

Awakenlion controller

林梓聰

2025 年 6 月 27 日

目录

§1 引言

此文档的初衷是让大家轻松地入门电控代码的控制器，缩短大家从理论到实践的时间。文档将会结合信号与系统和自动控制原理的相关知识，为大家讲解我们战队所需要的控制器，以及如何设计控制器。如果觉得作者本人写的不够详细或者有失偏颇，本人会在每一个知识点的后面附带上我学习过的网址，大家可以搜索着来学习。

本文档还会手把手教你如何操作，即使不看理论知识也可以按照步骤来直接得到结果。

§2 理论知识

以下是设计一个控制器所需要用到的理论知识，知识深度仅停留在工程应用的层面，不需要深究其数学原理，但是想要看懂还是需要基本的高数知识。以下的知识只需要将他们看作工具就好了，我们最终的目的就是得到传递函数，依据传递函数来设计所需要的控制器。我尽量会以是什么，怎么用的框架来解释下面的知识，其更深层的含义我会附上链接。

2.1 线性时不变系统(LTI)

为什么我们要先讲线性时不变系统呢？因为我们战队的代码所设计和分析的控制器都是基于线性时不变系统来研究的。

实际上线性时不变系统是一个理想化的数学模型，在生活中基本不存在完全符合线性时不变系统定义的系统。主要是以下两个原因：

1. 线性系统的限制:

- 许多系统在小信号范围内表现出线性特性，但在大信号条件下会出现非线性失真，如放大器在信号过强时的饱和现象。
- 机械系统如弹簧-质量系统在小振幅下可近似为线性，但超过弹性限度时会表现出非线性行为

2. 时不变性的挑战:

- 理想的时不变系统要求其参数不随时间变化，但在现实中，环境因素如温度、湿度可能影响系统参数，导致时变性。
- 电子元件可能因老化或温度变化而改变特性，从而引入时变性。

但因为LTI模型简单易于处理，其分析方法（如傅里叶变换、拉普拉斯变换、卷积等）是通用的，所以我们在一定合理的范围和时间内，将系统近似为LTI系统。

即使是遇到完全是非线性和时不变的系统我们都会尽量将其线性化（如工作点线性化、分段近似等）和时不变近似（如慢变参数近似、时间平均化等），以此来简化我们的分析。

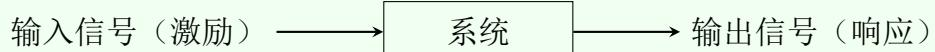
OK, 了解完上面的原因后，我们来真正的开始学线性时不变系统。

2.1.1 系统

什么是系统？

系统是指若干关联的事物组合而成具有特定功能的整体，系统可以是我们的云台，可以是一个电机，也可以是一个底盘。

系统的作用：对输入信号进行加工和处理，将其转换为输出信号
表示如下：



下面请记住这个重要的概念，

动态系统：系统的响应不仅与激励有关 $f(\cdot)$ 而且和过去的状态 $x_{(0)}$ 有关。含有记忆元件（电容、电感等）的系统就是动态系统。

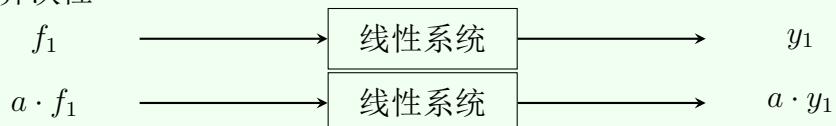
- $y(\cdot)$ 完全响应:零状态响应+零输入响应
- $y_{zs}(\cdot)$ 零状态响应：系统在初始时刻没有储能，所有响应均由输入信号引起。
- $y_{zi}(\cdot)$ 零输入响应：零输入响应完全由系统的初始状态决定，与输入信号无关。（如在电路系统中，电容一开始就储存了能量，这个时候即使没有输入信号也会有对应的响应，就像高中学的LC震荡电路一样）

2.1.2 线性系统

什么是线性系统

线性系统的充要条件为系统同时满足可加性和其次性

1、齐次性



思考一下：如果一个系统为 $y = ax + b$ 那么它是不是线性系统呢？

2、可加性

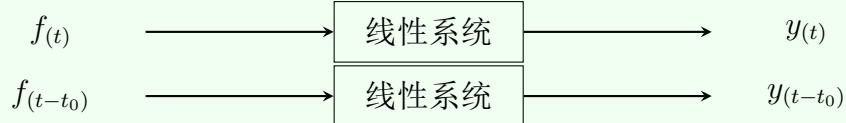


注： f_1 为激励， y_1 为响应，且线性系统都为同一个系统

2.1.3 时不变系统

什么是时不变系统

简单来说就是 $f_{(t)}$ 延迟多少， $y_{(t)}$ 就延迟多少



注： $f_{(t)}$ 为激励， $y_{(t)}$ 为响应，且线性系统都为同一个系统

如何判断一个系统是不是时不变呢，我们可以设一个延时激励，带入系统得到一个响应，然后再对原来响应延时，判断两者是否相等，稍微给个题目大家看看：

例：判断系统是否是时不变系统 $y(t - t_0) = tf(t)$

令 $g(t) = f(t - t_d)$, $T[in]$ 为激励信号输入后系统的响应

$$T[g(t)] = t \cdot g(t) = t \cdot g(t - t_d) \quad (1)$$

$$y(t - t_d) = (t - t_d) \cdot f(t - t_d) \quad (2)$$

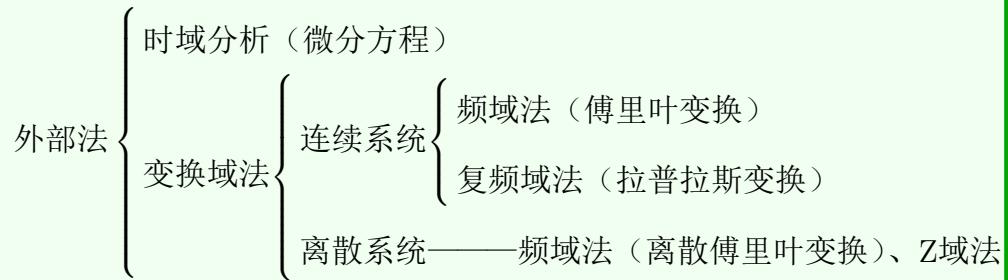
对比（1）、（2）式可得出，（1）式是延时激励，（2）式是原本的系统响应延时，两式不相等，所以该系统不是时不变系统。

2.1.4 线性时不变系统（LTI）

什么是LTI呢

显而易见，一个系统同时满足是线性和时不变性就是LTI系统了

LTI系统分析方法 $\begin{cases} \text{输入输出法 (外部法)} \\ \text{状态变量法 (内部法)} \end{cases}$



注：本文主要讲解外部分析法，内部法涉及现代控制理论可以自行了解

2.2 微分方程

了解了什么是线性时不变系统后，现在我教你们怎么在时域上分析连续的系统，这里我只会教公式解法而不会特别具体，因为等我们学习到了频域之后，我们基本上都会用频域的知识去分析系统

微分方程的经典解法

这下面的系统方程可以理解为这是我们分析系统的物理特性（系统建模）所列出来的方程

$$\begin{aligned} & y_{(t)}^{(n)} + a_{n-1} \cdot y_{(t)}^{(n-1)} + \dots + a_1 \cdot y_{(t)}^{(1)} + a_0 \cdot y(t) \\ & = b_m \cdot f_{(t)}^{(m)} + b_{m-1} \cdot f_{(t)}^{(m-1)} + \dots + b_1 \cdot f_{(t)}^{(1)} + b_0 \cdot f(t) \end{aligned}$$

注： $y_{(t)}^{(n)}$ 为 y 的 n 阶导

最后我们要求的是：

$$y(t) \text{ (完全解)} = y_h(t) \text{ (齐次解)} + y_p(t) \text{ (特解)}$$

$$\textcircled{1} \text{ 先求齐次解: } y_{(t)}^{(n)} + a_{n-1} \cdot y_{(t)}^{(n-1)} + \dots + a_1 \cdot y_{(t)}^{(1)} + a_0 \cdot y(t) = 0$$

$$\text{齐次解的特征根为: } \lambda^n + a_{n-1} \cdot \lambda^{n-1} + \dots + a_1 \cdot \lambda + a_0 = 0$$

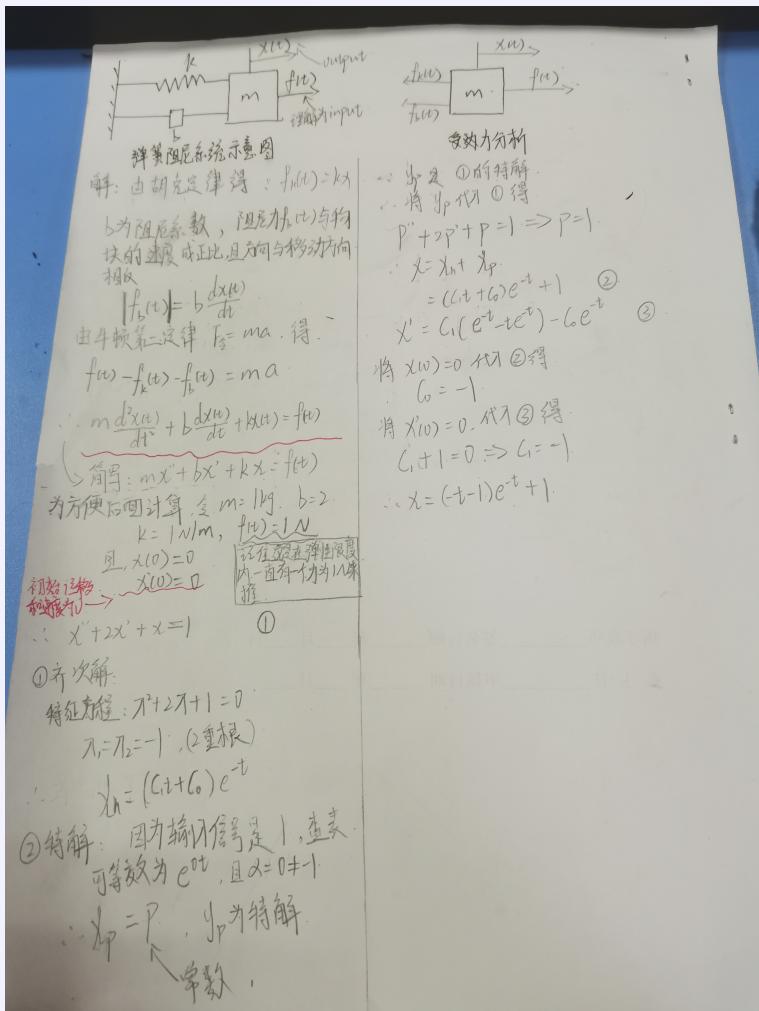
特征根 λ	齐次解 $y_h(t)$
单实根	$C e^{\lambda t}$
2重根	$(C_1 t + C_0) e^{\lambda t}$
一对共轭复根 $\lambda_{1,2} = \alpha \pm j\beta$	$e^{\alpha t} [C \cos(\beta t) + D \sin(\beta t)]$

②再求特解：

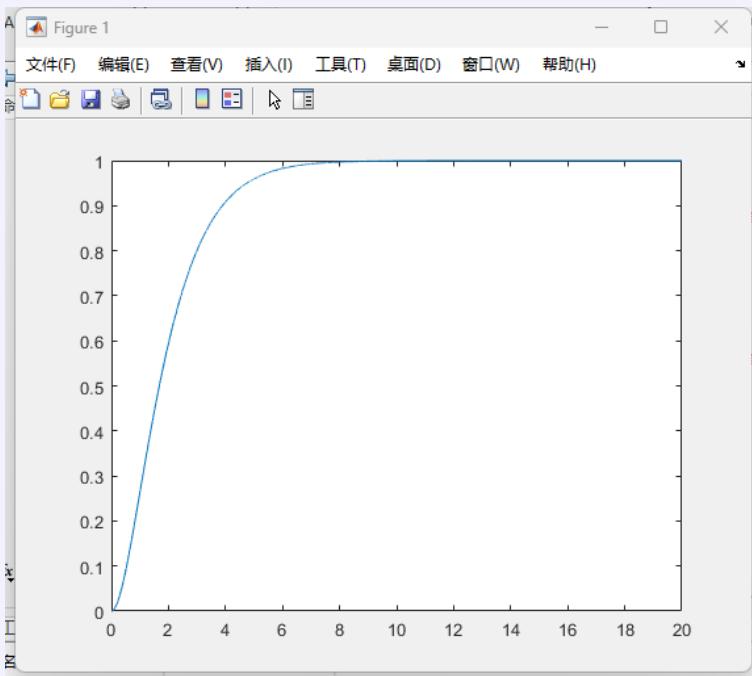
表2-2 不同激励所对应的特解

激励 $f(t)$	特解 $y_p(t)$
t	$P_1 t + P_0$ $t \cdot (P_1 t + P_0)$
$e^{\alpha t}$	$P e^{\alpha t}$ $(P_1 t + P_0) e^{\alpha t}$
$\cos(\beta t)$ 或 $\sin(\beta t)$	$P \cos(\beta t) + Q \sin(\beta t)$ 或 $A \cos(\beta t - \theta)$, 其中 $A e^{j\theta} = P + jQ$

看完上面的东西之后大家可能蒙蒙的，接下来我会完整地分析一个经典的二阶系统——弹簧质量阻尼系统



再看看最后计算的结果



从图像看出，用恒定1N的力去推物块，他会在9s左右到达1m处，然后到达二力平衡，一直停在1m处，

建议大家试试在此系统上用不同的系数（改变 b 、 m 、 k ），不同的输入看看得到的结果是什么，自己算一遍会有不一样的收获。

在现实生活中不一样的系统需要对应的系统建模方法，比如电路系统需要用KCL、KVL等，热力学需要 $PV=nRT$ 、卡诺循环等。这些在你们需要的时候可以自行去探索，这里就不过多赘述了

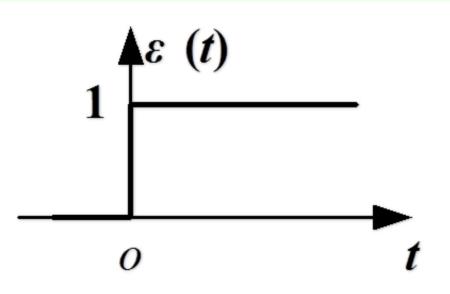
2.3 卷积

2.3.1 基本信号及其响应

在讲卷积之前，我需要引入两个基本信号，以此来推导卷积的公式，同时这两个基本信号也非常重要，大家一定要熟悉。

阶跃函数

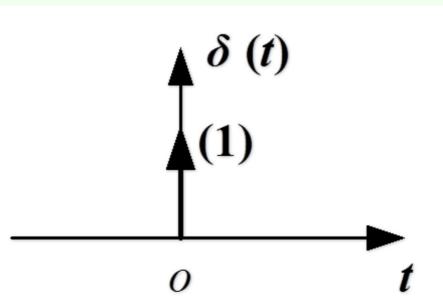
$$\varepsilon(t) = \begin{cases} 0 & , t < 0 \\ 1 & , t > 0 \end{cases}$$



