属度，F为模糊量化值

其实因为我们采用的隶属度函数的特性，在任何方向的计算隶属度的和均为 1，所以分母可以省略。于是每一个对象的计算实际上就是矩阵操作，公式如下：

$$K = [M_{e1} \quad M_{e2}] \begin{bmatrix} F_a & F_b \\ F_c & F_d \end{bmatrix} [M_{ec1} \quad M_{ec2}]^T$$

如果使用的是量化值，则还需要转为实际值，关于这一点直接使用物理量值也是没问题的，怎么处理根据实际需要确定。得到增量后，我们也可以引入系数来放大和缩小 Kp，Ki 和 Kd 变化量，具体实现公式如下：

$$K(n) = K(n-1) + \Delta K * \alpha$$

其中ΔK 为我们所计算得到的值，而α为系数，设定增量对最终只的影响。

3、模糊 PID 算法实现

前面我们描述了算法的全过程，接下来我们编码实现之。首先我们依然需要定义一个模糊 PID 控制器的结构对象。

```
/*定义结构体和公用体*/
typedef struct
{
   float setpoint;                  /*设定值*/
   float kp;                        /*比例系数*/
   float ki;                        /*积分系数*/
   float kd;                        /*微分系数*/
   float lasterror;                 /*前一拍偏差*/
   float preerror;                  /*前两拍偏差*/
   float deadband;                  /*死区*/
   float output;                    /*输出值*/
   float result;                    /*物理量输出值*/
   float maximum;                   /*输出值的上限*/
   float minimum;                   /*输出值的下限*/

   float maxdKp;                    /*Kp 增量的最大限值*/
   float mindKp;                    /*Kp 增量的最小限值*/
   float qKp;                       /*Kp 增量的影响系数*/
   float maxdKi;                    /*Ki 增量的最大限值*/
```

```
  float mindKi;                 /*Ki 增量的最小限值*/
  float qKi;                    /*Ki 增量的影响系数*/
  float maxdKd;                 /*Kd 增量的最大限值*/
  float mindKd;                 /*Kd 增量的最小限值*/
  float qKd;                    /*Kd 增量的影响系数*/
}FUZZYPID;
```

接下来，实现输入值的模糊化。我们前面已经设计了采用线性量化函数以及三角隶属度函数，所以实现就简单了。

```
/*线性量化操作函数,论域{-6，-5，-4，-3，-2，-1，0，1，2，3，4，5，6}*/
static void LinearQuantization(FUZZYPID *vPID,float pv,float *qValue)
{
  float thisError;
  float deltaError;

 thisError=vPID->setpoint-pv;                  //计算偏差值
  deltaError=thisError-vPID->lasterror;        //计算偏差增量

 qValue[0]=6.0*thisError/(vPID->maximum-vPID->minimum);
 qValue[1]=3.0*deltaError/(vPID->maximum-vPID->minimum);
}
```

对于量化函数实际上可根据需要采用不同的函数，如速降曲线函数，正太分布函数以及其它一元函数等都是可以的，但对于我们在这里的实现目标使用线性函数就足够了。还有隶属度函数也是一样有多种选择，我们这里采用计算量较小的三角隶属度函数：

```
/*隶属度计算函数*/
static void CalcMembership(float *ms,float qv,int * index)
{
 if((qv>=-NB)&&(qv<-NM))
  {
    index[0]=0;
    index[1]=1;
    ms[0]=-0.5*qv-2.0;   //y=-0.5x-2.0
    ms[1]=0.5*qv+3.0;    //y=0.5x+3.0
  }
  elseif((qv>=-NM)&&(qv<-NS))
  {
    index[0]=1;
    index[1]=2;
    ms[0]=-0.5*qv-1.0;   //y=-0.5x-1.0
    ms[1]=0.5*qv+2.0;    //y=0.5x+2.0
  }
  elseif((qv>=-NS)&&(qv<ZO))
  {
    index[0]=2;
    index[1]=3;
```

```
        ms[0]=-0.5*qv;          //y=-0.5x
        ms[1]=0.5*qv+1.0;     //y=0.5x+1.0
    }
    elseif((qv>=ZO)&&(qv<PS))
    {
        index[0]=3;
        index[1]=4;
        ms[0]=-0.5*qv+1.0;    //y=-0.5x+1.0
        ms[1]=0.5*qv;           //y=0.5x
    }
    elseif((qv>=PS)&&(qv<PM))
    {
        index[0]=4;
        index[1]=5;
        ms[0]=-0.5*qv+2.0;    //y=-0.5x+2.0
        ms[1]=0.5*qv-1.0;     //y=0.5x-1.0
    }
    elseif((qv>=PM)&&(qv<=PB))
    {
        index[0]=5;
        index[1]=6;
        ms[0]=-0.5*qv+3.0;    //y=-0.5x+3.0
        ms[1]=0.5*qv-2.0;     //y=0.5x-2.0
    }
}
```

