



内容目录

第一章.....	数学模型
.....	2
1.1 时域与复域.....	2
1.2 基本数学工具.....	4
1.2.1 微分方程与差分方程.....	4
1.2.2 拉普拉斯变换与传递函数.....	6
1.3 典型环节的微分方程、传递函数及 C 语言实现方法.....	7
1.3.1 比例环节.....	7
1.3.2 惯性环节.....	8
1.3.3 积分环节.....	9
1.3.4 震荡环节.....	10
1.3.5 微分环节.....	11
1.3.6 滞后环节.....	12
1.4 系统辨识方法建立系统模型.....	13
第二章.....	PID 控制及其 C 语言实现
.....	15
2.1 基本 PID 控制原理及实现.....	15
2.2 积分分离 PID 控制实现.....	18
2.3 抗积分饱和 PID 控制实现.....	20
2.4 变积分 PID 控制实现.....	25
2.5 不完全微分 PID 控制实现.....	28
图 2-8 不完全微分 PID 惯性环节仿真数据图.....	31
2.6 其他 PID 控制实现.....	31

第三章.....	工业常用智能算法及其 C 语言实现	32
3.1 专家系统及其 C 语言实现.....		33
3.2 模糊逻辑及其 C 语言实现.....		34
3.3 神经网络及其 C 语言实现.....		48
3.4 遗传算法及其 C 语言实现.....		55
3.5 人工智能与 PID.....		56
第四章.....	实例设计之电源仿真软件	56
4.1 电源控制系统模型.....		56
4.2 选择控制方法.....		56
4.3 实现与仿真.....		56
后记.....		56

第一章 数学模型

被控系统的数学模型是描述系统内在物理量之间关系的数学表达式。系统的数学模型体现了输入量、输出量之间的内在关系，将输入量与输出量通过物理关系连接起来。认清被控系统的数学模型，是设计控制系统的基础。从另外一层意义上讲，对于本领域被控对象数学模型认识的深入程度，直接决定了工程师在该领域所能取得的成就。认识系统的数学模型是进行控制系统设计的基础。

1.1 时域与复域

时域，是以时间做基本变量的范围，描述数学函数或物理信号对时间的关系。**时域**是真实世界，是惟一实际存在的域。我们的经历都是在时域中发展和验证的，我们习惯于事件按时间的先后顺序地发生。而评估数字产品的性能时，通常在时域中进行分析，因为产品的性能最终就是在时域中测量的。正是因为时序是真实世界的反映，在用编程语言描述系统模型或控制系统时，首先需要分析出系统的时域模型。

更多内容易信用户请关注



，微信用户请关注



复域，是指时域的微分方程通过拉普拉斯变换得到的变量在复数范围内的域。得到复域数学模型(传递函数)的目的在于分析系统的稳定性，常用的分析方法包括根轨迹和频域分析法。分析系统的稳定性是设计稳定系统的基础，读者务求深入了解。图 1-1 所示为系统分析设计的一般步骤。

实际系统是真实世界的客观存在，进行系统设计时，首先要根据实际系统，分析出系统的时域模型，然后通过拉普拉斯变换得到系统的传递函数，利用控制系统分析方法，分析系统的稳定性，并根据设计要求设计系统校正环节，然后将校正环节转换为时域系统，利用模拟量或者数字控制技术离散化各个环节，并实现基本的设计，然后根据实际系统控制状态完善控制方式，最终完成整个系统的设计。以上便完成了整个控制系统的设计流程。本书将分不同章节分别从模型、分析、设计、实现四个方面进行讲解，以期达到使读者融会贯通的目的。

本章是从模型的角度出发，分析并说明由实际系统向时域模型的抽象，并进一步生成传递函数的方法。以下能容为本节需要的基本数学知识，如果读者对微分、积分、拉普拉斯变化、拉普拉斯反变换能够充分了解，本节可略去不看。

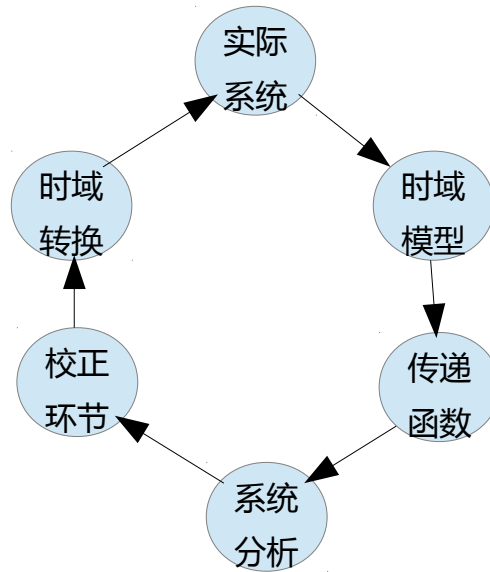


图 1-1 控制系统设计环图

1.2 基本数学工具

1.2.1 微分方程与差分方程

凡是表示未知函数、未知函数的导数与自变量之间关系的方程，都称之为**微分方程**。微分方程用来描述实际系统被关注特性随时间演变的过程，它可以描述系统的动态过程。

微分方程进行离散化变形成了差分方程，而差分方程的时间上的连续发生便会形成微分方程。为什么要建立微分方程与差分方程，因为人们对于变化非常关注，人们往往希望从目前已知的东西上加上合理的预测而得到未来的变化。也就是说如果我们知道“变化值 = 现在值 - 过去值”，那么我们就希望能够得到“未来值 = 现在值 + 变化值”。而微分方程的形式 dm/dt 与差分方程的形式 $(M_{N+1} - M_N)/(T_{N+1} - T_N)$ 恰恰反映了系统



的变化特性。

下面示例 1-1 说明了系统微分方程的建立过程。如图 1-2 所示，电路由电阻 R、电感 L、电容 C 组成，写出以 $U(t)$ 为输入， $U_0(t)$ 为输出的微分方程。

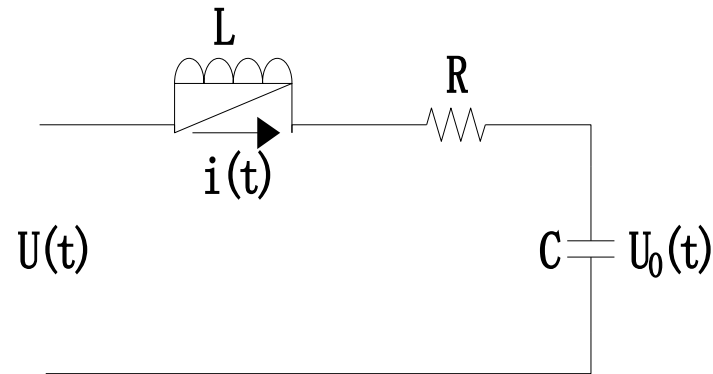


图 1-2 示例 1-1 电路图

基尔霍夫定律指出回路的电压和为 0，那么我们可以知道 $u(t) = L \frac{di}{dt} + \left(\frac{1}{C}\right) \int i(t) dt + Ri(t)$ ，其中我们又知道 $u_0(t) = \frac{1}{C} \int i(t) dt$ ，消去中间变量即可得到电压输入输出的微分方程。

$$LC \frac{d^2 u_0(t)}{dt^2} + RC \frac{du_0(t)}{dt} + u_0(t) = u(t)$$

由公式不难看出，微分方程体现了给定电压 $u(t)$ 与 $u_0(t)$ 随时间 t 的变化关系，是系统的一种动态的体现。对于控制系统而言，控制的过程是动态的过程，控制中所说的平衡也是一种动态的平衡，那么建立系统的动态模型就是完成控制系统设计的重要前提。

下面我们把上述微分方程转换为差分方程的形式。首先需要明确，二次微分是对一次微分求微分，明白这点后，差分方程也就容易写出了。

$$LC \frac{((u_0 t_{(n+1)} - u_0 t_{(n)}) / (t_{(n+1)} - t_{(n)}) - (u_0 t_{(n)} - u_0 t_{(n-1)}) / (t_{(n)} - t_{(n-1)}))}{(t_{(n+1)} - t_{(n)})} + RC \frac{(u_0 t_{(n+1)} - u_0 t_{(n)})}{(t_{(n+1)} - t_{(n)})} + u_0 t_{(n+1)} = ut_{(n+1)}$$

我们不难看出，差分方程能够将微分方程离散化，离散化的系统可以很容易用 C 语言表示出来，从而容易用 C 语言描述出相应的模型，进而为设计控制系统提供模型仿真依据。所以说，微分方程转化为差分方程，最终实现系统的离散化，是理论设计向实际应用转化的必要步骤。

1.2.2 拉普拉斯变换与传递函数

通过上一小节，我们可以知道，系统可以使用多阶微分的形式表示出来，但是问题在于，如果对系统进行分析，微分方程的求解相当麻烦，尤其对于复杂的系统而言，分析多阶微分方程是相当困难的，那么有没有一种方法，能够将复杂的微积分运算转化到简单的四则运算呢，拉普拉斯变换就是提供了这样一种方案，将时间域上的系统转化到复域上来，并在复域上进行分析设计，进行控制设计时，将复域控制系统通过拉普拉斯反变换的方式转化到时间域，进而可以离散化实现控制过程。

定义：时间函数 $f(t)$ ，当 $t \geq 0$ 使有定义，且广义积分 $\int_0^{+\infty} f(t)e^{-st} dt$ 在 s 的某一区域内收敛，则由此积分确定的参数为 s 的函数 $F(s) = \int_0^{+\infty} f(t)e^{-st} dt$ 叫做函数 $f(t)$ 的拉普拉斯变换。

零初始条件下，系统输出量的拉普拉斯变换与输入量的拉普拉斯变化之比就称为线性定常系统的传递函数。

小贴士

如何由微分方程快速进行拉普拉斯变换？

设线性定常系统由下述 n 阶线性常微分方程描述：

更多内容易信用户请关注



，微信用户请关注



$$a_0 \frac{d^n}{dt^n} c(t) + a_1 \frac{d^{(n-1)}}{dt^{(n-1)}} c(t) + \dots + a_{(n-1)} \frac{d}{dt} c(t) + a_n c(t) = b_0 \frac{d^m}{dt^m} r(t) + b_1 \frac{d^{(m-1)}}{dt^{(m-1)}} r(t) + \dots + b_{(m-1)} \frac{d}{dt} r(t) + b_m r(t)$$

公式中， $c(t)$ 为系统输出量， $r(t)$ 为系统输入量。且其各阶导数在 $t = 0$ 均为零。则对上述公式进行拉普拉斯变换后，得到的代数方程为：

$$[a_0 s^n + a_1 s^{(n-1)} + \dots + a_{(n-1)} s + a_n] C(s) = [b_0 s^m + b_1 s^{(m-1)} + \dots + b_{(m-1)} s + b_m] R(s)$$

则系统的传递函数可表示为：

$$G(s) = \frac{C(s)}{R(s)} = \frac{b_0 s^m + b_1 s^{(m-1)} + \dots + b_{(m-1)} s + b_m}{a_0 s^n + a_1 s^{(n-1)} + \dots + a_{(n-1)} s + a_n}$$

1.3 典型环节的微分方程、传递函数及 C 语言实现方法

无论多么复杂的系统，总是可以由简单的子系统构成，分析典型环节的特点，其目的是为了通过典型环节的特点分析更为复杂的系统，实际工程应用中，真正完全通过理论的方式建立模型是非常困难的，实际的模型建立过程是一个复杂的过程，需要通过假设、验证、参数实验给定等多种手段分析完善模型内容，利用实验获取模型的方法又称作系统辨识技术，在下一节中将重点讲解。假设的过程，其实就是根据系统特点，综合典型环节的过程。将不同的环节通过四则运算给予不同的参数进行运算，从而接近真实的模型。从这个角度上讲，透彻理解典型环节的特点，无论对于理论分析系统，还是对于实际建立系统模型都具有重要的意义。