阶跃函数是物理系统无法实现的，我们稍微想想就知道了，在生活中什么数值的上升下降都有一个过程比如温度：从1~100°C，在你对水加热的过程中，水肯定是每一个温度都会经历，而不是直接从1到100的直接跳变。

冲激函数

$$\varepsilon'(t) = \delta(t) = \begin{cases} 0 & , t \neq 0 \\ \int_{-\infty}^{+\infty} \delta(t) dt = 1 \end{cases}$$



$\delta(t)$ 是我们人为定义的一个函数，他是由 $\varepsilon(t)$ 求导可得。

可以理解为在 $\delta(t)$ 冲激函数可以理解为一个在 $t=0$ 处无限高、无限

窄，但面积为 1 的脉冲

冲激函数的性质：

- 取样性质： $\int_{-\infty}^{+\infty} \delta(t)\varphi(t) dx = \varphi(0)$ 可以自己想想为什么
- 缩放性： $\delta(at) = \frac{1}{|a|}\delta(t)$
- 奇偶性： $\delta(t) = \delta(-t)$ (偶函数)

基本信号的响应

就一个很简单的例子，我上面所分析的弹簧质量阻尼系统，我的输入就是一个阶跃信号，而得到的响应就是我们分析求的 $y(t)$ ，也就是阶跃响应。

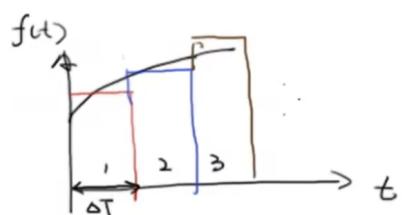
那如果我们想给一个冲激信号呢？你可以想象我们踢这个物块一脚，这一脚的力度很大，且脚和物块接触的时间无限短、趋近于0，那么你的输入就接近于一个冲激信号，你们可以尝试输入一个冲激信号看一看能不能求出来他的冲激响应(求的时候记住 $\varepsilon'(t) = \delta(t)$)

2.3.2 卷积的理解

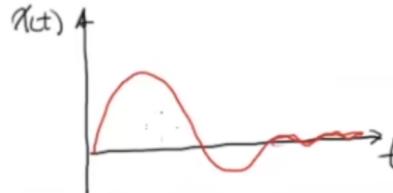
什么是卷积

卷积的本质：信号的分解。

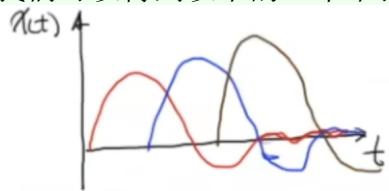
拿我们上面的弹簧阻尼质量系统来举例，假设我们现在的输入信号如下的黑色曲线，然后我们简单的把其离散化(此时我们只看那3个矩形波，不需要看曲线)



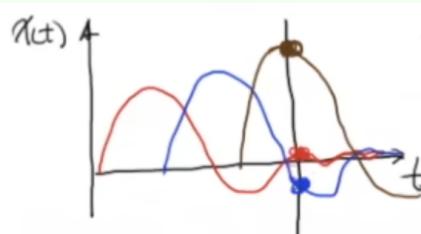
然后，我们假设只输入第一个矩形波的响应如下：



那么根据LTI系统的齐次性和时不变性，我们将三个输入信号看作单独作用于这个系统，我们可以得到以下的三个单独的响应

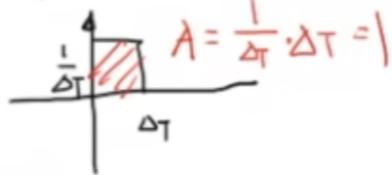


知道了三个信号单独作用的响应后，我们试着将他们一起作用在这个弹簧阻尼系统上（这里的“一起”不是说同时作用的意思，而是说，利用LTI系统的可加性，对t时刻之前的输入响应进行求和）。如下图，在画线的那个时间点我们的响应是三个单独信号在t时刻的叠加，所以 $x(t) = \Sigma$ (系统对t时刻前输入响应的和)。



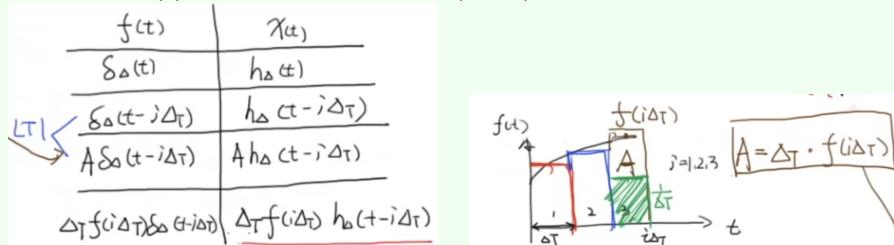
试想一下，如果我们将 ΔT 趋近于0，那么我们离散的矩形波就变成了一条直线且有无数条，而且重新接近回我们的曲线，此时我们的求和就成了积分（函数的面积），为了实现这一目的我们先将输入信号先单位化，就是将矩形波化简为下面这个 $\delta_{\Delta}(t)$ 函数的A倍：

思考一下当 ΔT 趋近于0,这个函数是什么函数



看下面左图，对于输入信号为 $f(t)$ ，我们的响应是 $h_\Delta(t)$ ，那么由LTI的性质我们可以推出第二和第三条式子（其实就是把前三幅图用公式给你们描述一遍，大家可以翻前去看看）

这里的 A 是代表矩形波的面积（看右图，底×高），也表示 A 个单位的 $\delta_\Delta(t)$ 。把 $A = \Delta T f(i\Delta T)$ 带入式三，最后得式四：



上面说过，求 t 时刻的响应就是将 t 时刻之前的输入响应进行求和：

$$t = i\Delta T \text{ 时: } x(t) = \sum_{i=0}^i \Delta T f(i\Delta T) \cdot h_\Delta(t - i\Delta T),$$

$$\text{令 } \lim_{\Delta T \rightarrow 0}, h_\Delta(t) \rightarrow h(t), \Delta T = d\tau, i\Delta T = \tau,$$

$$\begin{aligned} x(t) &= \int_0^t f(\tau) h(t - \tau) d\tau \\ &= f(t) * h(t) \end{aligned}$$

上面的第二条式子就记作 $f(t)$ 和 $h(t)$ 的卷积， $x(t)$ 则是他们通过卷积运算得出来的一个新函数。

随便解个密，当 $\lim_{\Delta T \rightarrow 0}$ 时， $\delta_\Delta(t) \rightarrow \delta(t)$ ，所以 $h(t)$ 是冲激响应。由此可以看出对于LTI系统，冲激响应可以完全定义系统（即系统冲激响应等于系

统)，也就是说只要能得出冲激响应，就可以知道系统长什么样。所以系统的输出等于输入卷积冲激响应（系统本身）

我们现在得出来了卷积的表达式，我们试试从另外一个角度来看待卷积，仔细观察卷积的表达式 $x(t) = \int_0^t f(\tau)h(t-\tau) d\tau$ 。

将 $f(\tau)$ 看作一个函数，则 $h(\tau)$ 是先翻转成 $h(-\tau)$ ，然后再平移到 t ，得到 $h(t-\tau)$ ，并且将从0平移到 t 这过程中和 $f(\tau)$ 一一对应的乘积进行求和。

所以，我们可以简单的把卷积理解为它将两个函数结合在一起，产生第三个函数。这个新函数可以看作是一个函数在另一个函数上的“滑动窗口”积分。

不懂的话就看下面第一条链接，回来后应该能大致理解我说的是什么意思了

由于本人对卷积的理解也不算特别深刻，而卷积又是一个很有趣的知识点，所以大家可以看下面链接的视频，加深大家对卷积的理解

<https://www.bilibili.com/video/BV1Vd4y1e7pj>。

<https://www.bilibili.com/video/BV1cs411W74f>。

2.4 拉普拉斯变换

前面我们运用时域分析对弹簧质量阻尼系统进行了分析，从这一节开始我们就开始运用频域的分析方法来解析系统，而拉普拉斯变换就是经典控制理论中很重要的一个数学工具，可以将时域转换为频域，从而降低分析难度。

2.4.1 频域和傅里叶变换

在学习拉普拉斯变换之前，大家需要先对频域有一个基本的认识，并且也简单的了解一下傅里叶变换，傅里叶变换如果想深入学习的话，可以自己查资料学习，傅里叶也是一个很重要的知识点，大家尽量去掌握。

什么是频域

在我们平常的生活中，我们习惯用时间来描述一件事情，如人的身高体重随时间的变化、上面我们以时间为自变量来描述的弹簧阻尼质量系统等，以时间作为参照来观察动态世界的方法我们称其为时域分析。

现在我们不用时域来分析事情，我们将坐标轴x轴的变量换成频率 ω ，我们用频率来分析和描述一件事，这就称为我们的频域分析。

知道了什么是频域，那么我们怎么样才能将时域的东西变成频域呢？这就要用到我下面说的傅里叶变换了

傅里叶变换

任何周期函数，都可以看作是不同振幅，不同相位正弦波的叠加。其定义式如下：

$$F(j\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t} dt$$

傅里叶变换需要 $f(t)$ 满足绝对可积的条件：

$$\int_{-\infty}^{\infty} |f(t)| dt$$

指数衰减函数，如 $f(x) = e^{-x}$ ；有界函数在有限区间上也绝对可以积，如 $f(x) = \sin(x)$ 在 $[0, \pi]$ 上绝对可积；而 $f(x) = 1$ 就不是绝对可积的

了。这些大家都可以代入到定义式自己验证一下。

大家可以点下面的链接去更深入的了解傅里叶变换 深入浅出的讲解傅里叶变换

2.4.2 拉普拉斯变换定义

什么是拉普拉斯变换

对一个函数 $f(t)$ 做拉普拉斯变化，可以将其从时域(t)转换到复数域(s)，它的定义为

$$\mathcal{L}[f(t)] = F(s) = \int_0^{\infty} f(t)e^{-st}dt$$

其中， $s = \sigma + j\omega$, 是一个复数

举个例子， $\mathcal{L}[e^{-at}] = \frac{1}{s+a}$

证：

$$\begin{aligned}\mathcal{L}[e^{-at}] &= \int_0^{\infty} e^{-at}e^{-st}dt \\ &= \int_0^{\infty} e^{-(a+s)t}dt \\ &= -\frac{1}{s+a}e^{-(a+s)t} \Big|_0^{\infty} \\ &= \frac{1}{s+a}\end{aligned}$$

基本上，如果你需要进行拉普拉斯变换的话都可以这样直接套公式，大家也可以求一下其他函数的拉普拉斯变换熟悉一下。但是如果每次都需要求的话就会显得十分麻烦，所以给大家一个表，一些基本函数的拉普拉斯变换都在里面，大家要用的时候可以直接查表。

2.4.3 常见的拉普拉斯变换公式

1		
原函数	拉普拉斯变换	收敛域(实部 σ)
$f(t)$	$F(s) = \mathcal{L}[f(t)]$	
$\delta(t)$	1	$-\infty < \sigma < \infty$
1	$\frac{1}{s}$	$\sigma > 0$
t^n	$\frac{n!}{s^{n+1}}$	$\sigma > -a$
e^{-at}	$\frac{1}{s + a}$	$\sigma > -a$
$\sin(at)$	$\frac{a}{s^2 + a^2}$	$\sigma > 0$
$\cos(-at)$	$\frac{s}{s^2 + a^2}$	$\sigma > 0$

2.4.4 拉普拉斯收敛域

看到上面那个表，肯定会有人疑惑收敛域是什么东西，其实上文我们讲傅里叶变换的时候我们也有涉及到一点，我们说过傅里叶变换需要满足这个函数是绝对可积的，而拉普拉斯变换通过引入衰减因子 $e^{-\sigma t}$ ，拓展了其适用性，使原来不满足傅里叶变换条件的信号也能进行变化。

我们来举个例子，

$$F(S) = \mathcal{L}[e^{-at}] = \int_0^\infty e^{-at} e^{-st} dt$$

将 $s = \sigma + j\omega$ 代入上式，得

$$F(S) = \mathcal{L}[e^{-at}] = \int_0^\infty e^{-(a+\sigma)t} \cdot e^{-j\omega t} dt$$

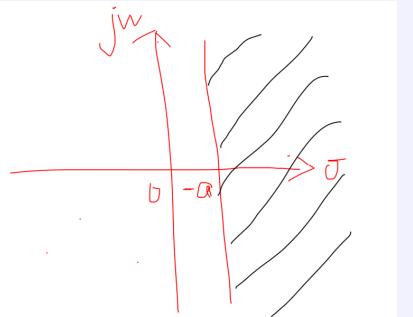
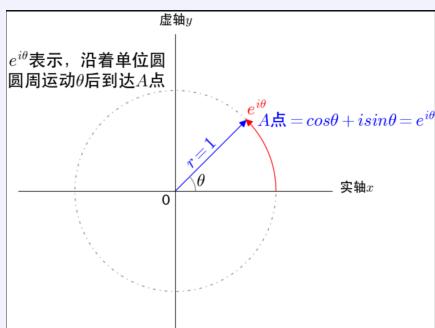
上面式子中的积分是由两部分相乘而得。首先看 $e^{-j\omega t}$ 这一项是一个

复数。根据欧拉公式得 $e^{-j\omega t} = \cos\omega t - j\sin\omega t$, 所以随着t的增加在复平面上做顺时针的运动, 而他的幅值 $|e^{-j\omega t}|$ 是恒定不变且等于1, 如下左图(对于欧拉公式得证明这个大家可以去查资料, 或者我记得的话会在文章最后面写一个附录来证明)。所以用它乘以 $e^{-(a+\sigma)t}$ 这一项不会对积分的收敛产生影响。这也可以从另外一个角度看, 根据欧拉公式, $e^{-j\omega t}$ 仅仅是引入了正弦和余弦量(引入振动), 而振动又会有正有负, 不会在单一方向上增加或减少。因此要满足函数绝对可以积, e 的指数部分要小于0, 即

$$-(a + \sigma) < 0$$

$$\sigma > -a$$

所以, $\sigma > -a$ 是 $\mathcal{L}[e^{-at}] = \frac{1}{s+a}$ 的收敛域, 如下右图



虽然拉普拉斯变换拓展了傅里叶变换的适用性, 但是也不是所有的函数都可以进行拉普拉斯变换的。拉普拉斯变换要求函数在某个区间内绝对可积。如果函数在 $t \rightarrow \infty$ 时振荡剧烈或发散, 可能无法满足这一条件, 如 $f(t) = \sin(t^2)$ 震荡过于剧烈