接下来，我们实现模糊推理的函数，有了前面的基础和模糊规则库，模糊计算的函数其实已经简单了。

```
*解模糊化操作,根据具体的量化函数和隶属度函数调整*/
static voidFuzzyComputation (FUZZYPID *vPID,float pv,float *deltaK)
{
    float qValue[2]={0,0};              //偏差及其增量的量化值
    int indexE[2]={0,0};                //偏差隶属度索引
    float msE[2]={0,0};                  //偏差隶属度
    int indexEC[2]={0,0};               //偏差增量隶属度索引
    float msEC[2]={0,0};                 //偏差增量隶属度
    float qValueK[3];

    LinearQuantization(vPID,pv,qValue);

    CalcMembership(msE,qValue[0],indexE);
    CalcMembership(msEC,qValue[1],indexEC);


qValueK[0]=msE[0]*(msEC[0]*ruleKp[indexE[0]][indexEC[0]]+msEC[1]*ruleKp[indexE[0]][in
```

```
dexEC[1]])

+msE[1]*(msEC[0]*ruleKp[indexE[1]][indexEC[0]]+msEC[1]*ruleKp[indexE[1]][indexEC[1]]);

qValueK[1]=msE[0]*(msEC[0]*ruleKi[indexE[0]][indexEC[0]]+msEC[1]*ruleKi[indexE[0]][ind
exEC[1]])

+msE[1]*(msEC[0]*ruleKi[indexE[1]][indexEC[0]]+msEC[1]*ruleKi[indexE[1]][indexEC[1]]);

qValueK[2]=msE[0]*(msEC[0]*ruleKd[indexE[0]][indexEC[0]]+msEC[1]*ruleKd[indexE[0]][in
dexEC[1]])

+msE[1]*(msEC[0]*ruleKd[indexE[1]][indexEC[0]]+msEC[1]*ruleKd[indexE[1]][indexEC[1]]);

  deltaK[0]=LinearRealization(vPID->maxdKp,vPID->mindKp,qValueK[0]);
  deltaK[1]=LinearRealization(vPID->maxdKi,vPID->mindKi,qValueK[1]);
    deltaK[2]=LinearRealization(vPID->maxdKd,vPID->mindKd,qValueK[2]);
}
```

至此，Kp、Ki 和 Kd 的增量已经得到，剩下的就是修正三个参数，并用于实现 PID 调节，与普通的增量型 PID 无异，不再赘述。

4、总结

模糊 PID 算法是模糊算法在 PID 参数整定上的应用，与纯粹的模糊控制算法是有区别的。普通的模糊控制器适用于直接推理控制器的输出，而模糊 PID 算法使用模糊算法修改 PID 参数，最终的控制器输出依然是由 PID 控制器来实现的。

模糊控制本身是非常复杂且具体应用方式很多。大多是针对特定对象的专业控制器，已经脱离了 PID 这种通用性控制器的范畴。此外比较热门的还有模糊多变量控制器是属于先进控制系统（APC）的范畴，有机会再讨论。