这里的典型环节包括比例、积分、微分、惯性、震荡、滞后六个子环节。下面进行单独讲解。

1.3.1 比例环节

比例环节是自然界普遍存在的一个环节，几乎所有的系统必定存在比例环节。比例环节的特点在于输入输出

量成正比例关系，没有失真与延时。其微分方程可表示为： $c(t) = Kr(t)$ 。其传递函数可表示为：

$$G(s) = \frac{C(s)}{R(s)} = K$$

对于比例环节而言，C 语言实现相对比较简单。下面进行 C 语言实现过程的说明。

```
float ProElement(float K, float GiveValue)
{
float result;
result = K*GiveValue;
return result;
}
```

以上代码用 C 语言实现了比例环节的处理。使用时直接定义一全局变量，例如 float ResultValue; 然后直接调用该函数即可，例如 ResultValue = ProElement(0.1, 10); 返回的 ResultValue = 1.0。

1.3.2 惯性环节

惯性环节是自然界普遍存在的另一个环节，其存在的广泛性不亚于比例环节，任何系统，只有时间精度足够高，都必然存在一定的惯性性能。惯性环节的特点是对变化的输入量，输出量不能立刻复现，或多或少的存在一定的延时，在延时的时间内，输出量会逐渐接近输入的给定值。其微分方程可表示为： $T \frac{dc(t)}{dt} + c(t) = r(t)$ 。

其中 T 为惯性时间常数，T 越大，惯性越大，当延时时间约为 3-4 倍的 T 时，输出接近输入给定值。其传递函数可表示为： $G(s) = \frac{1}{Ts+1}$ 。下面讲解用 C 语言实现惯性环节的过程。这里注意系统的输出为 c(t)，输入为

r(t)。首先将微分方程转化为差分方程的形式： $T \frac{c(t) - c(t-1)}{t - (t-1)} + c(t) = r(t)$ ，化简该公式可表示为：

更多内容易信用户请关注



，微信用户请关注



$c(t) = \frac{r(t) + Tc(t-1)}{T+1}$ 。上述公式表明，当前时刻输出量 $c(t)$ 与上一时刻输出量 $c(t-1)$ 相关。利用该原理，可使

用 C 语言通过迭代函数实现。首先定于全局变量，存放上一时刻输出量的值 `float ResultValueBack`。初值设定为 0，然后可通过下面函数实现迭代过程。

```
float InertialElement(float T, float GiveValue)
{
float result;
result = (T*ResultValueBack + GiveValue)/(1+T);
ResultValueBack = result;
return result;
}
```

1.3.3 积分环节

积分环节是设计校正系统是常用的一个环节，经典的 PID 算法就分别用到了比例、积分、微分三个环节。积分环节的显著特点是，输出量与输入量的积分成正比例关系，当输入量消失后，输出量具有记忆功能，能够储存部分能量。其微分方程可表示为： $c(t) = \int r(d)dt$ 。其传递函数可表示为： $G(s) = \frac{1}{s}$ 。对于离散系统而言，积分过程实质上是系统输入量的累加和，用 C 语言实现积分过程可表示为：

定义全局变量 `ResultValue`

```
float IntegralElement(float GiveValue)
```

```

{
float result;
ResultValue = ResultValue + GiveValue;
result = ResultValue;
return result;
}

```

1.3.4 震荡环节

震荡环节相对而言较为复杂，日常所见系统中，震荡环节体现的相对也少一些，比如 RC 震荡电路，单摆等系统输入震荡系统，该系统的特点在于，系统中存在两个独立储能元件，并可以进行能量交换，从而形成震荡。

其微分方程表示为： $T^2 \frac{d^2 c(t)}{dt^2} + 2\phi T \frac{dc(t)}{dt} + c(t) = r(t)$ ，其传递函数表示为： $G(s) = \frac{1}{T^2 s^2 + 2\phi Ts + 1}$ 。其

中 T 为时间常数， ϕ 为阻尼系数。上述公式表示成差分方程的形式为：

$$T^2 [c(t) - 2c(t-1) + c(t-2)] + 2\phi T [c(t) - c(t-1)] + c(t) = r(t)$$

整理化简得到如下公式： $c(t) = \frac{r(t) + (2T^2 + 2\phi T)c(t-1) - T^2 c(t-2)}{T^2 + 2\phi T + 1}$ ，由上述公式可知输出量 c(t) 与输入量

r(t)，以及前两次的输出量存在关系。类似惯性环节的 C 语言实现方式，我们使用迭代函数实现该环节。

定义全局 float 型变量 ResultValueBackOne, ResultValueBackTwo。

```
float OscilElement(float T, float WP, float GiveValue) //T 为时间常数，WP 表示阻尼系数
```

更多内容易信用户请关注



，微信用户请关注



```
{  
float result;  
result = (GiveValue+(2*T*T+2*WP*T)*ResultValueBackOne-T*T*ResultValueBackTwo)/(T*T+2*T*WP+1);  
ResultValueBackTwo = ResultValueBackOne;  
ResultValueBackOne = result;  
return result;  
}
```

1.3.5 微分环节

微分环节也经常运用到校正环节中，如果被控系统存在较大的惯性环节，可考虑校正环节中加入微分环节，微分环节关注的是给定变化率的特性，所以在一定程度上具有系统预测能力。微分环节根据微分阶数的不同分为一阶微分方程、二阶微分方程及多阶微分方程。这里我们只用C语言实现一阶和二阶微分方程，多阶微分方程可根据实际需要，读者利用本书提到的基本方法自行实现。微分环节的微分方程可表示为，一阶：

$$c(t) = \frac{dr(t)}{dt} \quad ; \quad \text{二阶：} \quad c(t) = T \frac{dr(t)}{dt} + r(t) \quad 。 \quad \text{其传递函数可表示为，一阶：} \quad G(s) = s \quad ; \quad \text{二阶：}$$

$G(s) = Ts + 1$ 。下面是C语言实现一阶微分方程与二阶微分方程的过程。定义全局变量 GiveValueBack。

```
float DervativeElementOne(float GiveValue) //一阶微分实现  
{  
float result;  
result = GiveValue - GiveValueBack;  
GiveValueBack = GiveValue;
```

```
return result;  
}
```

```
float DervativeElementTwo(float T, float GiveValue)//二阶微分实现  
{  
float result;  
result = (T+1) GiveValue - GiveValueBack;  
GiveValueBack = GiveValue;  
return result;  
}
```

1.3.6 滞后环节

滞后环节不同与惯性环节，滞后环节的特点是输入量给定经过一段延时后，输出信号完全复现输入信号。惯性环节则是从输入量给定开始输出量便逐渐趋近于输入量。读者要理解清楚这两个环节的不同。滞后环节的微分方程不易表示，但是 C 语言实现相对简单，只需要延时赋值即可。

更多内容易信用户请关注



，微信用户请关注



```
float DelayElement(float Time, float GiveValue) //Time 表示延时时间
{
float result;
if( T > Time)
{
    result = GiveValue;
}else{
    result = 0;
}
return result;
}
```

以上六小节，详细论述了典型环节的微分方程，传递函数，及 C 语言实现的方法步骤。需要反复强调的是，用 C 语言实现系统模型，首先将系统模型的传递函数转化为微分方程的形式，然后再将微分方程转化为差分方程的形式，最后根据差分方程表达式用 C 语言函数实现。这是控制方法用 C 语言实现的核心所在。

1.4 系统辨识方法建立系统模型

通过系统辨识的方法获得控制系统的数学模型是工程应用中最常用的方法之一。实际的系统远远比理论分析复杂的多，但是对于实际系统而言，如果进行精细化控制，有不得不建立系统的数学模型，如何将理论分析的数据与实际系统无限的接近，系统辨识的方法可以有效的解决此类问题。

系统辨识是一种通过观测一个系统或一个过程的输入与输出关系，确定系统或过程的数学模型的方法。系统辨识一般可分为完全辨识问题和部分辨识问题，所谓完全辨识问题是指，系统的数学结构完全不知道的情况下，完全将系统看做一个黑盒，通过输入与输出的关系推导系统的模型。此种方法在实际应用中非常困难，不易实现精确的数学模型。部分辨识问题是指，系统的主要逻辑关系已经清晰，基本的数学公式结构部分可以表示出来，关键的参数与过程需要试验数据进行修正，此时的辨识问题便成为了参数辨识的问题。此种问题在实际的应用中最为广泛，也最具可操作性。

系统辨识的一般步骤是：

1. 确定和预测被辨识系统数学模型的类型；
2. 给系统施加适当的实验信号，并记录输入——输出数据；
3. 利用最小二乘法进行参数辨识，获得有效的参数数据，获得系统模型；
4. 检验模型有效性，并循环上述步骤至模型符合设计标准。

这里提到了最小二乘法的概念，最小二乘法是实用的曲线拟合方法，其核心思想是使拟合出的曲线满足各个点的误差和最小。详细内容读者可参考线性代数相关教材，这里不再赘述。

更多内容易信用户请关注



，微信用户请关注



第二章 PID 控制及其 C 语言实现

PID 校正控制方式是工程应用中使用最为广泛的一种线性系统控制方法。其核心思想是利用比例、积分、微分三个环节作为校正环节，提高系统的响应速度、稳定性与准确性。其中比例环节能够成比例的反映误差信号，误差信号一旦产生，比例环节立即产生控制作用，以减少变差；积分环节能够有效的消除系统的静态误差，在系统控制过程中，只要存在误差，则积分一直持续，直至误差完全消除，积分环节的强弱直接决定了系统消除误差时间的快慢；微分环节反映了偏差信号的变化趋势，具有一定的预测功能，能够在偏差信号变化太大之前引入一个有效的早期修正信号，加快系统的动作速度，减少调节时间。

2.1 基本 PID 控制原理及实现

在工业应用中，最常用的控制方式是 PID 控制，PID 控制框图如图 2-1 所示。

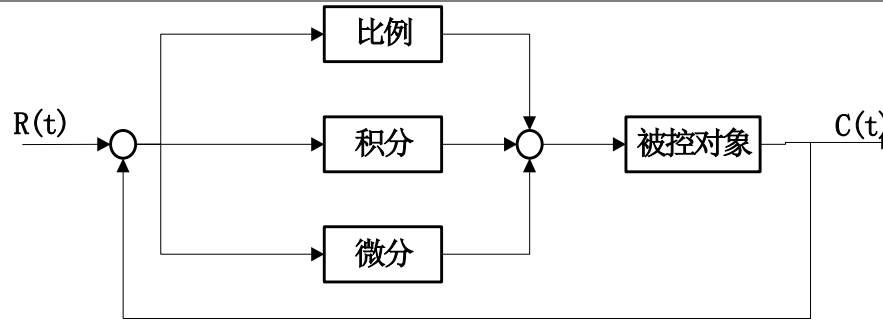


图 2-1 PID 控制框图

PID 是一种线性控制方法，其运用给定值 $R(t)$ 与实际输出值 $C(t)$ 的偏差 $e(t)$ 作为控制量进行控制。其控制规律的微分方程可表示为

$$U(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{de(t)}{dt}$$

其中 K_p 为比例系数、 K_i 为积分系数、 K_d 为微分系数。这里需要注意的是， $U(t)$ 并非系统最终输出 $C(t)$ ，而是被控对象之前节点的值。举个例子说明，假如控制电机以某一转速运行，通过 PID 运算得到的值可能是驱动桥电路的 PWM 开通值，而并非电机的转速值，PWM 开通后作用到电机上才能最终转化到转速上。PID 控制率的传递函数为 $G(s) = \frac{U(s)}{E(s)} = K_p + K_i \frac{1}{s} + K_d s$ ，这里给定其传递函数的目的在于分析系统的需要，进行 C 语言实现时，并不需要传递函数，而是要将微分方程转化为差分方程，再用 C 语言实现。