2.4.5 拉普拉斯性质

线性性质、尺度变换特性

1、线性性质

若 $\mathcal{L}[f_1(t)] = F_1(s)$, $\mathcal{L}[f_2(t)] = F_2(s)$

则 $\mathcal{L}[a_1 f_1(t) + a_2 f_2(t)] = a_1 F_1(s) + a_2 F_2(s)$

2、尺度变换

若 $\mathcal{L}[f(t)] = F(s)$

则 $\mathcal{L}[f(at)] = \frac{1}{a} F\left(\frac{s}{a}\right)$

时移特性、复频移特性

1、时移特性

若 $\mathcal{L}[f(t)] = F(s)$

则 $\mathcal{L}[f(t - t_0)] = F(s)e^{-st_0}$

2、复频移特性

若 $\mathcal{L}[f(t)] = F(s)$

则 $\mathcal{L}[f(t) e^{s_0 t}] = F(s - s_0)$

微积分性质

1、微积分性质

若 $\mathcal{L}[f(t)] = F(s)$

则 $\mathcal{L}[f'(t)] = sF(s) - f(0_-)$, $\mathcal{L}[f''(t)] = s^2 F(s) - sf(0_-) - f'(0_-)$

若 $f(t)$ 是因果系统:

则 $\mathcal{L}[f^{(n)}(t)] = s^n F(s)$

什么是因果系统呢? 系统在某时刻 t 的输出仅取决于时刻 t 及在 t 之前的输入, 而与 t 之后的输入无关, 简单来说就是我打你一巴掌让你脸疼, 肯定是我现在或者之前打的你, 不可能是未来的我打你导致你现在疼。我们生活中的系统都是因果系统所以上面的式子很重要

2、积分性质

若 $\mathcal{L}[f(t)] = F(s)$

$$\text{则 } \mathcal{L}[\int_0^t f(x)dx] = \frac{1}{s}F(s), \quad \mathcal{L}[(\int_0^t)^n f(x)dx] = \frac{1}{s^n}F(s)$$

终值定理

定义： $f(t)$ 当 $t \rightarrow \infty$ 存在，且 $\mathcal{L}[f(t)] = F(s)$, $\operatorname{Re}[s] > \sigma_0, \sigma_0 < 0$,

$$\text{则 } f(\infty) = \lim_{s \rightarrow 0} sF(s)$$

这个定理很有用，当你将系统进行拉普拉斯变换后，它可以利用终值定理判断该系统是否稳定，并且趋于哪个值。这个后面我在证明的时候会用上，大家可以留意一下。

上面的定理都可以代入拉普拉斯的定义进行证明，只是考大家的高数能力而已，在这我就不花过多的篇幅进行证明了。下面讲的卷积定理很重要，所以我会附上证明过程，而上面的性质大家只需背记即可。

卷积定理

$$\mathcal{L}[f(t) * g(t)] = F(s) \cdot G(s)$$

证：

$$\begin{aligned}\mathcal{L}[f(t) * g(t)] &= \int_0^\infty (f(t) * g(t)) e^{-st} dt \\ &= \int_0^\infty \left(\int_0^t f(\tau)g(t-\tau) d\tau \right) e^{-st} dt\end{aligned}$$

这是一个二重积分，可以用交换积分顺序与上下限的方式来进行化简

$$\int_0^\infty \left(\int_0^t f(\tau)g(t-\tau) d\tau \right) e^{-st} dt = \int_0^\infty \int_\tau^\infty f(\tau)g(t-\tau)e^{-st} dt d\tau \quad (\text{a})$$

对 $\int_\tau^\infty f(\tau)g(t-\tau)e^{-st} dt d\tau$ 进行使用换元法，令 $t - \tau = u$, 得（使用换

元法的原因是我们需要将函数里面的变量分开，进行解耦）

$$t = u + \tau$$

$$\Rightarrow dt = du + d\tau = du$$

$$\Rightarrow \int_{\tau}^{\infty} f(\tau)g(t - \tau)e^{-st} dt d\tau = \int_0^{\infty} f(\tau)g(u)e^{-s(u+\tau)} du d\tau \quad (\text{b})$$

这里积分上下限变的原因是原本 t 的积分范围是 (τ, ∞) ，而 $u = t - \tau$ ，所以很自然的 u 的积分范围就是 $(0, \infty)$

将(b)式代入(a)式得

$$\begin{aligned} \int_0^{\infty} \int_{\tau}^{\infty} f(\tau)g(t - \tau)e^{-st} dt d\tau &= \int_0^{\infty} \int_0^{\infty} f(\tau)g(u)e^{-s(u+\tau)} du d\tau \\ &= \int_0^{\infty} f(\tau)e^{-s\tau} d\tau \cdot \int_0^{\infty} g(u)e^{-su} du \\ &= F(s)G(s) \end{aligned}$$

上面在学习卷积的时候我们知道了系统的输出等于输入卷积冲激响应，但是如果每次都需要积分的话运算就会十分复杂，这也是我们将时域分析转换为频域分析的一个原因之一。所以，当我们想求一个系统的输出时，我们就会很自然的将它转换成频域，由卷积运算变为乘法运算降低我们的运算难度。

2.4.6 拉普拉斯逆变换

我们现在知道怎么样将时域转换为频域，那怎么将其频域转换回时域呢，毕竟频域只是我们分析的一个方法，我们更加擅长于看时域的东西。所以，当我们利用频域分析完了一个系统之后，得到的结果我们应该学会转换回时域。

将拉普拉斯逆变换主要有四个方法，①定义，②查表，③性质，④分式展开。定义是利用拉普拉斯反变换的定义式进行计算，由于计算复杂我们一般都不会用定义式来算。查表就是那些直接可以在我上面提供的拉普拉斯变换公式表里面可以查到的，所以主要教大家如何利用拉普拉斯的性质和分式展开来进行逆变换。

拉普拉斯逆变换定义式

$$f(t) = \mathcal{L}^{-1}[F(s)] = \frac{1}{2\pi j} \int_{\sigma-j\infty}^{\sigma+j\infty} F(s)e^{st}ds$$

其中： σ 是实常数

这里只是简单提一下，让大家知道有这个东西就行，很少用

下面就通过一些例题给大家讲解一下怎么样使用性质和分式展开进行拉普拉斯逆变换。

当我们拿到一个s域的函数 $F(s)$ 时，我们需要确定我们展开的式子是真分式，如果不是的话就化成真分式，这里的真分式是指分母最高幂次大于分子。

$$F(s) = P(s) + F_1(s)$$

其中 $P(s)$ 为整式， $F(s)$ 为真分式

① 先解 $P(s)$ ，例如当 $P(s) = a_2s^2 + a_1s + a_0$

查表得， $\mathcal{L}[\delta(t)] = 1$ ，由微分性质得，

$$P(s) = a_2\delta''(t) + a_1\delta'(t) + a_0\delta(t)$$

② 再求 $F_1(s)$, 将 $F_1(s)$ 进行分解

$$F_1(s) = \frac{B(s)}{A(s)} = \frac{k_1}{s - p_1} + \frac{k_2}{s - p_2} + \dots + \frac{k_i}{s - p_i}$$

对每个k进行求解,

$$k_i = (s - p_i)F(s) |_{s=p_i}$$

再对每个分式用性质和查表进行求逆

$$\mathcal{L}^{-1}\left[\frac{k_i}{s - p_i}\right] = k_i e^{p_i t} \varepsilon(t)$$

$A(s)$ 称为 $F_1(s)$ 的特征多项式, $A(s) = 0$ 为特征方程, 它的根为特征根, 特征根 p_i 也称为 $F(s)$ 的极点。可能有些迷, 下面直接解一道题给大家看一看。

例: $F(s) = \frac{s^2 + 5}{s^2 + 5s + 4}$, 求逆变换

由式子我们得出 $F(s)$ 为假分式, 用长除法将式子化为真分式得

$$\begin{aligned} F(s) &= P(s) + F_1(s) \\ &= 1 + \frac{-5s + 1}{s^2 + 5s + 4} \end{aligned}$$

$P(s)$ 由查表得,

$$p(t) = \mathcal{L}^{-1}[P(s)] = \mathcal{L}^{-1}[1] = \delta(t)$$

对 $F_1(s)$ 进行分解：

$$\begin{aligned}F_1(s) &= \frac{-5s + 1}{s^2 + 5s + 4} \\&= \frac{-5s + 1}{(s + 1)(s + 4)} = \frac{k_1}{(s + 1)} + \frac{k_2}{(s + 4)}\end{aligned}$$

$$\begin{aligned}k_1 &= (s + 1)F(s) |_{s=-1} \\&= \frac{-5s + 1}{(s + 4)} |_{s=-1} \\&= 2\end{aligned}$$

同理可得， $k_2 = -7$

$$\text{所以, } F_1(s) = \frac{2}{(s + 1)} + \frac{-7}{(s + 4)}$$

$$f_1(t) = \mathcal{L}^{-1}[F(s)] = (2e^{-t} - 7e^{-4t})\varepsilon(t)$$

所以，最后将两个式子加在一起就是我们所求的了

$$\begin{aligned}f(t) &= p(t) + f_1(t) \\&= \delta(t) + (2e^{-t} - 7e^{-4t})\varepsilon(t)\end{aligned}$$

上面的是单根的情况，如果出现了重根我们又应该怎么去算呢？

重根的解法：

$$F(s) = \frac{A(s)}{B(s)} = \frac{K_{11}}{(s - p_1)^r} + \frac{K_{12}}{(s - p_1)^{r-1}} + \dots + \frac{K_{1r}}{(s - p_1)}$$

利用下面公式可以直接得到每个系数

$$K_{11} = [(s - p_1)^r F(s)]|_{s=p_1}$$

$$K_{12} = \frac{d[(s - p_1)^r F(s)]|_{s=p_1}}{ds}$$

$$K_{1r} = \frac{1}{(r-1)!} \frac{d^{r-1}}{ds^{r-1}} [(s - p_1)^r F(s)]|_{s=p_1}$$

给大家一个具体的例题来看看是怎么用的：

$$F(s) = \frac{s-2}{s(s+1)^3}, \text{ 求逆变换}$$

解：

$$\text{设 } F(s) = \frac{k_{11}}{(s+1)^3} + \frac{k_{11}}{(s+1)^2} + \frac{k_{11}}{(s+1)} + \frac{k_2}{s}$$

$$\text{令 } F_1(s) = (s+1)^3 F(s) = \frac{s-2}{s}$$

$$k_{11} = F_1(s)|_{s=-1} = \frac{s-2}{s}|_{s=-1} = 3$$

$$k_{12} = \frac{dF_1(s)}{ds}|_{s=-1} = \frac{s-(s-2)}{s^2}|_{s=-1} = 2$$

$$k_{13} = \frac{1}{2} \frac{d^2 F_1(s)}{ds^2}|_{s=-1} = \frac{1}{2} \cdot \frac{-4s}{s^4}|_{s=-1} = 2$$

$$k_2 = sF(s)|_{s=0} = \frac{s-2}{(s+1)^3}|_{s=0} = -2$$

$$\therefore F(s) = \frac{3}{(s+1)^3} + \frac{2}{(s+1)^2} + \frac{2}{s+1} - \frac{2}{s}$$

$$f(t) = \mathcal{L}^{-1}[F(s)] = (\frac{3}{2}t^2e^{-t} + 2te^{-t} + 2e^{-t} - 2)\varepsilon(t)$$

2.5 传递函数和系统的设计

当我们知道了什么是频域、什么是拉普拉斯变换之后，我们就可以开始学什么是传递函数了，它是我们经典控制理论的基础，可以理解为它是由我们时域上的系统通过拉普拉斯变换转换成频域之后的一种表达式。

2.5.1 传递函数

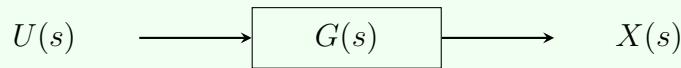
什么是传递函数

传递函数是描述线性系统动态特性的数学工具，主要用途是描述系统特性，分析系统响应，设计控制系统，判断系统稳定性

定义：在零初始条件下，系统输出的拉普拉斯变换与系统输入的拉普拉斯变换之间的比值，即

$$G(s) = \frac{X(s)}{U(s)}$$

上式的系统可以用如下框图表示



$$X(s) = \mathcal{L}[u(t) * g(t)] = U(s)G(s)$$

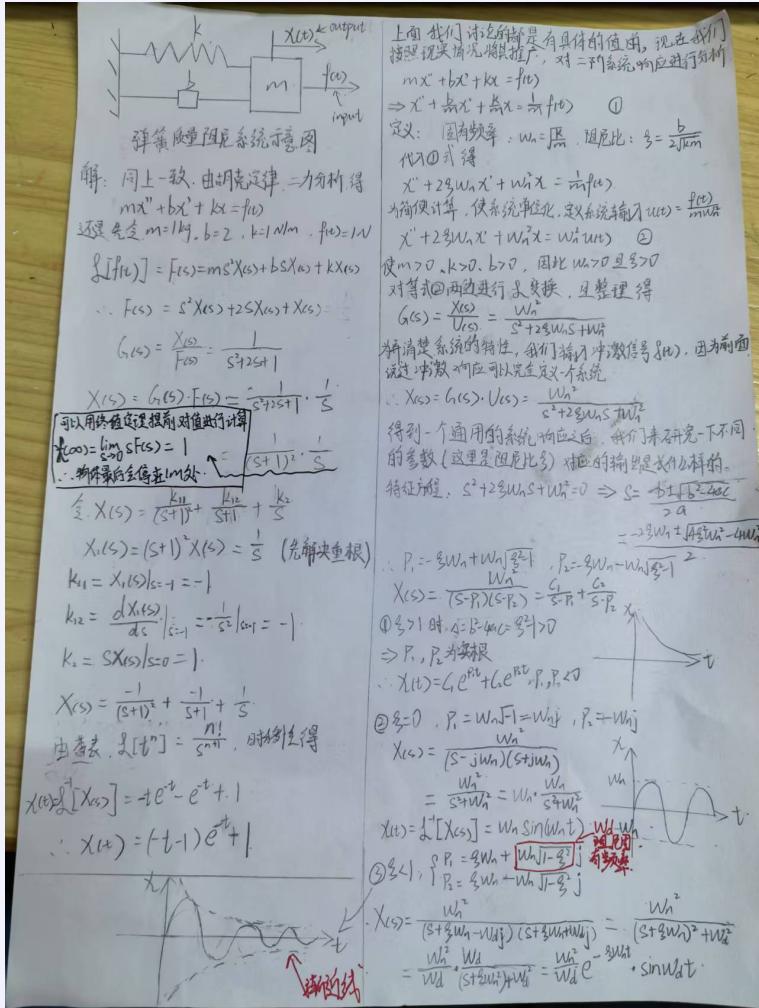
显而易见，我们将系统和输入之间的关系从卷积转换成了乘法

上面我说过一句话：冲激响应可以完全定义系统。在这里我们可以从频域的角度来看，令输入 $u(t) = \delta(t)$

$$\begin{aligned} X(s) &= U(s) \cdot G(s) = \mathcal{L}[\delta(t)] \cdot G(s) \\ &= G(s) \end{aligned}$$

这里可以更清楚的看到系统的冲激响应等于传递函数（系统本身）

现在我可以用频域分析给大家分析一下上面的弹簧质量阻尼系统了



对于不同的阻尼比，我们得到不同的系统冲激响应。

下面说的系统是对所有二阶系统都适用

- ①当 $\xi > 1$ 时, 系统称为过阻尼系统, 其响应不会出现震荡, 且快速收敛
- ②当 $\xi = 1$ 时, 系统称为界阻尼系统, 处于震荡的临界。
- ③当 $0 < \xi < 1$ 时, 系统称为欠阻尼系统, 其响应会不断地振动且幅度逐渐减少并最终停下来。
- ④当 $\xi = 0$ 时, 系统称为无阻尼系统, 他会一直振动下去, 且不会消耗

能量。

我们很容易看出对于系统的最终响应，表面上改变的是 $\xi = 0$ ，但究其原因是极点的改变，导致系统的响应变化。

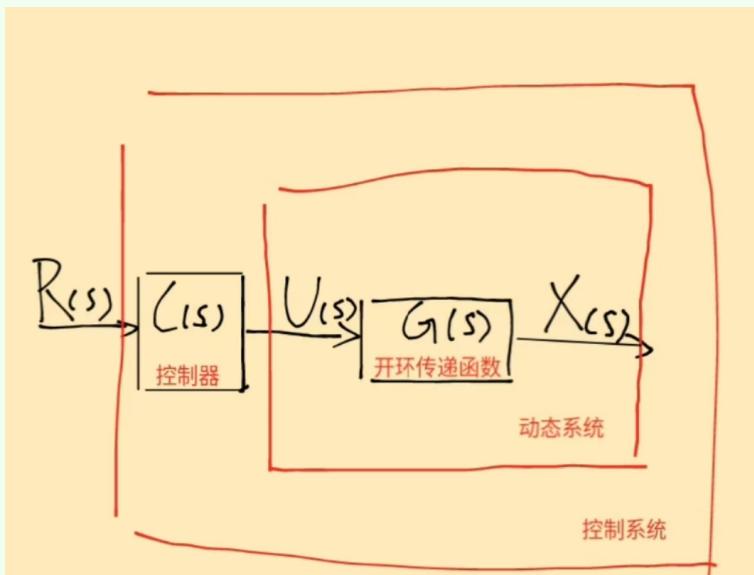
我们看到的系统基本是分式的形式，当 $\frac{k}{s-p}$ 进行拉式反变换的时候，得到的一定是 ke^{pt} 的形式，从这里我们可以看出系统是否震荡、是否收敛、收敛快慢、都会取决于极点（系统的整式部分不影响我上面说的）。这里我只是简单的铺垫一下，后面会有更详细的解释。

2.5.2 系统设计

大家上面看到的都是一个单纯的系统，是不包含控制器的，接下来我就给大家开始讲关于控制系统传递函数。

开环控制系统

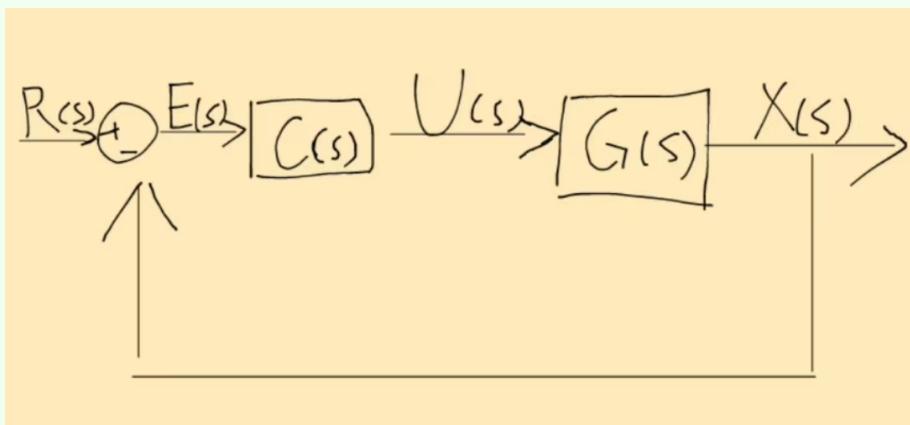
下图就是开环控制系统：其中 $R(s)$ 是参考值或目标值， $C(s)$ 是控制器，原动态系统的传递函数 $G(s)$ 被称为控制系统的开环传递函数。控制量是 $U(s)$ ，也就是原动态系统的输入。控制系统的输出等于原动态系统的输出 $X(s)$ 。



开环控制系统就是我们只管输入值，对输入值进行控制，达到我们的预设值。即使有误差我们也不管，也管不了，因为系统里面不存在反馈，控制器是不知道最后输入的结果的。

若将 $X(s)$ 反馈到输入端，则可以形成一个闭环控制系统

闭环传递函数



其中，参考值与输入之间的差称为误差， $E(s) = R(s) - X(s)$ ，其对应的时间函数是 $e(t) = r(t) - x(t)$ ，控制器 $C(s)$ 根据误差决定控制

量 $U(s)$ 。

根据传递函数的代数性质，可得

$$X(s) = U(s)G(s) = E(s)C(s)G(s)$$

将 $E(s) = R(s) - X(s)$ 带入上式得，

$$\begin{aligned} X(s) &= (R(s) - X(s))C(s)G(s) \\ \Rightarrow (1 + C(s)G(s))X(s) &= R(s)C(s)G(s) \\ \Rightarrow X(s) &= \frac{C(s)G(s)}{1 + C(s)G(s)} \end{aligned}$$

定义控制系统的闭环传递函数为

$$G_{cl}(s) = \frac{X(s)}{R(s)} = \frac{C(s)G(s)}{1 + C(s)G(s)}$$

所以我们可以得到一个简化后的闭环控制系统框图

$$R(s) \longrightarrow G_{cl}(s) = \boxed{\frac{C(s)G(s)}{1 + C(s)G(s)}} \longrightarrow X(s)$$

从这里我们也可以看到，我们设计和调整控制器根本上就是改变传递函数的极点和零点

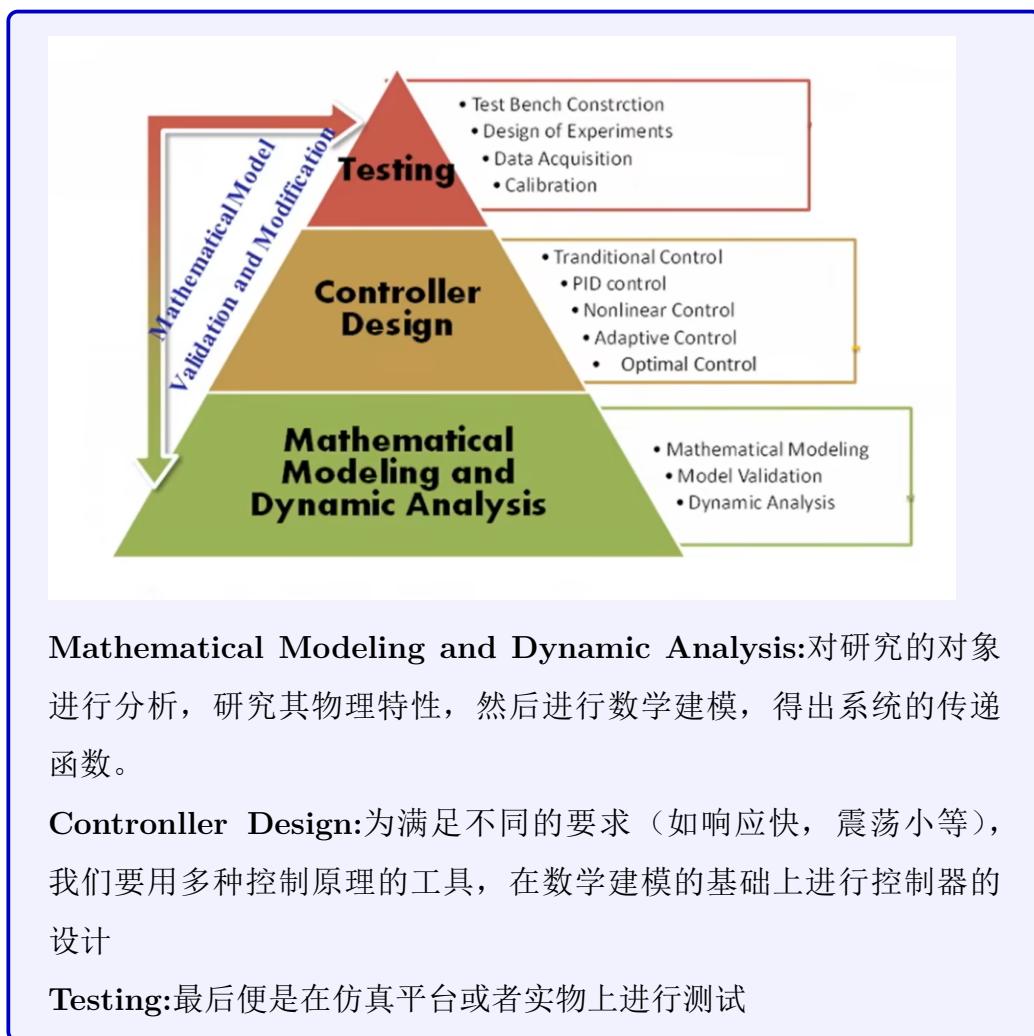
注意：此处的开环传递函数是针对闭环系统来说的，不是开环系统的传递函数。

由上面分析可知开环传递函数和闭环传递函数关系紧密，因此系统的开环频率特性很大程度上决定了系统的闭环性能。由于闭环系统包含反馈，校准过程较为复杂，而开环传递函数容易分析和设计，因此工程上常用开环传递函数来分析和设计控制器。

§3 实践知识

有了上面的基础知识之后，这一部分我会在实际的运用中，边教大家设计控制器边补全大家所需要的知识。接下来我们就是围绕着如何对我们的云台系统进行仿真和制作控制器。

对解决动态系统的控制问题我们可以从如下三个部分入手



3.1 系统传递函数

要对我们的云台进行控制器的设计首先我们需要先建模出一个系统 Yaw轴
云台的转角关于输入力矩是一个二阶的模型

$$J\ddot{\theta} + b\dot{\theta} = T$$

而Yaw轴云台的角速度关于输入力矩就是一个一阶的模型

$$J\dot{\omega} + b\omega = T$$

(其中J是云台转动惯量，b是阻尼系数，这二者可认为是常数，而T是我们控制输入的力矩，具体的推导过程就留给大家尝试，需要一定的物理知识)但是由于我们云台电机磨损情况不知、难以准确的得到转动惯量和阻尼系数,所以我们要借助Matlab的system identification工具进行系统识别。

3.1.1 速度环闭环识别

第①步：扫频

首先我们需要扫频，扫频就是说我们系统输入的信号需要从1-40Hz增加，从而得到每个频率对应的响应。

我们需要在云台代码中加入以下的代码

```
45  /*扫频 (1-40Hz) */  
46  float TIM_Fre = 500;  
47  uint8_t A = 3;           //扫频振幅  
48  uint32_t Period = 3 ;   //周期  
49  static uint32_t n = 0;  
50  uint32_t N;  
51  float CHECK ;
```

```

1139 float sweep_fre[60] = {1.000000, 1.500000, 2.000000, 2.500000, 3.000000, 3.500000,
1140     4.000000, 4.500000, 5.000000, 5.500000, 6.000000, 6.500000, 7.000000, 7.500000, 8.000000,
1141     8.500000, 9.000000, 9.500000, 10.000000, 10.500000, 11.000000, 11.500000, 12.000000, 12.500000,
1142     13.000000, 13.500000, 14.000000, 14.500000, 15.000000, 15.500000, 16.000000, 16.500000, 17.000000,
1143     17.500000, 18.000000, 18.500000, 19.000000, 19.500000, 20.000000, 20.500000, 21.000000, 21.500000,
1144     22.000000, 24.000000, 25.000000, 26.000000, 27.000000, 28.000000, 29.000000, 30.000000, 31.000000, 32.000000,
1145     33.000000, 34.000000, 35.000000, 36.000000, 37.000000, 38.000000, 39.000000, 40.000000};
1146 //matlab生成的频率
1147
1148 float Sweep(void)
1149 {
1150     float w,omg,speed_ref;
1151
1152     static u16 i_fre = 0;
1153     //TIM_FRE 为函数执行频率,等效于采样频率
1154     w = sweep_fre[i_fre]*6.28f/TIM_Fre;//模拟角频率
1155     CHECK(w);
1156     omg = sweep_fre[i_fre]*6.28f;//数字角频率
1157     N = (uint32_t)(2*3.14159/w);//周期点数 ,采样数量
1158
1159     if(n<(uint32_t)(Period*N))
1160     {
1161         speed_ref = A*sinf(w*n);//给定目标角速度
1162         n++;
1163     }
1164     if(n>=(uint32_t)(Period*N))
1165     {
1166         n = 0;
1167         i_fre++; //获取下一个频率
1168     }
1169
1170     return speed_ref;
1171 }