这里我们采用纯惯性环节作为被控对象，用 C 语言设计最一般的 PID 算法，并进行仿真验证。PID 的 C 语言实现过程如下：

更多内容易信用户请关注



，微信用户请关注



```
float PID(float Kp, float Ki, float Kd, float GiveValue, float ActualValue)
```

```
{  
    float result;  
    float Err, KpWork, KiWork, KdWork;  
    Err = GiveValue - ActualValue;  
    KpWork = Kp*Err;  
    KiWork = Ki*PIDErrADD;  
    KdWork = Kd*(Err-ErrBack);  
    result = KpWork+KiWork+KdWork;  
    PIDErrADD = PIDErrADD + Err;  
    ErrBack = Err;  
    return result;  
}
```

取 $K_p = 4.5$, $K_i = 0.5$, $K_d = 0.1$, 给定值为阶跃信号 100.2 , 惯性环节的惯性系数取 3。最终输出的数据取前 100 个点波形如图 2-2 所示。

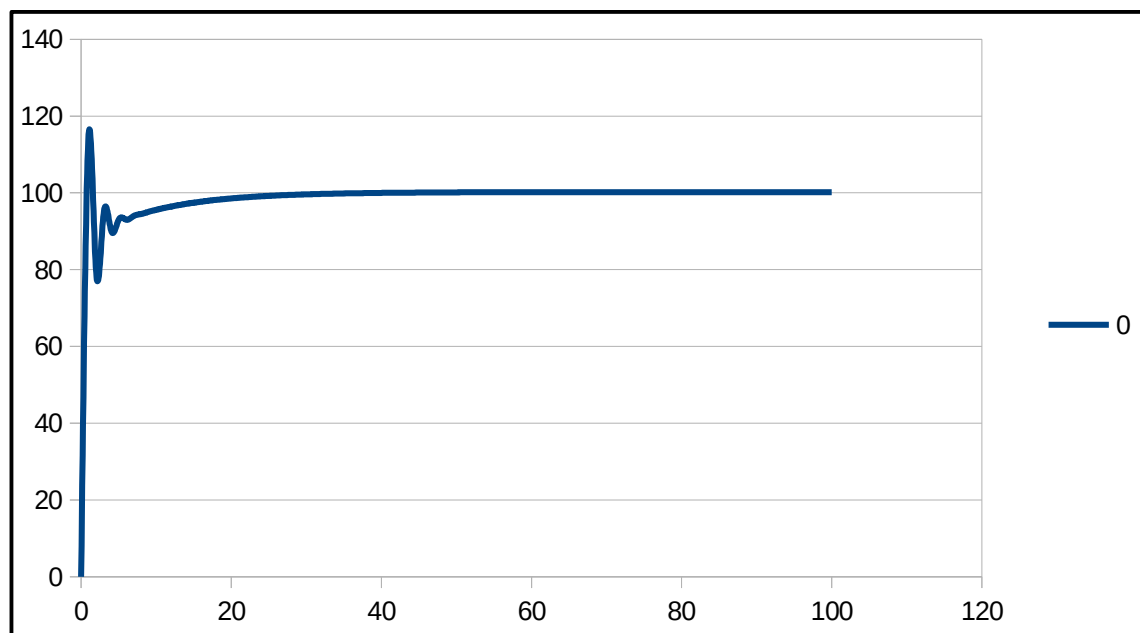


图 2-2 一般 PID 惯性环节仿真数据图

2.2 积分分离 PID 控制实现

控制环节中引入积分环节的目地，主要是为了消除静态误差，提高控制的精度。但是，在实际应用中，当系统处于启动、结束或大幅增减设定的过程中时，短时间内系统输出有很大的偏差，如此便会在短时间内产生较大的积分累计，导致控制量超过执行机构可能允许的最大动作范围对应的极限控制量，进而引起系统较大超调，甚至造成系统的震荡。

引入积分分离的目的在于解决上述问题，其基本思想是，当控制量接近给定值时，引入积分控制，消除静态

更多内容易信用户请关注



，微信用户请关注



误差；当控制量与给定值相差较大时，取消积分作用，避免积分累加和过大造成的系统不稳定因素增加。

这里我们采用纯惯性环节作为被控对象，用 C 语言设计积分分离的 PID 算法，并进行仿真验证。PID 的 C 语言实现过程如下：

```
float SeqIntPID(float Kp, float Ki, float Kd, float GiveValue, float ActualValue)
{
    float result;
    float Err,KpWork, KiWork, KdWork;
    Err = GiveValue - ActualValue;
    KpWork = Kp*Err;
    KiWork = Ki*SeqIntPIDErrADD;
    KdWork = Kd*(Err-SeqIntErrBack);
    if(fabs(Err) > 100)
    {
        result = KpWork+KdWork;
    }else{
        result = KpWork+KiWork+KdWork;
    }
    SeqIntPIDErrADD = SeqIntPIDErrADD + Err;
    SeqIntErrBack = Err;
    return result;
}
```

取 $K_p = 4.5$, $K_i = 0.5$, $K_d = 0.1$, 给定值为阶跃信号 100.2 , 惯性环节的惯性系数取 30。最终输出的数据取前 100 个点波形如图 2-3 所示。

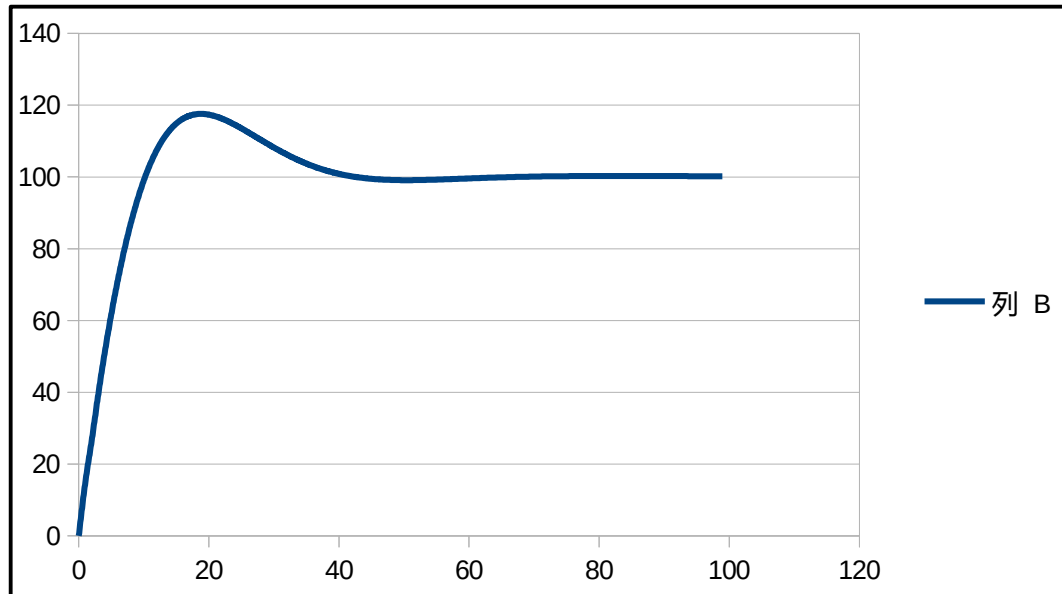


图 2-3 积分分离 PID 惯性环节仿真数据图

2.3 抗积分饱和 PID 控制实现

实际系统的积分项饱和是指，在实际应用系统中，由于误差在一段较长的时间内在一个方向上积累，造成积分的累计误差和过大，从而造成整个积分环节的值过大，当系统趋于稳定或者超调过冲的时候，比例项与微分项的调节作用相对减弱，从而减慢调节速度，甚至长时间不能够到达给定值。

比如对于一个较大的惯性系统，给定系统一个值后，当控制系统极限输出时，惯性系统仍然需要一段时间消



除误差，那么在这段时间内容，正常误差一直在累加，等到输出值大于给定值时，由于前面一段时间累积的误差和太大，从而不能很快的通过误差调整过来。图 2-4 说明了积分饱和的概念。

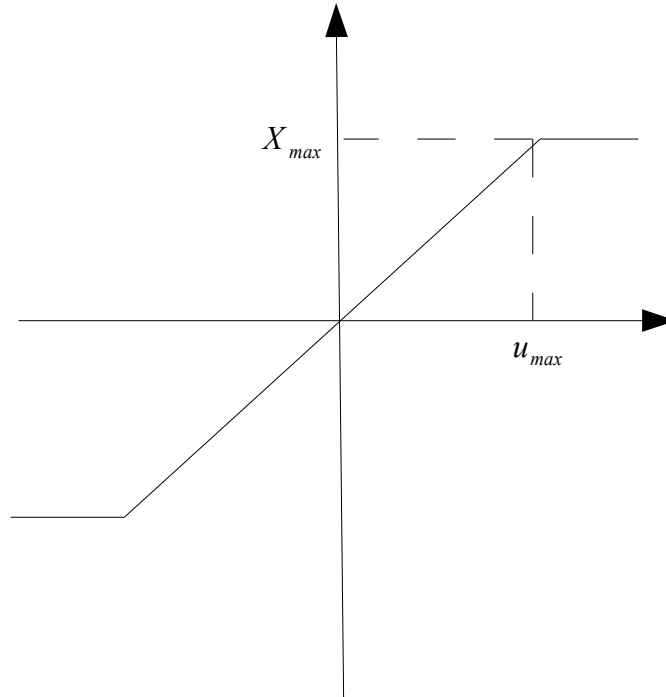


图 2-4 积分饱和说明图

如图 2-4 所示， u 为控制器输出， X 为执行机构的开度，若系统存在一个方向的偏差，PID 控制器的输出由于积分作用累积，导致执行机构达到极限位置 X_{max} ，如果此时控制器输出 u 继续增大，执行器的实际开度已经不可能再增大，此时计算机输出的控制量超出了正常运行的范围而进入饱和状态，一旦系统出现反向偏差，控

制器的输出 u 会逐渐的退出饱和区。但是进入饱和区越深，退出所需时间越久，从而造成系统的失控。为了防止积分项饱和，需要对积分项进行限制。实际当中常用的限制方法方法有两种：

方法一：直接限制积分累加和最大值。所谓直接限制积分累加和是指通过限制积分累加和最大值的方式限制积分项的值使其不能过于深入的进入饱和状态。比如稳定状态下积分累加和的值是 100，那么可以将积分累加和的上下限分别限定到+110 与-110，但是此值的选择需要斟酌，此值选取太小，容易造成系统调整速度慢，甚至不能调整到给定值，此值限制的太大，起不到抗饱和积分的作用。在实际应用中，此种方法以其操作简单被广泛用到。

方法二：动态限制积分累加和。该方法的思路是在计算本次输出 $u(k)$ 时，首先判断上一时刻的控制量 $u(k-1)$ 是否已经超出限制范围 u_{max} ；若 $u(k-1) > u_{max}$ ，则只累加负偏差；反之则只累加正偏差。这种算法可以避免控制量长时间停留在饱和区。