```

上面的数组不用自己一个一个的敲，可以用matlab生成。

根据经验和时域的响应情况，调节PID参数使得云台可以较为稳定的跟随目标角速度。然后记住将速度环的输入改为我们的扫频信号

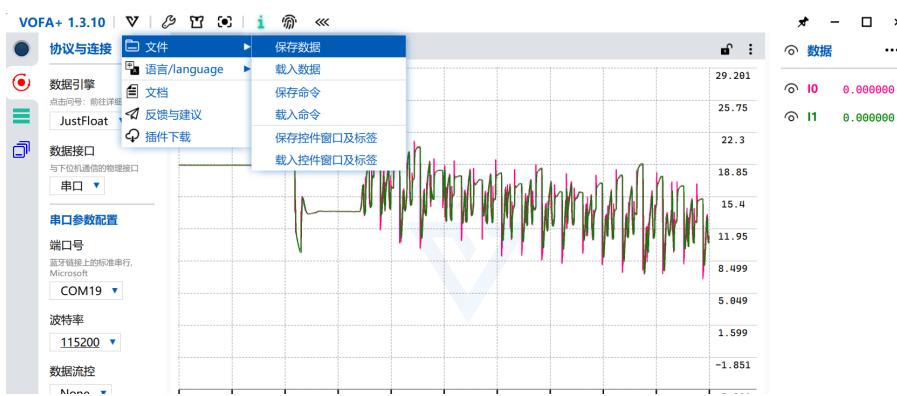
```

437     gimbal.pid.yaw_spd_ref =Sweep();
438     gimbal.pid.pit_spd_ref = pid_pit.out;
439     gimbal.pid.yaw_spd_fdb = gimbal.sensor.yaw_palstance;
440     gimbal.pid.pit_spd_fdb = gimbal.sensor.pit_palstance;

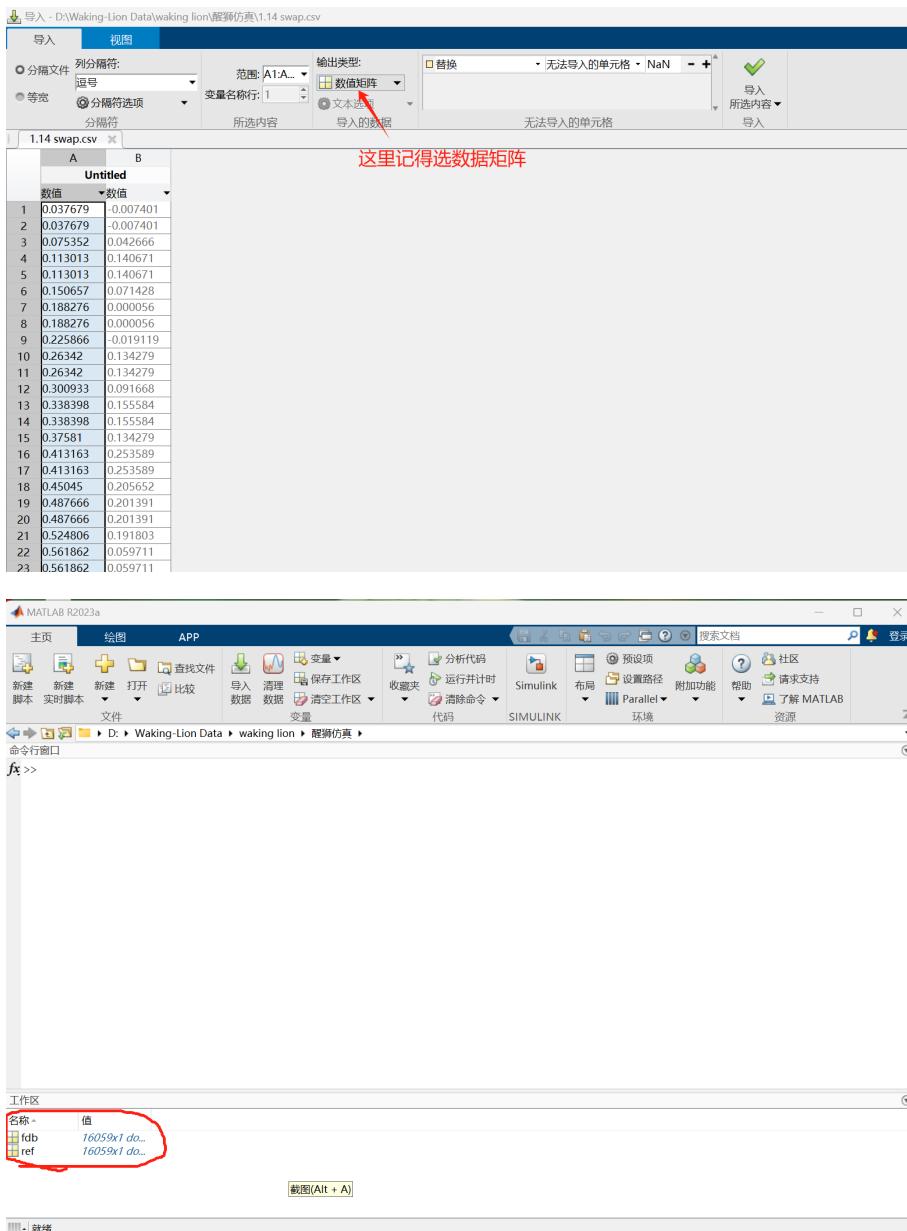
```

第②步:采集数据、导入数据

我们只需要用vofa+来采集数据就好了



在matlab上方的菜单中选择导入数据选项



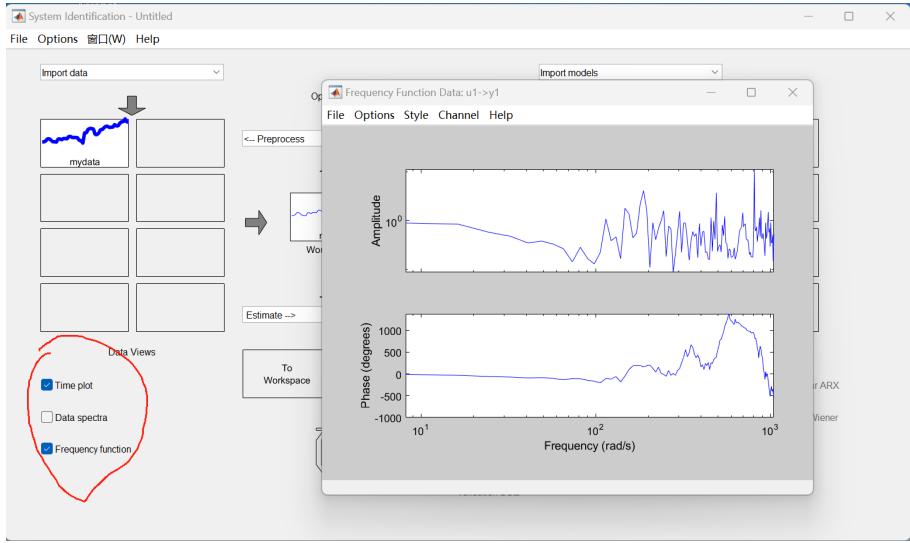
留意工作区的变量，记得自己修改名字。

将两组数据导入后我们就可以进行下一步了

第③步：系统识别

找到并打开system Identification工具箱，选择 Import data > Time domain data，打开导入数据对话框，在 input 和 output 选项中键入工作区的输

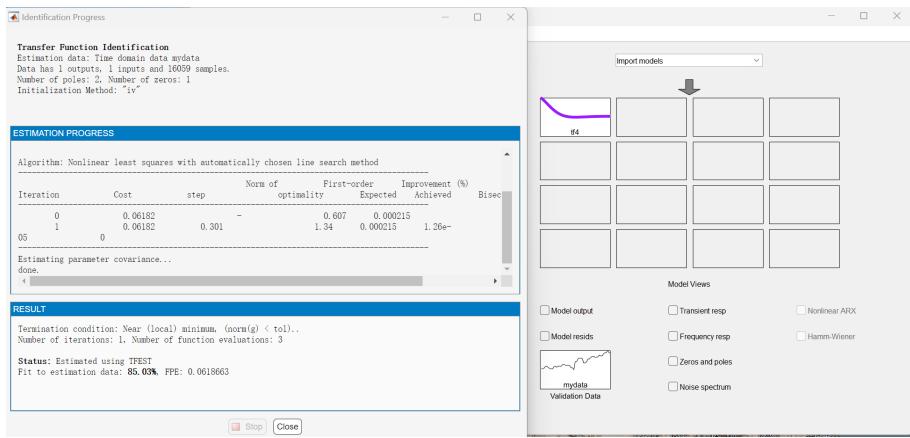
入输出变量名，Starting time 和 Sample time 分别输入 0 和 0.003（这个采样时间需要自己确定，往下看）。其他部分选填。点击 Import 将数据加入 System Identification 工具，然后关闭对话框。



注意勾

选上面的选项，来查看输入数据对应的bode图，假若图像如图属于正常状态(虽然很丑)，因为对于我们的云台系统而言，前面的低频时是基本符合规律的，而后面频率过高云台响应跟不上输入的信号，就可能会出现这种奇怪的图像。

直接选择 Estimate > Transfer Function Models 打开传递函数对话框，设置零极点个数分别0和1，点击 Estimate，得到系统传递函数。



至此，我们的系统就识别出来了

3.1.2 系统开环识别

RM官方开源教程-系统辨识基础教程

【Matlab】利用System identification App辨识对象模型。

云台参数识别与调试

3.1.3 知识补充

大家按照我的步骤做完之后肯定会有许多疑惑，首先为什么用一阶系统，为什么我们仿真要用速度环，而不直接仿真系统的传递函数呢。

先记住我们的6020电机用的是电压控制，然后出门左转看云台控制 – Yaw轴系统辨识与控制器设计，然后再看6020电压控制与电流控制的特别说明

采样时间的确定

打开bsp_dwt.c文件，在里面加下面的全局变量，并且在其头文件进行外部声明

```
21 uint32_t last_time;  
22 float deta_t;
```

将下面的函数放到vofa+的函数发送上，然后debug看deta_t这个变量，正常来说是一个定值0.002或0.003

```
150 aa[0] = gimbal.pid.yaw_spd_ref;  
151 aa[1] = gimbal.pid.yaw_spd_fdb;  
152 deta_t = DWT_GetDeltaT(&last_time);  
153 JustFloat_Send(aa, 2, USART1);
```

伯德图的作用

伯德图是我们分析系统频率响应的工具，对于一个系统来说输入不同频率的信号其对应输出信号的频率不会变化，但是幅度和相位的大小会发生改变，其改变就是通过伯德图反映出来的。

具体的推导请参考下面的链接：

频率响应的推导

Dr.Can——伯德图

看完推导后大家一定要记住结论：

当正弦输入 $u(t) = M_i \cdot \sin(\omega_i t + \phi)$ 通过线性时不变系统 $G(s)$ 后， ω_i 不变；振幅变化 $|G(j\omega)|$ 倍，相位移动 $\angle G(j\omega)$

在伯德图中我们能看到的东西：

(1) 系统增益

低频增益：反映了系统对直流信号或低频信号的放大能力。例如：

积分器在低频段增益趋向于无穷大 (-20dB/decade)。

高频增益：反映了系统对高频噪声的衰减能力。例如：低通滤波器在高频段增益趋向于0。

(2) 系统的转折频率

幅频特性曲线中，斜率发生变化的位置称为转折频率，通常对应系统中极点或零点的位置。通过转折频率可以辨识系统的动态特性。

(3) 系统的带宽

带宽是系统能够有效响应信号的频率范围（通常以幅值下降至-3dB处的频率为准）。带宽越大，系统响应速度越快。

(4) 系统的相位滞后

相频特性曲线显示了系统在不同频率下的相位滞后特性：

相位滞后越小，系统响应越快。

相位滞后过大可能导致系统稳定性变差。

想要深究的同学可以看下面的链接:

[伯德图的深入理解](#)

3.2 PID控制器的设计与调参

3.2.1 理论分析

当我们得到了一个系统之后我们就可以设计控制器进行调参了，PID控制器在理论和实践中都有丰富的经验积累，许多工程问题和解决方案已经得到了深入研究，让我们不需要对大量的控制器进行尝试和筛选，仅需要调参即可。

但是我们往年来都是进行的经验调参，对PID没有具体的概念和知识，这样会使我们的调参很漫长和痛苦，所以我们需要一些理论的知识来指导我们的实践，从而减少我们调参所需要的时间。

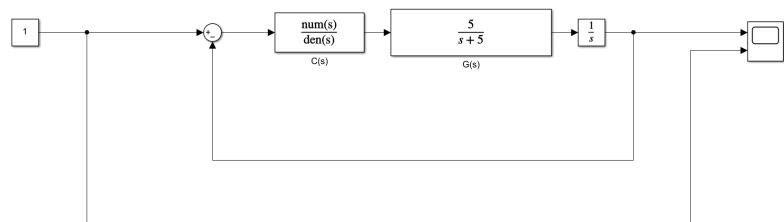
在经典控制理论中，对于控制器系统的分析我们会有多种的工具，如根轨迹、伯德图、奈奎斯特图等，因为伯德图更加直观和容易理解，所以这里从伯德图的角度，给大家来分析PID控制的本质及其原理。

首先我们需要有一个概念就是控制器实际上就是改变系统的零点和极点，从而修改系统的频率响应的幅值和相位（参考上面的频率响应推导结论），而伯德图正是用来观察系统的频率和相位特性，所以我们会利用伯德图来帮助我们理解PID的控制器。

现在我们有一个一阶系统:

$$G(s) = \frac{5}{s + 5}$$

我们将要对他进行控制器的设计:

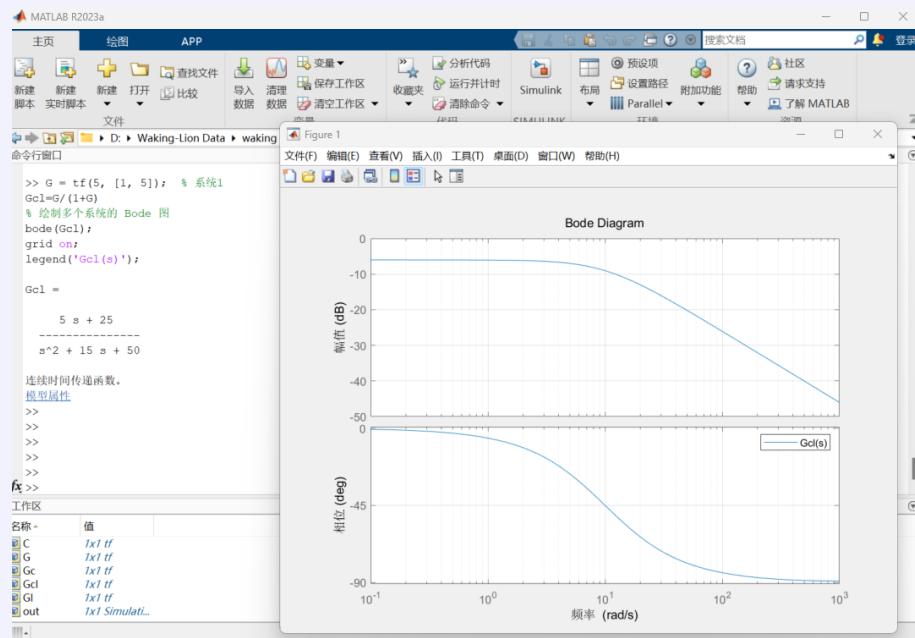


其闭环传递函数为:

$$G_{cl}(s) = \frac{C(s) \cdot G(s)}{1 + C(s)G(s)}$$

$$R(s) \longrightarrow \boxed{G_{cl}(s) == \frac{C(s)G(s)}{1 + C(s)G(s)}} \longrightarrow X(s)$$