以上两种方法，对于方法一用 C 语言实现较为简便，不再赘述，本处着重对第二种方法进行 C 语言实现。

```
float OverIntPID(float Kp, float Ki, float Kd, float GiveValue, float ActualValue)
```

```
{
    float result;
    float Err,KpWork, KiWork, KdWork;
    Err = GiveValue - ActualValue;
    if(OverIntResultBack > 120)
    {
        if(Err < 0)
        {
```

更多内容易信用户请关注



，微信用户请关注



```
        OverIntPIDErrADD = OverIntPIDErrADD + Err;
    }
}else if(OverIntResultBack < 120){
    if(Err > 0)
    {
        OverIntPIDErrADD = OverIntPIDErrADD + Err;
    }
}else{
    OverIntPIDErrADD = OverIntPIDErrADD + Err;
}

KpWork = Kp*Err;
KiWork = Ki*OverIntPIDErrADD;
KdWork = Kd*(Err-OverIntErrBack);
result = KpWork+KiWork+KdWork;
OverIntErrBack = Err;
OverIntResultBack = result;
return result;
}
```

取 $K_p = 4.5$, $K_i = 0.5$, $K_d = 0.1$, 给定值为阶跃信号 100.2 , 惯性环节的惯性系数取 30。最终输出的数据取前 100 个点波形如图 2-5 所示。

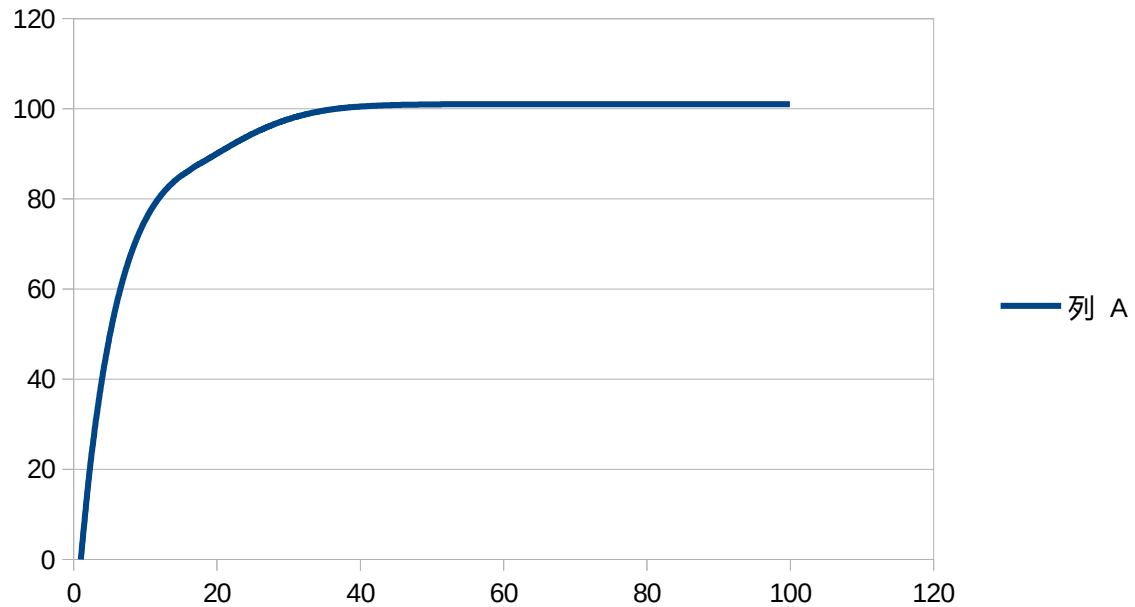


图 2-5 抗饱和积分 PID 惯性环节仿真数据图

对比 2-3 与 2-5，可以发现抗饱和积分可以使系统更加稳定。

2.4 变积分 PID 控制实现

一般的 PID 控制算法中，积分系数 K_i 是常数。系统对积分项的要求是，当系统偏差大时积分作用应该减弱，系统偏差小时，积分作用应该增强。对于固定积分系数而言，当积分系数取大时容易产生超调，甚至积分饱和，取小时又迟迟不能消除静态误差。因此如果能够根据偏差大小改变积分的速度就能解决上述课题。

变积分的思想就是根据偏差大小来改变积分项的累加速度。偏差越大则积分越慢，偏差越小积分便越快。但是需要注意的是，变积分在工业应用中需要根据实际系统客观分析选用。如果实际系统的惯性环节较小，系统的响应速度较快，系统变化的控制较为剧烈，则需要谨慎使用变积分的方法，因为剧烈的偏差变化容易造成系统的不稳定，甚至严重的震荡。在本节中，对于一般情况的变积分进行处理，本节依然采用惯性环节作为理想的控制对象。

进行变积分 PID 设计的核心思想是，给定累加的偏差一个权值，当系统偏差较小时，可以累加完全的偏差值，当系统的偏差较大时可以累积不完全的偏差值，甚至不进行偏差值的累加。

设置系统 $f(e(k))$ ，它是 $e(k)$ 的函数，则变积分的 PID 积分项表达式可表示为：
 $u(k) = K_i \left\{ \sum e(i) + f[e(k)]e(k) \right\}$ ， $f(e(k))$ 需满足以下两个条件：

1. $f(e(k))$ 在区间[0,1]内；
2. $f(e(k))$ 与 $e(k)$ 成反比例关系。

根据以上要求，设计 C 语言，利用延时环节作为被控对象。

```
float ChangeIntPID(float Kp, float Ki, float Kd, float GiveValue, float ActualValue)
{
    float result;
    float Err, KpWork, KiWork, KdWork, ErrCont;
    Err = GiveValue - ActualValue;
    KpWork = Kp*Err;
    KiWork = Ki*ChangeIntPIDErrADD;
```

更多内容易信用户请关注



，微信用户请关注



```
KdWork = Kd*(Err-ChangeIntErrBack);
result = KpWork+KiWork+KdWork;
if(fabs(Err)<= GiveValue*0.1)
{
    ErrCont = Err;
}else if((fabs(Err)<= GiveValue*0.9)&&(fabs(Err)> GiveValue*0.1))
{
    ErrCont = ((0.9*GiveValue)-fabs(Err))*Err/(0.8*GiveValue);
}else{
    ErrCont = 0;
}

ChangeIntPIDErrADD = ChangeIntPIDErrADD + ErrCont;
ChangeIntErrBack = Err;
return result;
}
```

取 $K_p = 4.5$, $K_i = 0.5$, $K_d = 0.1$, 给定值为阶跃信号 100.2 , 惯性环节的惯性系数取 30。最终输出的数据取前 100 个点波形如图 2-6 所示。

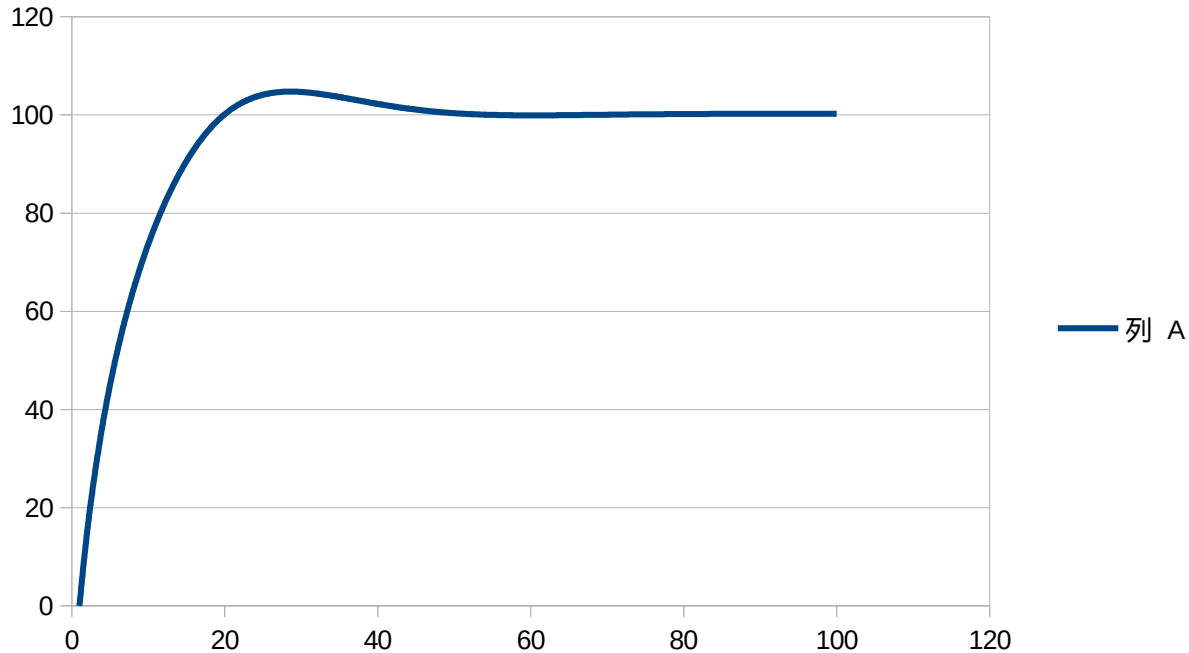


图 2-6 变积分 PID 惯性环节仿真数据图

2.5 不完全微分 PID 控制实现

在 PID 控制中，微分信号的可以改善系统的动态特性，但也容易引入高频干扰，在误差信号存在扰动是，更是能够显示微分的不足之处。克服上述问题的方法之一就是在采用不完全微分 PID 控制，所谓不完全微分法就是在 PID 控制算法的微分项中加入一个一阶惯性环节。如此可是系统性能得到改善。

微分项中加入惯性环节可以理解为让微分作用不能够立即体现，加入说微分作用给定的信号输出时 100，那么加入惯性环节后，其不能立即输出 100，而是由 0 逐渐上升到 100，上升的速率与惯性环节的系数有关。采用上述方法的优点在于，如果误差信号存在高频干扰，那么与误差信号直接关系的微分项不能发生突变，从而



起到滤波的作用。不完全微分的结构如图 2-7 所示。

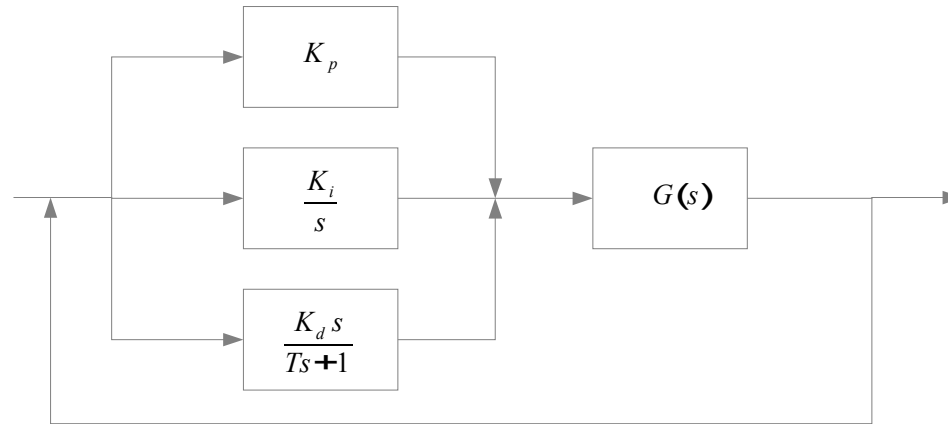


图 2-7 不完全微分 PID 控制结构图

我们利用第一章中的内容将微分项的传递函数转化为差分方程的形式，设定微分项的输入为 $E(s)$ ，输出为 $U(s)$ ，则根据框图可知 $U(s) = \frac{K_d s}{Ts+1} E(s)$ ，转化为差分方程，可表示为

$$U(k) + T[U(k) - U(k-1)] = K_d [Err(k) - Err(k-1)]$$

化简整理的出： $U(k) = \frac{K_d [Err(k) - Err(k-1)] + TU(k-1)}{T+1}$ ，令 $\frac{T}{T+1} = \alpha$ ，则最终的差分方程可表示为：

$U(k) = K_d(1-\alpha)(Err(k) - Err(k-1)) + \alpha U(k-1)$ ，利用上述公式可用 C 语言进行实现并仿真。

```
float NoComDPID(float Kp, float Ki, float Kd, float Aifa, float GiveValue, float ActualValue)
```

```
{
```

```

float result;
float Err,KpWork, KiWork, KdWork;
Err = GiveValue - ActualValue;
KpWork = Kp*Err;
KiWork = Ki*NoComIntPIDErrADD;

if(Aifa > 1)
{
    Aifa = 1;
}

if(Aifa < 0)
{
    Aifa = 0;
}
KdWork = Kd*(1-Aifa)*(Err-ErrBbk)+ Aifa*KdWorkBbk;

result = KpWork+KiWork+KdWork;

NoComIntPIDErrADD = NoComIntPIDErrADD + Err;
KdWorkBbk = KdWork;
ErrBbk = Err;

```

更多内容易信用户请关注



，微信用户请关注



```
return result;
```

```
}
```

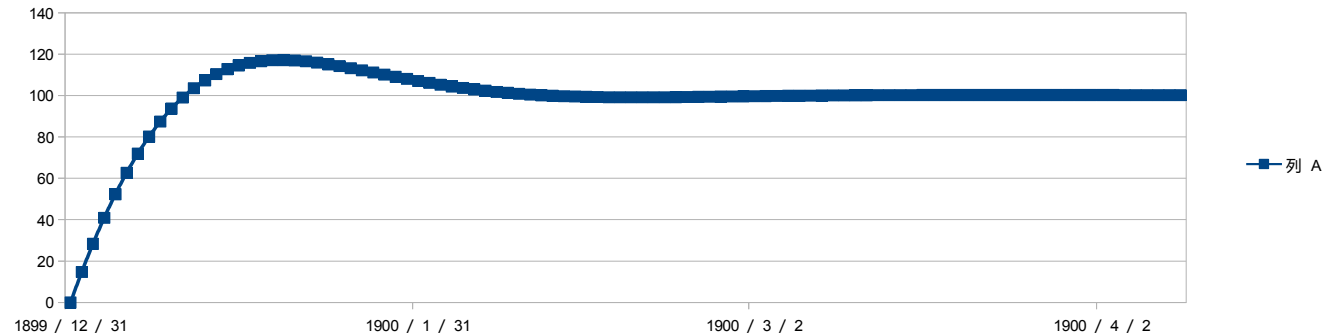


图 2-8 不完全微分 PID 惯性环节仿真数据图

2.6 其他 PID 控制实现

PID 的在工业上广泛的应用，具体使用过程中会针对不同的控制系统研究选择不同的 PID 控制形式，PID 实际应用中多种多样，上述五个小节中讲述了五种 PID 的实现形式，实际的工业应用中常用的 PID 还有带前馈的 PID、基于滤波器的 PID 以及智能 PID 等其他使用形式，读者需掌握系统与控制校正环节的分析方法，灵活运用，

实现适合自身系统的 PID 控制器。

第三章 工业常用智能算法及其 C 语言实现

所谓智能控制算法是指使系统具备分析、推理、学习能力的控制方法。我们在学习自动控制原理的时候，总是在强调一句话：古典控制理论只是适合于单输入单输出的线性系统，PID 控制是古典控制理论的典型应用。但是现实的世界往往不是单输入单输出的，更不可能是纯线性的系统。所以，单纯的 PID 控制往往在系统的局部上能够其作用。而且现实当中的我们往往无法准确的建立起系统的数学模型，可以想象我们希望得到精确的系统控制是一件多么复杂的事情。

智能算法的出现在一定程度上能够解决上述问题，智能算法模拟人的思维方式，通过分析、推理与学习，逐渐逼近事物的真相，不断优化控制参量，从而实现自适应的控制方式。通过上述的感性描述，我们不难得到这样的结论：

第一，智能控制更适合复杂的、非线性的、多变量输入输出系统；

第二，由于智能控制需要分析、推理与学习来进行系统控制，故而其控制的速度一般不如传统的控制方式快。

当然，随着 CPU 处理速度的加快，智能控制速度上的劣势逐渐的不再明显，而其广泛适用性的特点确能够体现的非常充分。本章通篇讨论智能控制，其目的也是希望能够推广智能控制在工业上的应用，最终实现系统的最优化控制，制造出优秀的产品。

常用的智能算法包括专家系统、模糊逻辑、神经网络、遗传算法，本章的四个小节将分别尝试性的进行算法的设计与仿真。

更多内容易信用户请关注



，微信用户请关注



3.1 专家系统及其 C 语言实现

专家系统落实到理论层次上感觉不太容易，而实际上，专家系统即为人对事物认识程度的体现，比如对于天气预报的认识，我们认为当天空是蓝色、有太阳、无风，那么就说明未来 10 小时是晴天。上述推论是基于我们对于天气变化的认识给出的结论。其实 C 语言的 if 表达式，switch 表达式在一定程度上就是一种基于常理的推论，其可以作为一种较为简单的专家系统来看待。比如上述天气的例子可以用 C 语言表述为

```
if((Sky == blue)&&(Sun == true)&&(Wind == false))
{
    WeatherAfterTenHours = Sunny;
}
```

基于这个认识，我们可以将所有的逻辑推理与分析过程都归结到理论上的专家系统的概念上，那么专家系统也就不再神秘。

专家系统实际上就是一个基于经验的数据库。经验越丰富，那么数据库越全面，那么控制精度越高。专家数据库是数据库的一种，当然这个数据库可以使 sqlite、mysql 等任何数据库形式，也可以使一个数组或者存放在 FLASH 中的数据表，与普通数据库不同的是，专家数据库更侧重于逻辑上的推力，其基本形式可如下表所示：

条件 1	条件 2	结果
1	2	3
1	3	4

2	2	4
2	3	9
2	5	8

也就是说，数据库本省表述了条件到结果的关系，由条件 1 与条件 2 能够推出结果，条件 1 与条件 2 之间可以是四则运算关系，也可以是逻辑关系，也可以是某种毫无法判定的非线性关系，但是条件 1 与条件 2 的给定能够得到相应的结果就足够了。也正是这样，专家系统可以满足非线性的需求。专家数据库一般靠大量的实验获得。

专家数据库实际上是模拟人类智能的记忆能力，记忆能力是智能生命最基本的能力，所有的分析、计算、处理能力都是建立在大数据量信息记忆的前提下。如果要实现真正的智能算法，专家数据库是不可或缺的重要环节。当前进入了后互联网时代，智能手机、智能硬件一夜之间成为了追捧的对象，就目前的所谓的“智能”系统而言，更多的仍然是建立在大数据的前提下——即互联网模式。在人们的认识中，联网就是智能。而实际上，能够连接互联网，并通过互联网获得与分享数据，也就是所说的物联网，仅仅是对于智能记忆能力的一种体现，而真正智能硬件处理具备联网能力，还应该具备逻辑分析与联想分析的能力，这样每个终端都是智能终端，而互联网本身也具备信息综合、处理、加工能力，才能构建智能网络，才是真正的智能。下一节开始，我们讲解智能的另外一个方面，模糊处理的能力。

3.2 模糊逻辑及其 C 语言实现

模糊逻辑一般是与专家数据库一同使用的。现在也许你还无法理解这句话，但是我必须把这句话说在最前面。接下来解释什么是模块逻辑，也称之为模糊控制方法。模糊逻辑是人类智能最常用的逻辑分析方法。当然，这种逻辑分析方法一定是建立在一定的记忆（专家数据库）的前提下的。比如我们人类可以区分颜色，当我们看

更多内容易信用户请关注



，微信用户请关注



到天空的时候（无雾霾的情况下），我们会说这是蓝色，当我们看到叶子的时候，我们会说是绿色，当我们看到红玫瑰的时候，我们会说是红色。可实际上，蓝色的 RGB 值为 $(0,0,255)$ ，绿色的 RGB 值为 $(0,255,0)$ ，红色的 RGB 值为 $(255,0,0)$ 。那么难道说天空、叶子、玫瑰的 RGB 值与理论的 RGB 值完全一致吗？显然这是不可能的。那么我们为什么能够恰当的区别这些不同颜色，而且能够恰当的归类到不同的颜色当中呢。这其中体现的就是一种模糊逻辑的思维方式。当然，这个思维方式一定时间里在专家数据库的基础上，试想，如果当一个婴儿出生的时候，他的父母就反复告诉他叶子是红色的，天空是黑色的，那么，等这个婴儿长大了，他就会得到与其他孩子完全不同的颜色体验。也就是说，这个婴儿数据库中的给定值是存在问题的。而专家数据库之所以称之为专家数据库，就证明这个数据库是正确的、可信的、符合人类正常认识习惯与评价标准的。其实对于人工智能的研究，越往深处研究，研究范畴越脱离了控制与数学的范畴，而是更偏向哲学的范畴，当然这不是我们所关注的内容了，我们所关注的是人工智能的数学应用范畴。

下面我们通过一个简单例子来实现模糊逻辑。注意：这里仅仅说明什么是模糊逻辑，并没有涉及到模糊控制。模糊控制是采用模糊逻辑实现的控制方法。前面我们介绍模糊概念的时候，混淆这两个概念，是让大家对模糊从方法到应用上有一个感性的认识。但是在这里我们需要弄清楚了。我们以受教育程度和工作经验来决定某个人的工资待遇为话题说明模糊逻辑。

假定一个人的入职工资待遇收到学历高低与工作经验（工作时间）两方面因素的影响。一般而言受教育时间表示一个人的受教育程度；工作经验从刚毕业到工作 30 年甚至更长，用工作时间长短表示某个人的工作经验。那么受教育多长时间算是受教育程度高呢，工作多长时间算是工作经验长呢？一般来讲，我们认为，受教育超过 25 年算是受教育程度很高了，工作时间超过 10 年就算是工作经验丰富了。那么受教育 10 年算受教育程度高吗？工作 5 年算工作经验丰富吗？根据不同人的不同立场，可能有人认为这样也算不错了，有人可能认为，这

根本不算什么。那么就需要就一个模糊的标准来评判这个事情。下面我们分步骤的来分析这个案例。

第一步：明确对象变量。这个评判系统中，对象的输入量为受教育时间 T_E 与工作时间 T_W ，而输出量是工资待遇 M ，我们假定工资待遇从 1000—10000 不等。

第二步：将每个变量划分为若干个模糊子集，并给每个模糊子集分配一个语言符号。受教育时间 T_E 与工作时间 T_W 我们可以用长短来说明。工资待遇 M 我们可以用高低来说明。这样，长、短、高、低本来就是一种模糊的概念了。