当 $C(s) = 1$ 时, 即没有控制器的情况下, 其闭环函数的伯德图如下



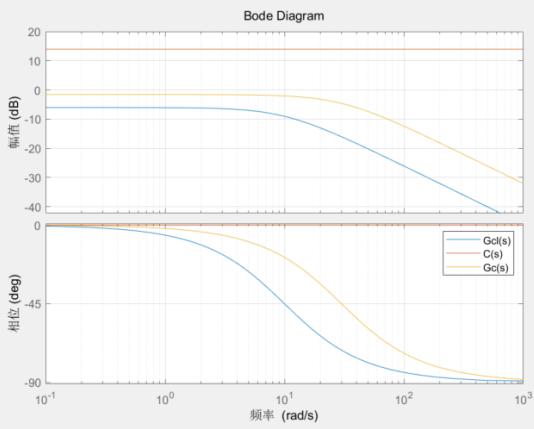
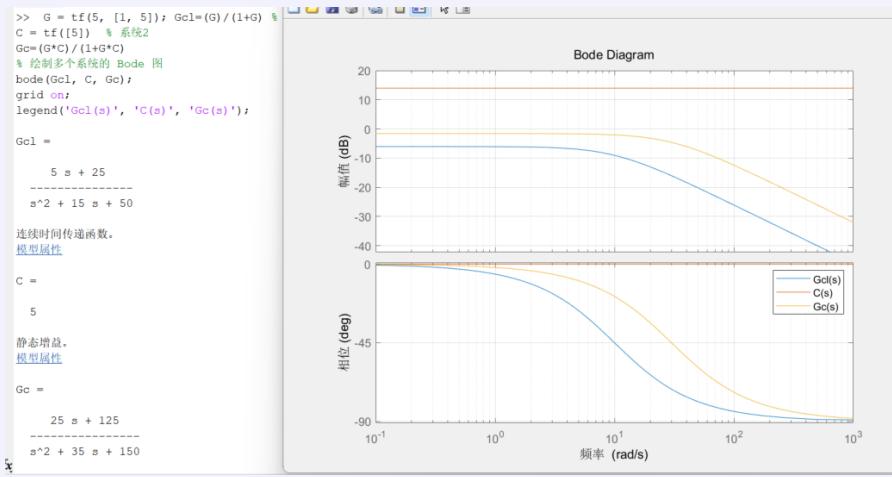
从伯德图可以看出，系统闭环函数存在一个稳态误差使得 $|G_{cl}(j\omega)| < 1$ 具体差多少可以将 $j\omega$ 代入 G_{cl} 中并且取 $\omega = 0$ ，得出与参考值相差的倍数。

而且我们还能看出这个系统是一个对高频噪声有良好的抑制效果的系统

我们PID的在s域的表达式为：

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

我们先只让 $K_p = 5$, $K_i = 0$ 、 $K_d = 0$, 可得到伯德图如下：

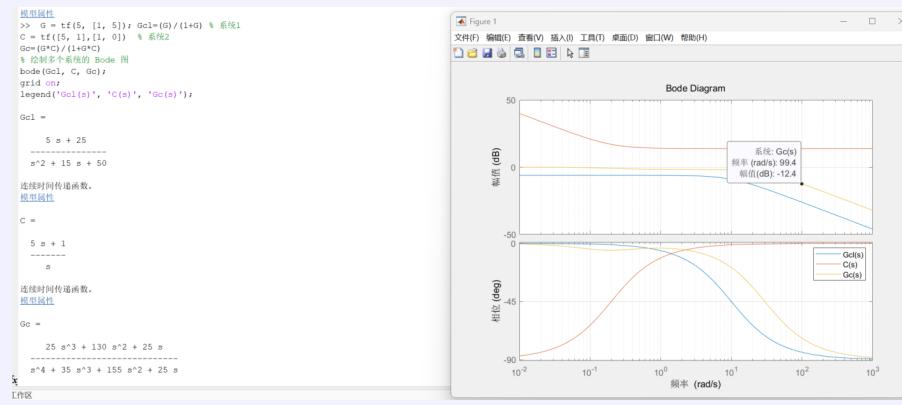


观察伯德图我们可以看到 $C(s) = 5$, 将系统的伯德图整体拉高了, 使得稳态误差减少了。

但是我们看到他的抗噪声能力下降了, 对比原系统的伯德图, 假如输入信号都有一个100Hz的噪声, 原系统可以抑制到-20多dB, 但是加了只有比例项的PID后只能抑制到-10dB。这个和 $C(s) = 5$ 的伯德图本身就有关系 (关于伯德图的叠加关系记得看上面dr.can的视频)。所以一旦有这个频段的噪音, 系统就会剧烈抖动。

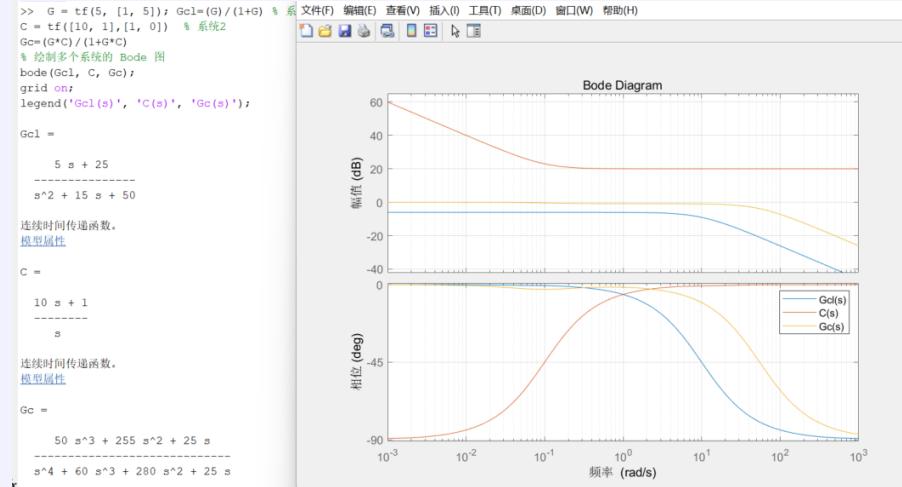
为什么不用考虑低频的噪音, 因为噪音普遍是高频的

所以在比例项的基础上我们会引入积分项, 让 $K_p = 5$, $K_i = 1$ 、 $K_d = 0$, 可得到伯德图如下



我们可以看到稳态误差消失了，但是随着频率的升高其幅度响应还是会下降，现在我们的目的就是尽量在不提高噪声对系统的影响下，去提高低频段对0dB的贴合

让 $K_p = 10$, $K_i = 1$ 、 $K_d = 0$, 伯德图如下，我们可以看到低频段更贴合了



基本上比例积分控制就可以将系统控制到我们想要的效果。大家可以到matlab去尝试如何改变PID的值才能达到我们的目的

3.2.2 代码实现

按照我们的思路我们需要把我们的输入信号（云台角度参考值）改成可以改变频率的正弦信号，然后我们可以从1rad/s开始，调PID的参数，使输出值能贴近参考值，然后增大频率继续调，大概到10多rad/s就可以了。如果再大的话抗噪声的能力就会下降，系统就容易高频振动，而且我们云台6020电机实际的最大角速度和角加速度也不能达到这么快的响应。

按照上面的分析随着我们输入的频率不断增大， K_p 值也要不断地增大，才能让输出跟上输入，当 K_p 增大到一定的值系统就容易受到噪声干扰而振动。

```
float sweep_freq;
float TIM_Fre = 500;
uint8_t A = 3;           //扫频振幅
uint32_t Period = 3;    //周期
static uint32_t n = 0;
float Sine_wave(void)
{
    float w,omg,angle_ref;

    //TIM_FRE 为函数执行频率,等效于采样频率
    w = sweep_freq*6.28f/TIM_Fre;// 数字角频率
    omg = sweep_freq*6.28f;// 模拟角频率
    N = (uint32_t)(2*3.14159/w);//周期点数 ,采样数量

    if(n<(uint32_t)(Period*N))
    {
        angle_ref = A*sinf(w*n); //给定目标角度
        n++;
    }
    if(n>=(uint32_t)(Period*N))
        n = 0;

    return angle_ref;
}
```

普通模式下我们用正弦信号就好了，但是自瞄模式我们需要用锯齿波来进行调参。为什么需要锯齿波呢，是因为在小陀螺的时候我们收到算法的值

是一个突变的值，就类似于锯齿波的性质。参考代码如下

```
float sweep_freq=1.0;
float TIM_Fre = 500;
uint8_t A = 3;           //扫频振幅
uint32_t Period = 3 ;    //周期
static uint32_t n = 0;
float Sawtooth_wave(void)
{
    float w,omg,angle_ref;

    //TIM_FRE 为函数执行频率,等效于采样频率
    w = sweep_freq*6.28f/TIM_Fre;// 数字角频率
    omg = sweep_freq*6.28f;// 模拟角频率
    N = (uint32_t) (2*3.14159/w); //周期点数 ,采样数量

    if(n<(uint32_t)(Period*N))
    {
        // 线性增长, 范围从 0 到 A
        angle_ref = A * (float)(n % N) / N;//给定目标角度
        if(angle_ref>A)angle_ref=0;
        n++;
    }
    if(n>=(uint32_t)(Period*N))
        n = 0;

    return angle_ref;
}
```

然后跟调正弦波一样调参就行了

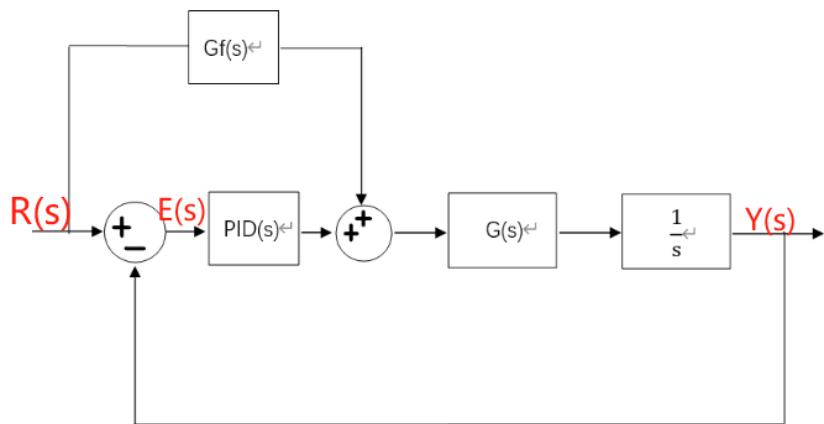
3.3 前馈控制器

3.3.1 理论知识

什么是前馈控制

前馈控制是一种控制策略，没有固定的表达形式。它可以提高系统的响应速度，减小误差，增加带宽但是不改变系统的稳定性。

从字面上来理解就是我们会给系统多输入一个值从而让我们更快的达到目标值。



其中 $G(s)$ 为速度环， $PID(s)$ 为外环PID对应的传递函数， $G_f(s)$ 是前馈传递函数。

前馈控制有三种基本表现的形式，在这里符合我们需求的是基于受控系统倒数的形式。

下面给大家公式推导来一下(先不用管那个积分器):

$$Y(s) = [R(s)G_f(s) + E(s)C(s)]G(s)$$

$$Y(s) = R(s)G_f(s)G(s) + E(s)C(s)G(s)$$

$$R(s) - E(s) = R(s)G_f(s)G(s) + E(s)C(s)G(s)$$

$$E(s) = \frac{1 - G_f(s)G(s)}{1 + C(s)G(s)}R(s)$$

另外还可以再推导出整个系统的开环函数：

$$\frac{Y(s)}{R(s)} = \frac{(G_f(s) - C(s))G(s)}{1 - C(s)G(s)}$$

根据我们的推导结果，我们可以看到如果 $G_f(s) = \frac{1}{G(s)}$ ，那么在理论上误差 $E(s)$ 就等于 0。同时在开环函数中我们可以看到 $G_f(s)$ 改变的是仅是零点，所以可以改变响应的快慢而不影响系统的稳定性。

前馈控制三种形式

3.3.2 代码实现

①速度环的前馈控制

就用我们上面仿真出来的速度闭环函数：

$$G(s) = \frac{32.36}{s + 33.96}$$

所以内环的传递函数为

$$G_1(s) * \frac{1}{s} = \frac{32.36}{s^2 + 33.96s}$$

然后就得到前馈控制的函数：

$$G_f(s) = \frac{1}{G_1(s)} = \frac{s^2 + 33.96s}{32.36}$$

然后根据脉冲不变法就可以得到差分方程了，转换为C语言方程。

```
float FFC_OUT(float x_n)
{
    static float ts = 0.003;
    static float x_n_1 = 0.0f;
    static float x_n_2 = 0.0f;
    static float a1 = 96.89f/89.38f;
    static float a2 = 1/89.38f;
    float x_dot_2 = 0.0f;
    float x_dot_1 = 0.0f;
    float y_n;

    x_dot_1 = (x_n - x_n_1)/ts;
    x_dot_2 = (x_n - 2*x_n_1 + x_n_2)/(ts*ts);

    y_n = a1*x_dot_1 + a2*x_dot_2;

    x_n_2 = x_n_1;
    x_n_1 = x_n;
    return y_n;
}
```

下面我给大家推导一下代码中的差分方程怎么来的，这里涉及一些离散信

号的知识暂时不懂也没关系，后面有可能就会用到。

$$\begin{aligned} G_f(s) &= \frac{s^2 + 33.96s}{32.36} \\ &= \frac{1}{32.36}s^2 + \frac{33.96}{32.36}s \\ g_f(t) &= \mathcal{L}^{-1}[G_f(s)] = \frac{1}{32.36}\delta''(t) + \frac{33.96}{32.36}\delta'(t) \end{aligned}$$

将信号离散化， $t = nT_s$, T_s 为采样时间

$$\begin{aligned} g_f(t) &= \frac{1}{32.36}\delta''(nT_s) + \frac{33.96}{32.36}\delta'(nT_s) \\ &= \frac{1}{32.36} \frac{[\delta'(nT_s) - \delta'((n-1)T_s)]}{T_s} + \frac{33.96}{32.36} \frac{\delta(nT_s) - \delta((n-1)T_s)}{T_s} \\ &= \frac{1}{32.36} \frac{[\delta(nT_s) - \delta((n-1)T_s) - \delta((n-1)T_s) - \delta((n-2)T_s)]}{T_s^2} \\ &\quad + \frac{33.96}{32.36} \frac{\delta(nT_s) - \delta((n-1)T_s)}{T_s} \\ &= \frac{1}{32.36} \frac{[\delta(nT_s) - 2\delta((n-1)T_s) - \delta((n-2)T_s)]}{T_s^2} + \frac{33.96}{32.36} \frac{\delta(nT_s) - \delta((n-1)T_s)}{T_s} \end{aligned}$$

最后转换成差分方程，设输入信号为 $x(n)$ ，输出信号为 $y(n)$

$$\frac{1}{32.36} \frac{[x(nT_s) - 2(x(n-1)T_s) - (x(n-2)T_s)]}{T_s^2} + \frac{33.96}{32.36} \frac{x((n)T_s) - x((n-1)T_s)}{T_s}$$