第三步：确定每个模糊子集的隶属函数。这一步，我们将说明多长时间算长，多高工资算高的问题。先看下图。

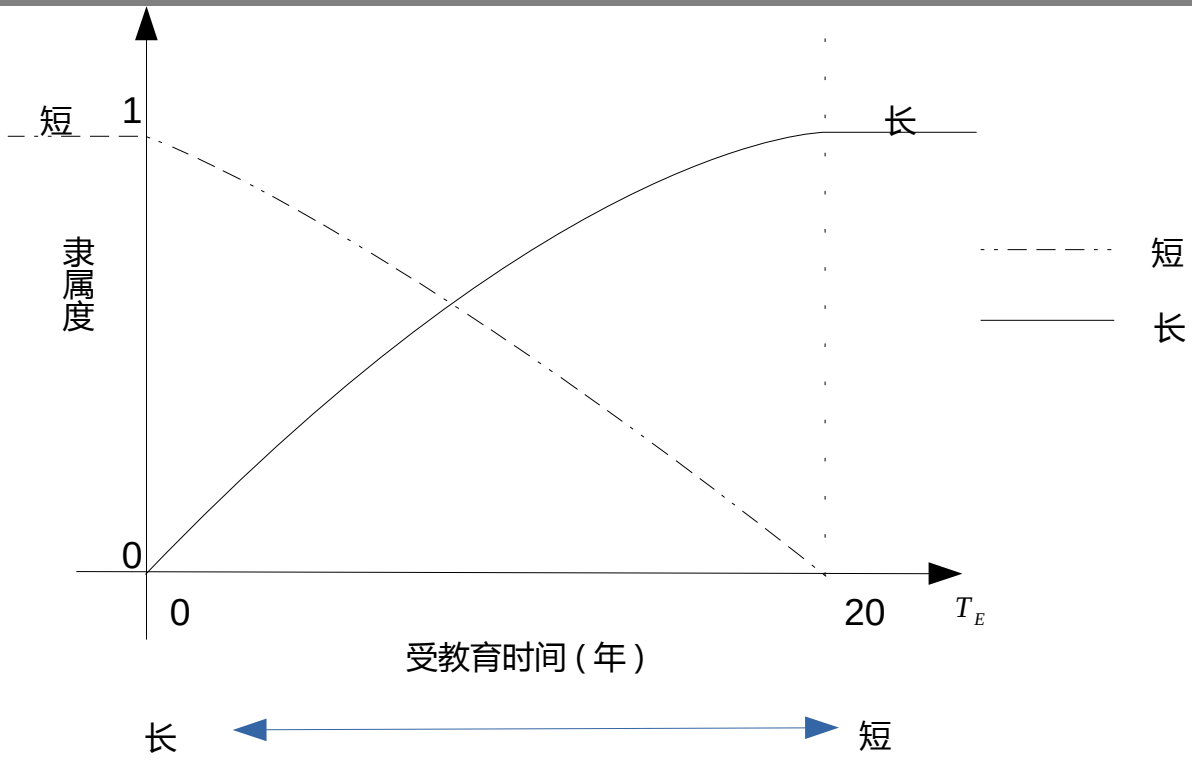


图 3-1 受教育时间隶属度函数图

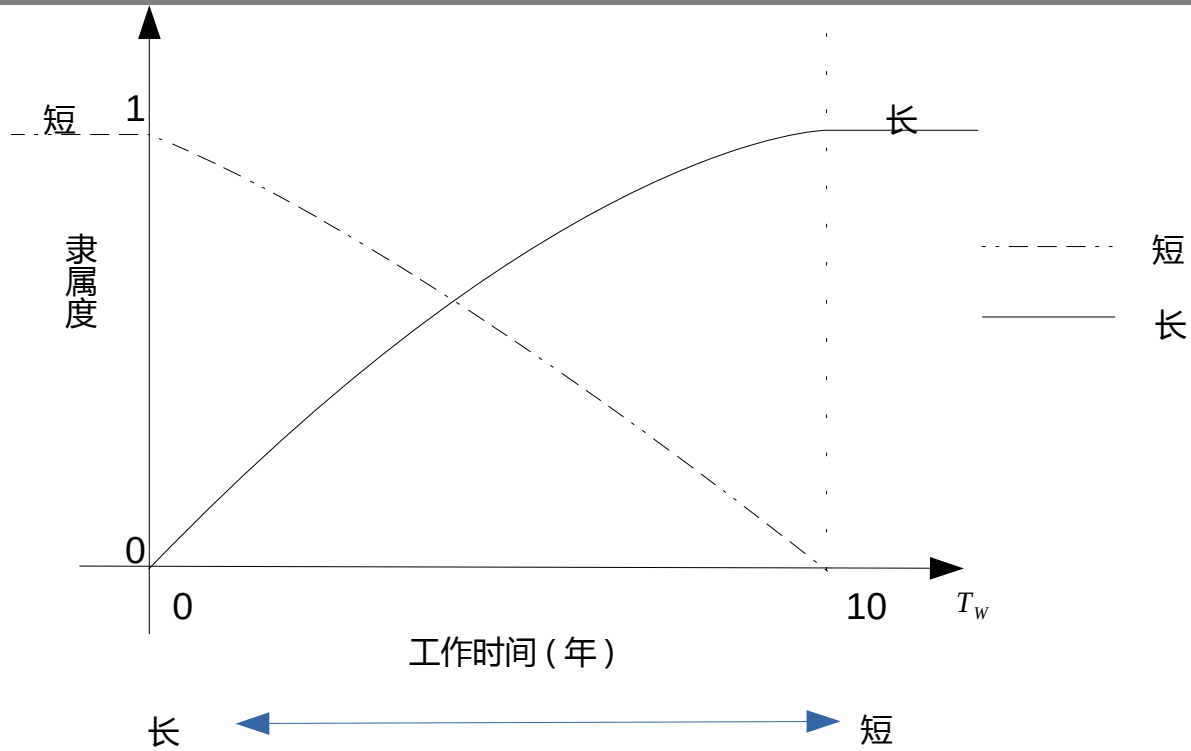


图 3-2 工作时间隶属度函数图

图 3-1 表示受教育时间隶属度函数，图 3-2 表示工作时间隶属度函数。下面详细说明一下图 3-1 的内容。图 3-2 内容同图 3-1 相似。图 3-1 给出了受教育时间长、短两种模糊描述的隶属函数，图 3-1 的实线曲线表示受教育时间长的隶属函数，从曲线当中可以看出，当受教育时间为 0 年时，那么受教育时间长的隶属度为 0，当然根据实际情况我们可以知道，实际当中受教育时间不可能为负数。当受教育时间大于 20 年时，那么受教育时间长的隶属度为 1，而 0-20 年之间则分别对应了 0-1 之间不同的隶属度，趋势是随着受教育时间则增长，受教育时间长的隶属度也变得越来越大。

更多内容易信用户请关注



，微信用户请关注



同理，长短线代表了受教育时间短的隶属度函数曲线。当受教育时间为 0 时，受教育短的隶属度为 1；当受教育时间大于 20 时，受教育短的隶属度为 0。当然，这个隶属度函数曲线我们可以进行自行的定义，他可以是线性的，也可以是非线性。当然也可以使用一些技术手段使得隶属函数变得更加合理，3.3 节中的神经网络就是可以提高隶属度函数合理性的一个不错的方案。在这里，我们姑且认为隶属度函数是线性的。那么设受教育时间长的隶属度函数为 $F_{EL}(x)$ ，受教育时间短的隶属度函数为 $F_{ES}(x)$ ，工作时间长的隶属度函数为 $F_{WL}(x)$ ，工作时间短的隶属度函数为 $F_{WS}(x)$ 。那么，

$$F_{EL}(x) = \frac{1}{20}x \quad x \in [0, 20] \quad \text{当 } x > 20, F_{EL}(x) = 1$$

$$F_{ES}(x) = -\frac{1}{20}x + 1 \quad x \in [0, 20] \quad \text{当 } x > 20, F_{ES}(x) = 0$$

$$F_{WL}(x) = \frac{1}{10}x \quad x \in [0, 10] \quad \text{当 } x > 10, F_{WL}(x) = 1$$

$$F_{WS}(x) = -\frac{1}{10}x + 1 \quad x \in [0, 10] \quad \text{当 } x > 10, F_{WS}(x) = 0$$

第四步，形成规则库。结合上述实例，我们可以尝试用如下描述语言进行规则描述。

规则一：如果受教育时间 T_E 短，且工作时间 T_W 短，那么工资待遇 M 低；

规则二：如果受教育时间 T_E 短，且工作时间 T_W 长，那么工资待遇 M 中；

规则三：如果受教育时间 T_E 长，且工作时间 T_W 短，那么工资待遇 M 中；

规则四：如果受教育时间 T_E 长，且工作时间 T_W 长，那么工资待遇 M 高。

我们可以用下表来表示上述四条规则：

M	T_E 短	T_E 长
T_W 短	低	中
T_W 长	中	高

第五步：规则计算。这里需要说明的是，模糊运算是基于概率论发展而来的控制策略。这点我想是好理解的，因为模糊的本身就是基于概率的一种推测。其最终的目的就是根据模糊时间出现推导出结果的概率，从而进一步进行结果归类，得到预计的输出。既然是根据概率论演化而来的控制策略，那么其计算过程就离不开概率的运算。上述例子中，我们用到了“且”的关系，概率论中还存在“或”的关系，当然“或”的关系也可以通过若干个“且”的关系进行表述。但是我们仍然要知道“且”与“或”的概率运算。例如，事件 A 的概率为 a，事件 B 的概率为 b，那么 A 且 B 的概率为 $a \times b$ ；A 或 B 的概率为 $\neg a \times b + a \times \neg b$ 。接下来我们根据规则来计算工资待遇。

首先根据规则一计算工资待遇低的隶属度： $M_{Low} = F_{ES}(x) \times F_{WS}(x)$ ；

然后根据规则二、规则三计算工资待遇中的隶属度函数： $M_{Mid} = F_{EL}(x) \times F_{WS}(x) + F_{ES}(x) \times F_{WL}(x)$

最后根据规则四计算工资待遇高的隶属度函数： $M_{High} = F_{EL}(x) \times F_{WL}(x)$

更多内容易信用户请关注



，微信用户请关注



第六步，去模糊化。设定工资待遇低为 1000，工资待遇中为 5000，工资待遇高位 10000。那么最终的工资待遇可表示为：

$$M = \frac{1000 M_{Low} + 5000 M_{Mid} + 10000 M_{High}}{M_{Low} + M_{Mid} + M_{High}}$$

我们举例来测试一下模糊规则是否可靠，假如某人的受教育时间为 15 年，工作了 5 年，则利用上述 6 步可以得到该人的工资待遇为约 5047.619 元/月。接下来，我们用 C 语言来实现上述过程。

```
#define EDUMAX 20 //定义受教育时间最大值
#define EDUMIN 0 //定义受教育时间最小值
#define WORKMAX 10 //定义工作时间最大值
#define WORKMIN 0 //定义工作时间最小值
#define WAGEMAX 10000 //定义工资待遇最大值
#define WAGEMID 5000 //定义工资待遇中值
#define WAGEMIN 1000 //定义工资待遇最小值
```

```
/*
```

```
*分别定义教育时间长、短，工作时间长、短的隶属度函数
```

```
*/
```

```
float EduL(float eduTime);
```

```
float EduS(float eduTime);
```

```

float WorkL(float workTime);
float WorkS(float workTime);
/*
*定义教育时间,工作时长,工资待遇三个变量
*/
float EduTime;
float WorkTime;
float Wage;
/*
*定义工资待遇高、中、低的隶属度函数
*/
float WageH();
float WageM();
float WageL();
//定义最终工资输出函数
float WageFinal();

```

实现定义的函数

```

float EduL(float eduTime)
{
    float result;
    if(eduTime > EDUMAX)

```

更多内容易信用户请关注



，微信用户请关注



```
{
    result = 1;
}else if(eduTime < 0)
{
    result = 0;
}else{
    result = eduTime/EDUMAX;
}
return result;
}
float EduS(float eduTime)
{
    float result;
    if(eduTime > EDUMAX)
    {
        result = 0;
    }else if(eduTime < 0)
    {
        result = 1;
    }else{
```

```

    result = 1-eduTime/EDUMAX;
}
return result;
}