我们的代码里面 x_n 的参数已经是根据采样时间得到的，因为我们的采样时间是按照freertos执行的频率来算的，当我们的代码执行到这里的时候也就是0.003s了，这就是为什么代码里面不需要再对 x_n 进行处理

写完函数之后我们就可以直接用了，

```
//Yaw轴加前馈
gimbal.pid.yaw_spd_ref += FFC_OUT(gimbal.pid.yaw_angle_ref/57.3);
if(gimbal.pid.yaw_spd_ref > 30)
    gimbal.pid.yaw_spd_ref = 30;
if(gimbal.pid.yaw_spd_ref < -30)
    gimbal.pid.yaw_spd_ref = -30;

pid_calc(&pid_vision_yaw_spd, gimbal.pid.yaw_spd_fdb, gimbal.pid.yaw_spd_ref);
pid_calc(&pid_vision_pit_spd, gimbal.pid.pit_spd_fdb, gimbal.pid.pit_spd_ref);
```

在我们的代码里面我们的“angle_ref”的单位是度，而我们建模的单位是弧度(rad)，所以我们需要进行角度的转换才能代入函数，而且由于电机的物理特性限制我们最大的角速度是±30，所以我们需要进行一个限幅（这些都可以通过查手册、计算知道的）

②电流环的前馈控制

上面我们学习了速度环的前馈控制，接下来我们学习一下电流环的前馈控制。至于为什么我们学习了速度环的控制还要再学电流环的控制呢？虽然在普通模式下用速度前馈的是非常流畅且静态误差很少，但是在进入自瞄的情况下云台会出现非常严重的超调现象。这是因为算法发给我们的自瞄值是一个突变的值，而且还带有比较严重的噪声信号，所以导致云台在自瞄的时候会出现非常大的超调现象。

而电流前馈控制可以有效的解决这样状况，因为电流前馈是直接作用于电压指令（PWM调制前），补偿电气的 L/R 时间常数（通常微秒级）。电流环带宽可提升至1kHz 10kHz，适合高频扰动抑制。再加上电流前馈对于PID参数的敏感度更低

3.4 阻力扭矩补偿

3.4.1 理论知识

怎么理解阻力扭矩补偿

首先，我们需要先知道我们为什么需要阻力扭矩补偿？

考虑到物理现实的原因，我们云台转轴相关连接件之间是有摩擦阻力的这是无法避免的。所以，我们电机时间的云台转角和输入力矩之间的真实关系并不是如上面的模型所说的那样理想，其真实的物

理模型应当如下：

$$J\ddot{\theta} + b\dot{\theta} + N = T$$

其中， b 为粘滞摩擦系数（可以简单的理解为和我们上面说的阻尼系数是一样的）， N 为动摩擦或静摩擦力矩（即阻力扭矩）

这里我们可以看到，由于静摩擦或动摩擦力矩的出现，模型引入了非线性量 N ，通过我们前面的PID的学习我们可以知道，PID对非线性时不变系统的处理是比较差的。

为什么对非线性的处理会比较差呢？在这里我略加解释一下，有利于大家理解，以我们控制的电机为例：

对于比例项P来说，非线性会导致增益 K_p 的有效性变化

- 在低速时，静摩擦占主导，需要更大的 K_p 才能克服。
- 而在高速时，动摩擦+粘滞摩擦占主导， K_p 需求降低。

在控制系统调试过程中，由于参数整定的局限性， K_p 的可调范围受到约束。具体表现为：在低速工况下整定的 K_p 值，若直接应用于高速工况，可能导致系统失稳甚至发散（如出现显著的超调或振荡）。这一现象凸显了系统动态响应（如快速性）与稳定性（如超调抑制）之间的固有矛盾，本质上是由于系统在不同工作点下的动态特性（如带宽、相位裕度）随工况变化而导致的鲁棒性不足问题。

对于积分项I来说，非线性会导致积分饱和（Integral Windup）或静态误差波动。

- 齿槽效应导致扭矩周期性波动→积分器持续累积误差，引发振荡。
- 静摩擦导致“死区”，积分器需要更长时间补偿。

下面我会结合传递函数的知识给大家讲解一下，有助于大家理解，如果还看不懂的话那就记住上面的结论吧。

先将上面的微分方程进行拉普拉斯变换，得到：

$$\begin{aligned} Js^2\Theta(s) + bs\Theta(s) + \frac{N}{s} &= T(s) \\ \Rightarrow \Theta(s) &= \frac{T(s)}{Js^2 + bs} - \frac{N}{s(Js^2 + bs)} \end{aligned}$$

前面一项是我们理想的系统，后面一项是非线性项（扰动项）。而这个非线性项就是我们需要补偿的阻力扭矩。

这个阻力扭矩在我们小陀螺的时候会尤其明显，在和算法联调的时候可以发现我们离目标会有一个稳定的偏角，所以我们需要对阻力扭矩进行补偿，来消除非线性项对系统的影响，即消除动摩擦或者静摩擦的影响。至于为什么不用积分，我相信我上面已经说清楚了。

说白了其实这个阻力扭矩补偿也是一种前馈控制器，是对已知的量进行补偿，这个请参考我上面让大家看的前馈控制三种形式。

这里可参考中科大的阻力扭矩补偿的视频

3.4.2 代码实现

首先我们应该怎么获取这个阻力扭矩呢？我们可以通过实验的方法来获取该值，我们让云台的yaw轴缓慢匀速的转动，缓慢的目的是忽略粘滞摩擦力的影响，而匀速的目的是让云台的动力和阻力二力平衡从而间接获得阻力扭矩。

OK，理论成立，我们就要知道怎么去实现。

我们可以通过vofa+来采样匀速转动的电流值，为什么是电流值呢？因为电流值和力矩是成正比的，电流越大，力矩越大。至于怎么让他匀速转动，我相信能看懂这个文档的学弟学妹应该不需要我教了，这里我就不赘述了。

这里需要注意一下，对于6020来说有两种控制电机的方式：电流控制和电压控制。如果是电压控制的话，这个补偿的效果不会很好，因为电压和力矩不是线性关系。6020如果是电压控制的话就刷一下固件，如果是4310的达秒电机则无所谓，因为4310只有电流控制

以哨兵的云台为例，哨兵云台的yaw轴是4310的达秒电机，我们让他以转速为1.4rad/s进行旋转。然后，对其电机的电流进行采样。得到如下图所示的波形。（提醒一下采样前记得把底盘注释掉）

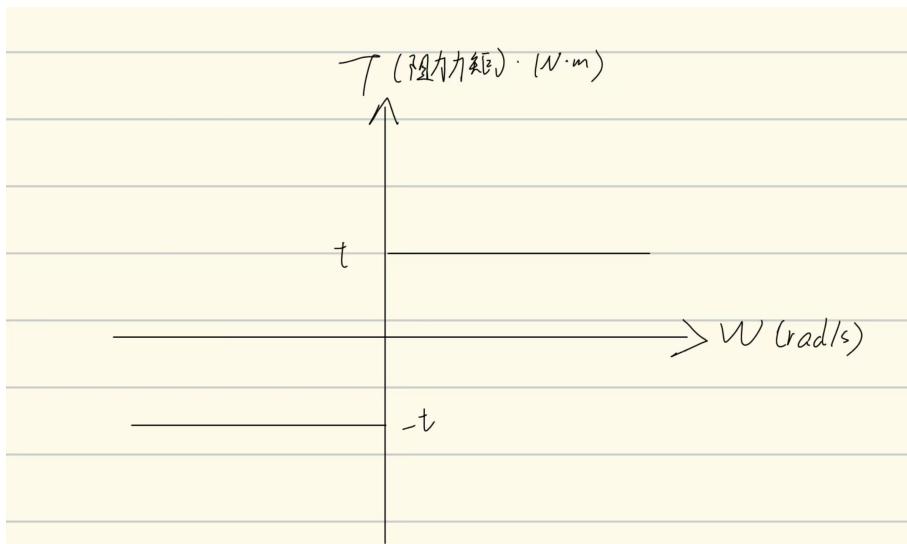


这里可以看到我们的电流曲线并不是一条完美的直线，首先就是因为我们电机的电流采样是有噪声的，其次就是因为我们的转速并不是一个完美的匀速转动，我们需要进过PID的闭环运算，所以无法达到绝对的匀速，所以我们得到的曲线并不是一条完美的直线。

我们只需要取它的平均值，这个平均值可以不用太准确，对于现在的波形图来说，他的平均值就是1050左右

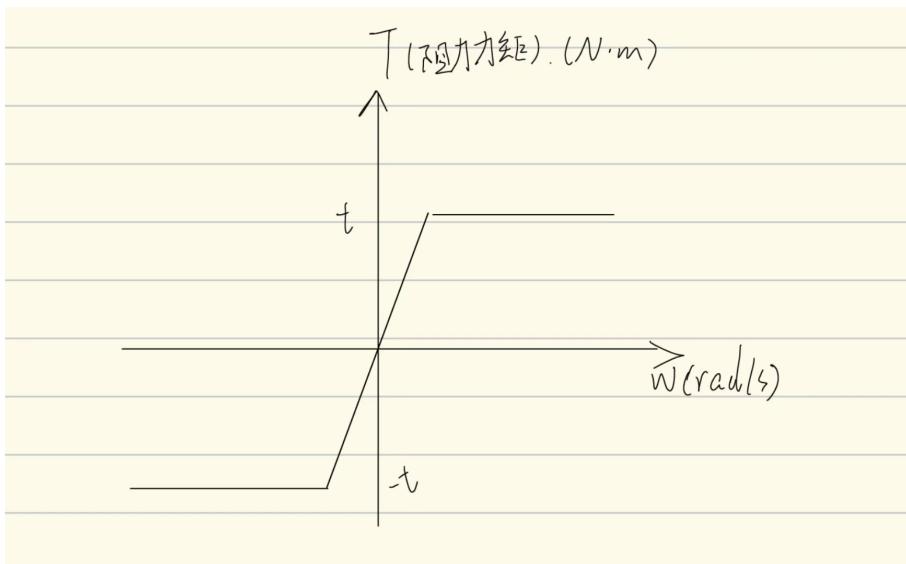
所以我们可以得到一个低配版的转速和阻力扭矩的关系（要时刻记住直流无刷电机电流和力矩是成正比的），这个函数图是一个非常理想的图，它忽略了静摩擦的影响，直接简化为一个阶跃函数，对于电压控制的电机可以

这样粗糙的给补偿：



但是在这个低配版的阻力扭矩补偿中，假如我们的值处于零点附近的话，那么就可能出现零点震荡的现象，这是因为倘若转速在零点附近波动时，我们会发现电流值会在1050和-1050之间来回波动，这个时候电机就会容易抽搐，所以我们需要对这个函数进行一个平滑处理，才能得到一个比较理想的函数图。

至于如何处理：我们可以将零点附近的一小段区间用正比例函数贴合过去，称之为“一阶连续化”。



按照上面的函数图我们就可以写出以下的代码，实现这个函数的功能。（注意这是电流补偿，所以是直接加到你要发送的电流上）

```
//Yaw_Compensation(gimbal.pid.yaw_spd_fdb,pid_yaw_spd.out)
float Yaw_Compensation(float w_ref, float w)
{
    float uq_positive = 1050.0f;
    float uq_negative = -1050.0f;
    float uq_comp = 0;
    float width = 0.2f;
    float b = (uq_positive + uq_negative)/2;
    float k = (uq_positive - b)/width;
    if(w >= 0)
    {
        if(w < width)
            uq_comp = k*w + b;
        else
            uq_comp = uq_positive;
    }
    else
    {
        if(w > width)
            uq_comp = k*w + b;
        else
            uq_comp = -uq_negative;
    }
    if(w_ref >= 0)
        return uq_comp;
    else if(w_ref< 0)
        return -uq_comp;
    else
        return 0;//防报错
}
```

3.5 参考输入微分的前馈补偿

3.5.1 理论知识

什么是参考输入微分的前馈补偿

大家有可能会在代码中看到这个函数float getFeedforwardControl，也有可能已经被删了，但是不管怎么样我这里也顺便说一下。但是我们一般不使用这个前馈方式，所以我也就稍微讲一下理论知识，也把前馈的最后一种形式说一下，让文档更完整一点。

参考输入微分的前馈补偿是指在控制系统中，直接对参考输入信号进行微分处理，以便更好地预测和补偿系统的动态响应。其主要目的是通过提前计算出系统对参考输入的响应，从而提高系统的跟踪性能和响应速度。

我们从最开始的微分方程开始推导：

$$J\ddot{\theta} + b\dot{\theta} = T$$

将其差分离散，让我们可以用代码来实现：

$$\begin{aligned} J \left(\frac{\theta(n+1) - 2\theta(n) + \theta(n-1)}{T_s^2} \right) + b \left(\frac{\theta(n+1) - \theta(n)}{T_s} \right) &= T(n) \\ \Rightarrow \frac{J}{T_s^2} [\theta(n+1) - 2\theta(n) + \theta(n-1)] + \frac{b}{T_s} [\theta(n+1) - \theta(n)] &= T(n) \end{aligned}$$

其实上面可以理解为，在知道精确的系统的微分方程时，用几个输入的离散点来对输出进行一个短时预测。

假如我们需要将在位置环中加入前馈，就需要用第一项来进行补偿，如果在速度环中加入前馈则需要用到第二项进行补偿

但是这个控制器的实现我们需要考虑两个问题：

①我们的转动惯量J和阻尼系数b并没有办法知道，即使云台的转动惯量可以问机械拿到（其实SolidWorks也并不准确，但是只是作为一个前馈控制器的参数也勉强可以用），但是阻尼系数b是无法知道的。这时可能会有读者问为什么不可以用上面的仿真出来的系统函数来反推转动惯量和阻尼系数呢？这个就关乎于我们的第二个问题了。

②我们曾经说过，我们可以简单的将的电机输出力矩可以看做与输出电流成正比的关系，即 $T = K_t I$ ，其中 K_t 为电机的力矩常数。我们的输入和输出都是电流值，所以说我们仿真出来的系统函数与实际的系统函数其实是还差一个 K_t 值即：

$$\begin{aligned} JsW(s) + bW(s) &= K_t I(s) \\ \Rightarrow G(s) &= \frac{K_t}{Js + b} \end{aligned}$$

这个 K_t 值我们暂时没有办法得出来，即使得出来了最后的效果也不一定好，因为我们只是理想的认为力矩电流和力矩是成正比的，其实其中还有非线性因素存在。

以上的问题基于我现在水平和能力暂时没有办法解决，由于无法实操其理论知识依然停留在表层，有兴趣的学弟学妹可以尝试去解决一下，或者有更好的方法也可以告诉我，我会在文档中进行更新。