```

```

float WorkL(float workTime)
{
    float result;
    if(workTime > WORKMAX){
        result = 1;
    }else if(workTime < 0)
    {
        result = 0;
    }else{
        result = workTime/WORKMAX;
    }
    return result;
}

```

```

float WorkS(float workTime)
{
    float result;

```

更多内容易信用户请关注



，微信用户请关注



```
if(workTime > WORKMAX){
    result = 0;
}else if(workTime < 0)
{
    result = 1;
}else{
    result = 1-workTime/WORKMAX;
}
return result;
}

float WageH()
{
    float result;
    result = EduL(EduTime)*WorkL(WorkTime);
    return result;
}

float WageM()
{
```

```

float result;
result = EduL(EduTime)*WorkS(WorkTime)+EduS(EduTime)*WorkL(WorkTime);
return result;
}

```

```

float WageL()
{
    float result;
    result = EduS(EduTime)*WorkS(WorkTime);
    return result;
}

```

```

float WageFinal()
{
    float result;
    result = WAGEMAX*WageH()+WAGEMID*WageM()+WAGEMIN*WageL();
    return result;
}

```

主函数实现

```

printf("Please input the Edu Time:\r\n");
scanf("%f",&EduTime);

```

更多内容易信用户请关注

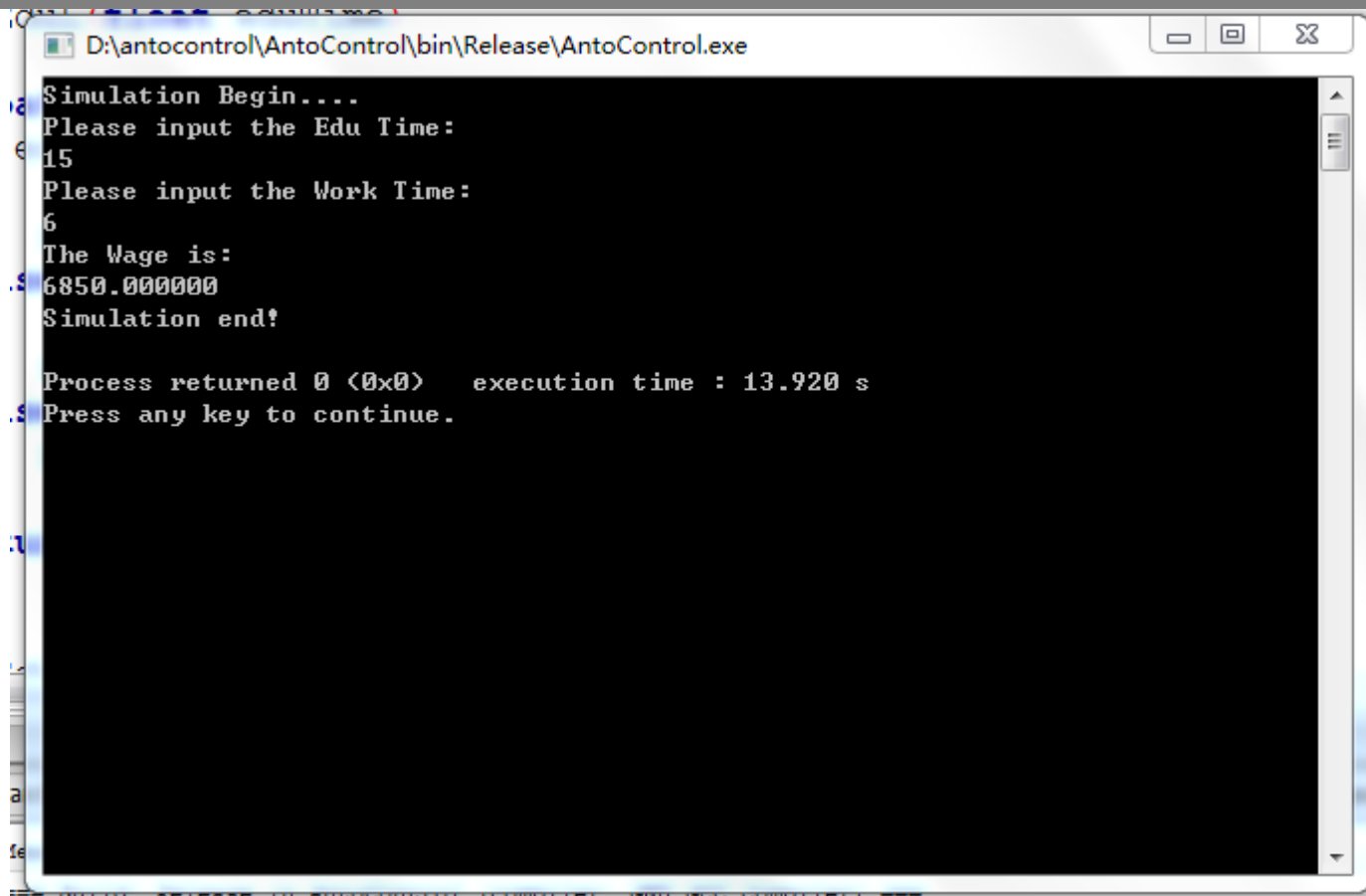


，微信用户请关注



```
if(EduTime < 0)
{
    EduTime = 0;
}
printf("Please input the Work Time:\r\n");
scanf("%f",&WorkTime);
if(WorkTime < 0)
{
    WorkTime = 0;
}
Wage = WageFinal();
printf("The Wage is:\r\n");
printf("%f\r\n",Wage);
printf("Simulation end!\r\n");
```

运行结果如下：



```
D:\antocontrol\AntoControl\bin\Release\AntoControl.exe
Simulation Begin...
Please input the Edu Time:
15
Please input the Work Time:
6
The Wage is:
6850.000000
Simulation end!

Process returned 0 (0x0)   execution time : 13.920 s
Press any key to continue.
```

3.3 神经网络及其 C 语言实现

如果说模糊逻辑控制是一种宏观上的人工智能形式，那么神经网络就更为的深入到智能的本质，从微观的角度上解释与运用人工智能来进行系统的控制。神经网络的根本就是通过构造不同拓扑的相互连接的神经元来模拟智能的核心——大脑的本质。生物学上讲，大脑的本质就是无数个神经元组成的集合体，人类通过对大脑神经反复的刺激、训练，最终形成优秀的记忆与思维能力。神经网络模拟大脑的组成，从而具备一定的记忆能力、



自适应能力与推力能力。这里我们不再介绍生物学上的神经元与连接方式，也不再考虑什么轴突、树突之类的专业名词。如果读者对神经网络没有基本的认识的话，建议先到互联网搜索相关信息作为了解。下面的一些介绍，我都默认各位读者已经理解了神经网络的基础构造知识。下面讲解控制系统上用到的神经元结构。

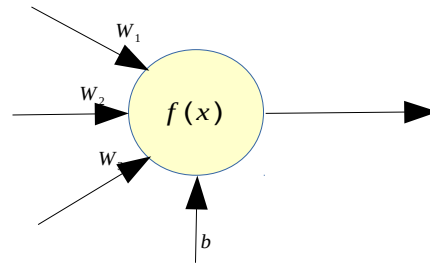


图 3-3 单个神经元结构图

单个神经元由输入、偏置、输出三个变量部分组成，就单个神经系统而言，是一个多输入单输出系统。多个输入变量输入到神经元之后，神经元将输入进行处理，加上自身的偏置信息，经神经元加工后传导到下一个神经元。这也就在物理上模拟了大脑连接的过程。下面给出神经元的激励函数

$$Y=f(\sum w_i \cdot x_i - b)$$

可以看出，神经元激励函数 $f(x)$ 实际上是对 $\sum w_i \cdot x_i - b$ 的输出函数。激励函数可以是多种形式，在这里我们只介绍比较常用的简单的 BP 网络，其他网络形式读者可参考本文介绍，自行研究 C 语言的实现方法。BP 网络中最常用的激励函数有三种，线性函数、指数函数、双指数函数。分别表示为： $f(x)=x$ ， $f(x)=\frac{1}{1+e^{-x}}$ ，

$$f(x) = \frac{2}{1+e^{-2x}} - 1, \text{ 其中对于三个函数均有 } f(x) \in [0, 1] \text{。}$$

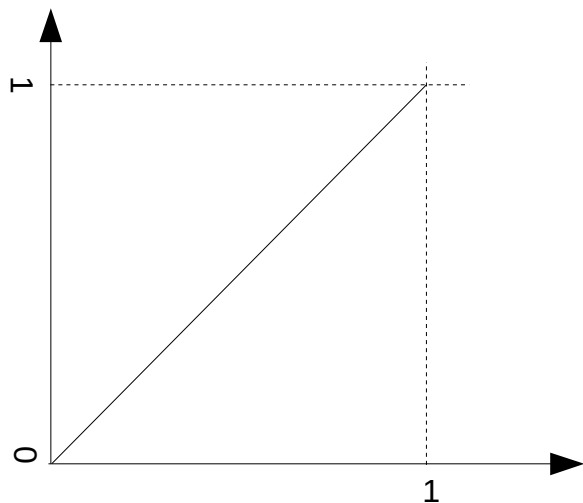


图 3-4 线性函数曲线图

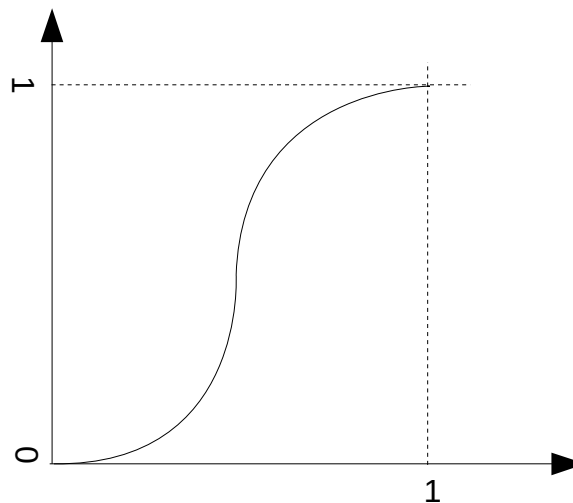


图 3-5 指数函数曲线图

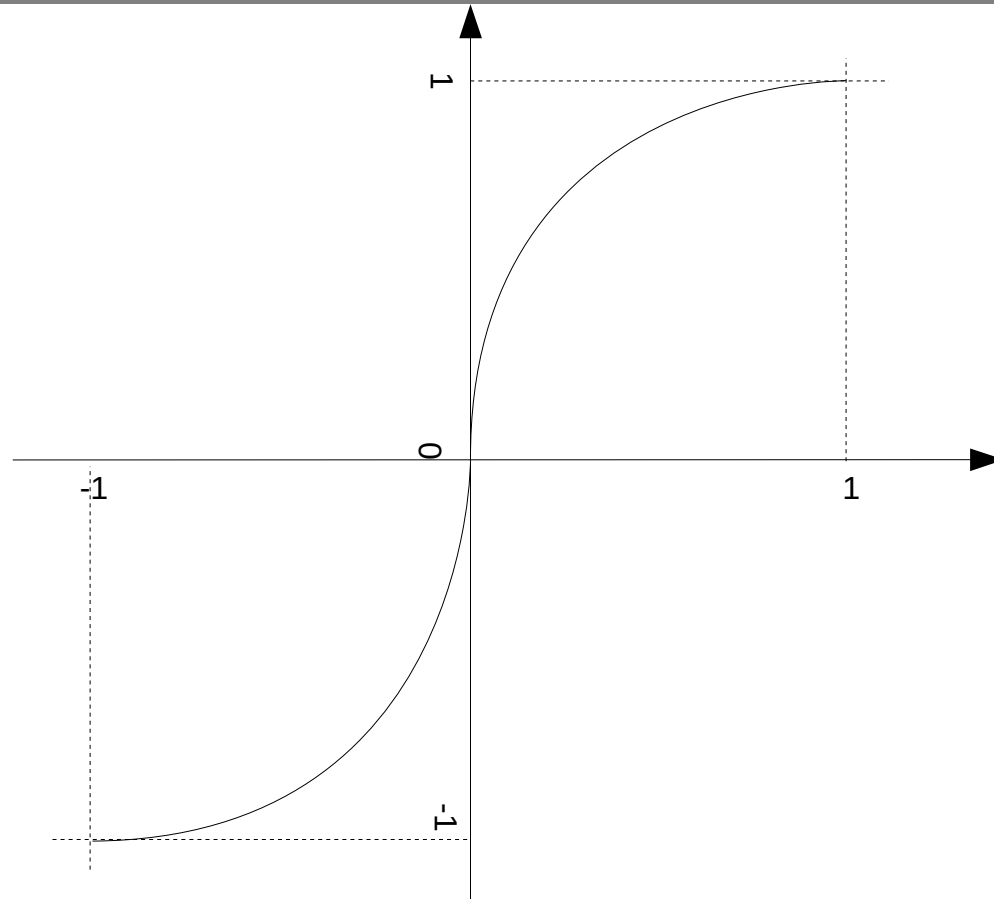


图 3-6 双指数函数曲线图

BP 网络是一种多层前馈神经网络，其主要特点是信号向前传递，误差反向传播。信号的前向传递过程是指输入信号从输入层经隐含层逐层处理，直至输出层。每一层神经元的状态只影响下一层神经元状态。如果输出

层得不到期望的输出，则转入反向传播，根据预测误差调整网络权值和阈值，从而使 BP 网络预测的输出不断逼近期望输出。BP 网络的拓扑结构如图 3-7 所示。

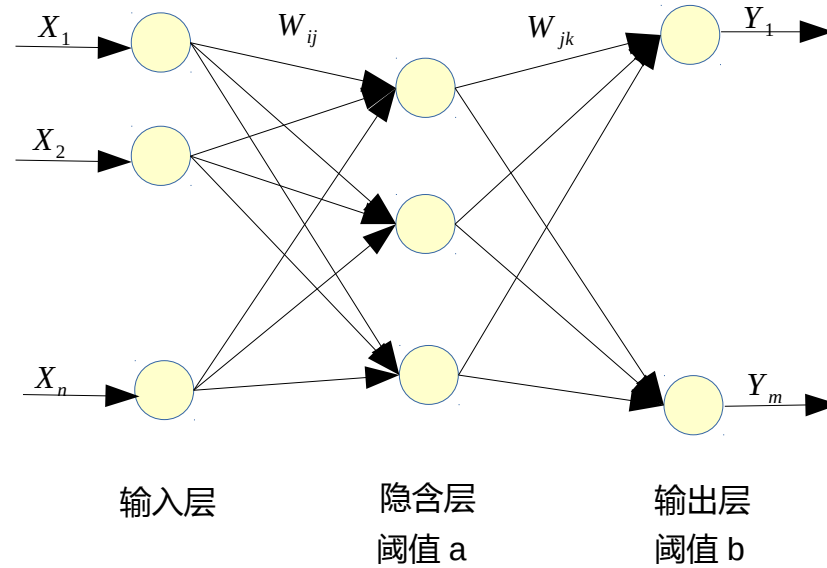


图 3-7 BP 网络拓扑结构图

BP 网络在进行预测或拟合之前，需要先进行网络训练，通过网络训练使网络具有联想记忆和预测能力。BP 网络的训练方法有多种，这里我们介绍一种误差反向传播的方法进行说明。当然，我们也可以利用下一节中介绍的遗传算法进行网络训练，这些内容在下一节会进行一些讲解，这里不再赘述。下面详细介绍误差反向传播法的步骤。

准备步骤：归一化输入向量 X 。对于输入向量而言，数据可能并不在一个数量级上，为了平衡每个数据的权重，将输入向量进行归一化处理，根据隐含层选择权的激励函数，可选择归一化到 $[0,1]$ 或者 $[-1,1]$ 之间。

更多内容易信用户请关注



，微信用户请关注



步骤一：初始化网络。根据系统输入输出序列 (X, Y) 确定网络输入层节点数 n ，隐含层节点数 l ，输出层节点数 m 。初始化输入层、隐含层和输出层神经元之间的连接权值 W_{ij} ， W_{jk} 。初始化隐含层阈值 a ，输出层阈值 b ，给定学习速率和神经元激励函数。

步骤二：隐含层输出计算。根据输入向量 X ，输入层和隐含层之间的连接权值 W_{ij} 以及隐含层的阈值 a ，计算隐含层输出 H 。

$$H_j = f\left(\sum_{i=1}^n w_{ij} x_i - a_j\right) \quad j=1, 2, \dots, l$$

其中 $f(x)$ 可以上面我们所述的三种激励函数之一。

步骤三：输出层输出计算。根据隐含层输出 H ，连接权值 W_{jk} ，以及输出层阈值 b ，计算 BP 网络的预测输出 O 。

$$O_k = \sum_{j=1}^l w_{jk} H_j - b_k \quad k=1, 2, \dots, m$$

步骤四：误差计算。根据网络预测输出 O 与期望输出 Y ，计算网络预测误差 e 。

$$e_k = Y_k - O_k \quad k=1, 2, \dots, m$$

步骤五：权值更新。根据网络预测误差 e 更新网络连接权值 W_{ij} ， W_{jk} 。

$$W_{ij} = W_{ij} + \partial H_j (1 - H_j) x_i \sum_{k=1}^m W_{jk} e_k \quad i=1,2,\dots,n, \quad j=1,2,\dots,l$$

$$W_{jk} = W_{jk} + \partial H_j e_k \quad j=1,2,\dots,l, \quad k=1,2,\dots,m$$

式中， ∂ 表示学习速率。

步骤六：阈值更新。根据网络预测误差 e 更新网络节点阈值 a ， b 。

$$a_j = a_j + \partial H_j (1 - H_j) \sum_{k=1}^m W_{jk} e_k \quad j=1,2,\dots,l$$

$$b_k = b_k + e_k \quad k=1,2,\dots,m$$

步骤七：判断算法迭代是否结束，没有结束返回步骤 2。结束则进行网络输出测试。

这里需要注意的是，网络输出的变量是归一化后的变量。需要采用归一化逆操作，将数据还原为输出数据。下面通过用 C 语言实现一个曲线拟合的例子来解释神经网络的实现过程。

假我们拟合 $y = x^2$ 函数的输入与输出。我们按照神经网络的具体设计步骤进行设计。

第一步，获取样本数据。输入样本数据取 $[0,1,2,3,4,5,6,7,8,9,10]$ ，对应的输出样本数据为 $[0,1,4,9,16,25,36,49,64,81,100]$ 。

第二步，构建神经网络。我们构建如下图所示的神经网络。

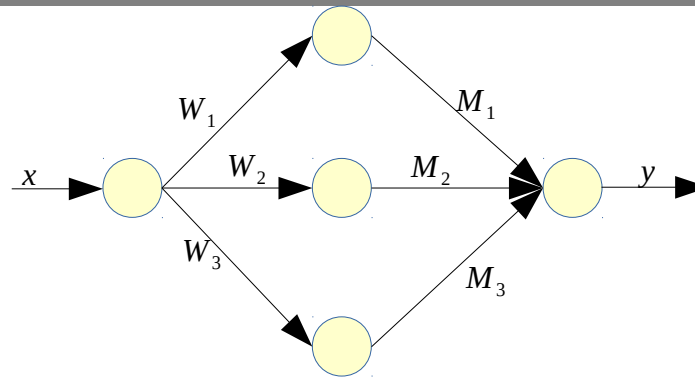


图 3-8 曲线拟合 BP 网络

第三步，确定激励函数。本例子中使用指数函数曲线。即激励函数取 $f(x) = \frac{1}{1+e^{-x}}$ 。

3.4 遗传算法及其 C 语言实现

遗传算法是模拟自然界生物遗传机制和进化论而产生的一种并行随机搜索最优化方法。它把“物竞天择、优胜劣汰、适者生存”的生物进化原理引入到优化参数，按照所选择的适应度函数，利用生物遗传中的选择、交叉和变异对个体进行筛选，使适应度值较好的个体基因得以保留，适应度值交叉的个体基因被淘汰，新的群体继承上一代的遗传信息，又优于上一代。如此反复循环，直到满足条件。遗传算法的基本操作有：

1. 选择操作。从旧群体中以一定的概率选择个体到新群体中，个体被选中的概率跟其适应度的值相关，个体的适应度越好，其被选中的概率越大；

2.交叉操作。从个体中选择两个个体，通过两个个体的染色体相互组合，来产生新的个体。交叉过程为从群体中任选两个染色体，随机选择一点或多点染色体位置进行交换。

3.变异操作。从群体任选一个个体，选择染色体的一点进行变异以产生新的个体。

3.5 人工智能与 PID

第四章 实例设计之电源仿真软件

4.1 电源控制系统模型

4.2 选择控制方法

4.3 实现与仿真

后记

记得我在读研究生的时候，导师将所有的学生分为理论组与实际组。理论组主要工作就是搜集国内外的优秀文章，然后研究其控制策略与控制方法，改进后发表 Paper。实际组的工作就是构建实际仿真与应用平台，主要就是一些实际的工程工作。当然，我是在实际组的。其实当时的情况是，理论组与实际组是存在认识矛盾的——理论组认为实际组工作没有理论水平，就是一些“打杂”的工作；实际组则认为理论组工作没有什么意义，就是在“忽悠”。当然参与这种无谓的争论当中的还有那时候的我。

更多内容易信用户请关注



，微信用户请关注



后来参加工作了，我很庆幸，因为当前国内的企业更愿意他的员工具备更强的实际操作能力，而非理论能力。所以我很轻松的找到一份各方面待遇还算不错的工作。随着工作内容的不断深入，各种各样的课题也是层出不穷，解决课题也变得越来越感觉力不从心了。看看当时的就职简历，精通 C 语言、精通嵌入式 Linux.....，这些在实际的工作中确实感觉远远不够，现实当中，很多课题并不是精通几项技能就可以解决的。此时我意识到，必须从更深层次上进行学习研究，才能够摆脱这种力不从心的状态。然后利用工作的业余时间，我又零零散散的学习了线性代数、控制工程、智能控制、系统辨识、信号处理等理论内容。学习配合实际工作的不断实验、验证，结合用户的实际体验与反馈，逐渐的形成了一套适合我自己工作的解决课题的思维模式。

走过这几年，现在回想起当前学生时代理论与实际的争论，实在是没有什么意义。理论真的是工程课题解决的核心，理论是一种数学化、经验化的解决方法。但是理论如果仅仅体现到文章上，那么距离实际的应用还是相差较远的，尤其是一些理论研究完全脱离了实际的背景，甚至研究的前提就是与实际背景完全不相符的，那么这样的理论成果没有什么现实意义（当然，也许有前瞻性的价值）。而当前所谓 C 语言、嵌入式之类的实际工作（我们姑且这么叫），实际上不能算是实际工作，只能算是完成实际工作的工具。能够熟练的掌握工具是非常了不起的事情，但是如果掌握工具的人没有思想，那么，最多也只能算是高级技工，是无法用手中的工具创造超脱于其他的兼备艺术性与实用性的作品的。

更多内容：

易信用户请关注



微信用户请关注