讲到这里我小小的提一嘴，其实前馈的三种形式我们已经全部学完了，并且能很好的融入到我们的代码之中，还有不理解的可以再次参考一下我上面推荐的前馈控制的三种形式的视频再回头看看文档，希望大家有更深刻的理解。

3.6 重力补偿

云台参数识别与调试

3.7 濾波器

3.7.1 引言

在我们控制器的设计中，我们会遇到一个问题，就是我们的输入信号是有噪声的，这个噪声会导致我们的控制器输出不稳定，甚至会导致系统失稳。所以我们需要对输入信号进行滤波处理，滤波器可以有效的滤除噪声信号，从而提高系统的稳定性。

而在这里我会跟大家讲一下滤波器的基础知识，教大家如何利用MATLAB生成我们所需要的滤波器。至于说如何设计滤波器，这个我不讲了（其实目前我也没学），因为滤波器的设计是一个非常复杂的过程，需要根据具体的系统来进行设计，这里我只给大家介绍一下滤波器的种类和作用，以及如何用MATLAB生成我们想要的滤波器。想要知道是怎么设计的同学可以去学习《数字信号处理》这本书。

3.7.2 理论知识

滤波器的作用是滤除噪声信号，滤波器的种类有很多种，常见的有低通滤波器、高通滤波器、带通滤波器、陷波滤波器等。

这里我们只需要用到低通滤波器就可以了，低通滤波器可以有效的滤除高频噪声信号。

我们先从最简单的一阶低通滤波器开始学习，其传递函数为：

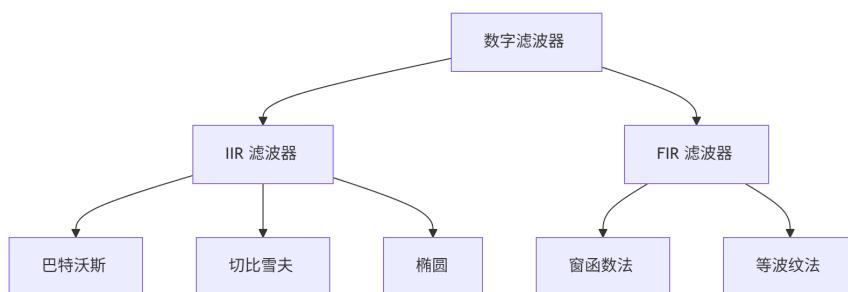
$$G(s) = \frac{\omega_c}{s + \omega_c}$$

其中 ω_c 为截止频率，截止频率越大，滤波效果越差。（这里大家就自己去用MATLAB生成一下伯德图研究一下不一样的截止频率所对应的图像吧，我就不过多描述了）

然而在很多情况下，一阶滤波器并不能满足我们的滤波要求，比如一阶滤波器衰减不够快、衰减幅度不够等，所以我们需要更高阶的滤波器。

我们可以用MATLAB来生成一个低通滤波器。跟着下面的教程你就可以生成你想要的滤波器。

我们先说说要怎么选择的滤波器，这里介绍一下几种滤波器的区别和适用情况。

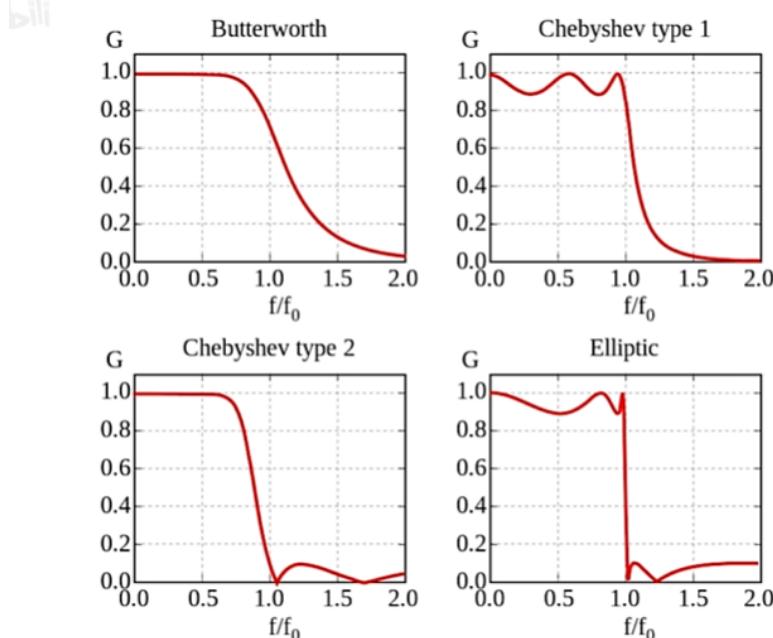


巴特沃斯滤波器：具有平坦的幅频响应，适用于对幅度响应要求较高的场合，但相位响应较差。

切比雪夫I型滤波器：具有较陡的幅频响应，但会引入波纹，适用于对幅度响应要求较高且可以接受波纹的场合。

切比雪夫II型滤波器：具有较陡的幅频响应，但相位响应较差，适用于对幅度响应要求较高且可以接受相位失真的场合。

同样的过渡带陡峭度从大到小依次为：切比雪夫I型、切比雪夫II型、巴特沃斯。也就是说如果需要相同过渡带陡峭度的话，巴特沃斯滤波器就会比切比雪夫滤波器的阶数更高，所需要的计算资源也会爆炸性增长



剩下的各种滤波器大家都可以在网上查到，这里就不一一介绍了。道理都是互通的

RM入门教学系列数字滤波器设计与实现

通过MATLAB得出来了一堆系数，那这堆系数要怎么用呢？在工程上我们对滤波器的计算方法会有两种：一种是直接型，一种是级联型。

直接型：直接将传递函数进行差分计算

级联型：将传递函数化为多个分子分母都为一阶或二阶的传递函数累乘，然后再差分计算

而那堆系数就是我们级联型所需要的系数，在传递函数中其实级联型和直接型是没有区别的，而误差累积的根源是从无限精度的传递函数到有限精度实现的映射过程。

如果是直接型，每个时间步的误差会被反馈系数 反复放大并重新注入系统，形成闭环，误差会被反馈系数反复放大（指数爆炸）。而级联型的每个节（Biquad）的误差仅影响下一节的输入，但不会通过反馈环路重新注入自身。第m节的误差 Δy_m 会传递到第m+1节，但不会反向影响第m节自身

两者的实现代码如下：

直接型：

```
0 const float Cherby_num[IIR_ORDER_Cherby+1] = {
1   0.0002160326112, -0.0005290300469, 0.0008167113992, -0.0008575262036, 0.0008167113992,
2   -0.0005290300469, 0.0002160326112
3 };
4 const float Cherby_den[IIR_ORDER_Cherby+1] = {
5   1,      -5.05384922,    10.70690346,   -12.16372299,    7.811606407,
6   -2.687725067,   0.3869373798
7 };
8 float IIR_Cherby(float data, float* x_input, float* y_output)
9 {
0   u8 i = 0;
1   float output = 0;
2   for(i=IIR_ORDER_Cherby;i>0;i--)
3   {
4     x_input[i] = x_input[i-1];
5     y_output[i] = y_output[i-1];
6   }
7   x_input[0] = data;
8   y_output[0] = Cherby_num[0]*x_input[0];
9   for(i=1;i<IIR_ORDER_Cherby+1;i++)
0   {
1     y_output[0] += Cherby_num[i]*x_input[i] - Cherby_den[i]*y_output[i];
2   }
3   output = y_output[0];
4   return output;
5 }
```

级联型：

```

/*************************************
MATLAB生成的chebyshevII 9-15Hz
*************************************/

ChebyshevFilter FilterII;
const int NL[MWSPT_NSEC] = {1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1}; // 分子分段阶数
const int DL[MWSPT_NSEC] = {1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1}; // 分母分段阶数
const float NUM_CF[MWSPT_NSEC][3] =
{
    {0.4159689546F, 0.0F, 0.0F}, // Section 1 (1st-order)
    {1.0F, -1.918473959F, 1.0F}, // Section 2 (2nd-order)
    {0.3523512483F, 0.0F, 0.0F}, // Section 3 (1st-order)
    {1.0F, -1.900286555F, 1.0F}, // Section 4 (2nd-order)
    {0.2476982921F, 0.0F, 0.0F}, // Section 5 (1st-order)
    {1.0F, -1.843962789F, 1.0F}, // Section 6 (2nd-order)
    {0.1212379187F, 0.0F, 0.0F}, // Section 7 (1st-order)
    {1.0F, -1.641412258F, 1.0F}, // Section 8 (2nd-order)
    {0.0260618832F, 0.0F, 0.0F}, // Section 9 (1st-order)
    {1.0F, -0.186531499F, 1.0F}, // Section 10 (2nd-order)
    {1.0F, 0.0F, 0.0F} // Section 11 (1st-order)
};

const float DEN_CF[MWSPT_NSEC][3] =
{
    {1.0F, 0.0F, 0.0F}, // Section 1 (1st-order)
    {1.0F, -1.922878742F, 0.9567910433F}, // Section 2 (2nd-order)
    {1.0F, 0.0F, 0.0F}, // Section 3 (1st-order)
    {1.0F, -1.834950328F, 0.8700845838F}, // Section 4 (2nd-order)
    {1.0F, 0.0F, 0.0F}, // Section 5 (1st-order)
    {1.0F, -1.738752127F, 0.7774022818F}, // Section 6 (2nd-order)
    {1.0F, 0.0F, 0.0F}, // Section 7 (1st-order)
    {1.0F, -1.641025662F, 0.6845001578F}, // Section 8 (2nd-order)
    {1.0F, 0.0F, 0.0F}, // Section 9 (1st-order)
    {1.0F, -1.572530627F, 0.6197930574F}, // Section 10 (2nd-order)
    {1.0F, 0.0F, 0.0F} // Section 11 (1st-order)
};

```

```

float ChebyshevFilter_Update(ChebyshevFilter *f, float input)
{
    float x = input;
    for (int section = 0; section < MWSPT_NSEC; section++) {
        if (DL[section] == 1)
        {
            // 一阶部分(纯增益)
            x = NUM_CF[section][0] * x;
        }
        else if (DL[section] == 3)
        {
            // 标准二阶部分(双二阶结构)
            float w0 = x - DEN_CF[section][1] * f->w[section][0]
                      - DEN_CF[section][2] * f->w[section][1];
            x = NUM_CF[section][0] * w0
                + NUM_CF[section][1] * f->w[section][0]
                + NUM_CF[section][2] * f->w[section][1];
            // 更新状态变量
            f->w[section][1] = f->w[section][0];
            f->w[section][0] = w0;
        }
        else if (DL[section] == 2)
        {
            // 特殊分段(一阶零点+一阶极点)
            float w0 = x - DEN_CF[section][1] * f->w[section][0];
            x = NUM_CF[section][0] * w0 + NUM_CF[section][1] * f->w[section][0];
            f->w[section][0] = w0;
        }
    }
    f->output = x;
    return x;
}

```

我对级联型的二阶部分进行数学推导，其余的一阶和直接型的差分方程都是如下的推导，大家可以自行推导。

12:33 路 X 频响-bode-N... X | Rotation + Q ABC

SOS 级联，二阶的推导。

$$H(s) = \frac{b_0 + b_1 s + b_2 s^2}{a_0 + a_1 s + a_2 s^2}, \text{ 双线变换法: } S = \frac{2}{T} \frac{1-z^{-1}}{1+z^{-1}}$$

$$H(z) = \frac{b_0 + b_1 \frac{1-z^{-1}}{1+z^{-1}} + b_2 \frac{1}{(1+z^{-1})^2}}{a_0 + a_1 \frac{2}{T} \frac{1-z^{-1}}{1+z^{-1}} + a_2 \frac{1}{T} \frac{(1-z^{-1})^2}{(1+z^{-1})^2}}$$

$$H(z) = \frac{b_0 T^2 (1+z^{-1})^2 + 2b_1 T (-z^{-1}x + z^{-1}) + (b_2 - 1) (1-z^{-1})^2}{a_0 T^2 (1+z^{-1})^2 + 2b_1 T (-z^{-1}x + z^{-1}) + 4b_2 (1-z^{-1})^2}$$

$$= \frac{(b_0 T^2 + 4b_2 T + 2b_1 T) + (2b_0 T^2 - 8b_1) z^{-1} + (b_0 T^2 - 2b_1 T + 4b_2) z^{-2}}{(a_0 T^2 + 4a_2 T + 2a_1 T) + (2a_0 T^2 - 8a_1) z^{-1} + (a_0 T^2 - 2a_1 T + 4a_2) z^{-2}}$$

化简

$$H(z) = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{A_0 + A_1 z^{-1} + A_2 z^{-2}} \stackrel{A_2 = 1}{\Rightarrow} \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 + A_1 z^{-1} + A_2 z^{-2}} \quad (1)$$

$$Y(z)(1 + A_2 z^{-1} + A_2 z^{-2}) = X(z)(B_0 + B_1 z^{-1} + B_2 z^{-2})$$

$$Y(z) + A_2 Y(z) + A_2 z^{-1} Y(z) = B_0 X(z) + B_1 z^{-1} X(z) + B_2 z^{-2} X(z)$$

$$Y[n] + A_2 Y[n-1] + A_2 Y[n-2] = [B_0 X[n] + B_1 X[n-1] + B_2 X[n-2]] \quad (2)$$

引刀 $W[n]$ 辅助函数：

$$W[n] + A_1 W[n-1] + A_2 W[n-2] = X[n]$$

整理得， $W[n] = X[n] - A_1 W[n-1] - A_2 W[n-2] \quad (3)$

$\left\{ \begin{array}{l} B_0 X[n] \rightarrow LTI \rightarrow W[n] \\ B_1 X[n-1] \rightarrow \oplus \rightarrow W[n-1] \\ B_2 X[n-2] \rightarrow \oplus \rightarrow W[n-2] \end{array} \right.$

$B_0 X[n] + B_1 X[n-1] + B_2 X[n-2] \rightarrow y[n]$

$X[n] \sum_{n=0}^{n-1} w[n] \rightarrow D \rightarrow w[n-1] \rightarrow D \rightarrow w[n-2] \rightarrow B_2 \rightarrow \sum_{n=0}^{n-2} y[n]$

$\therefore y[n] = B_0 W[n] + B_1 W[n-1] + B_2 W[n-2]$

双线变换： $Z^{-1} \{ Y(z) Z^{-k} \} = y[n-k]$

怕大家不理解中间的辅助函数是什么，大家可以去看看这个视频，中间的双线变换（其实还有前向和后向欧拉变换等方法）大家可以自行了解，只

是域的转换而已微分方程模拟框图

3.8 跟踪微分器

为什么我会讲解这个控制器呢？因为在我之前使用键鼠调车的时候发现我鼠标输入的值变成了阶跃信号，而非一条平滑的曲线，这导致了我的云台一卡一卡的。后面有师兄帮我加了这个控制器之后曲线就变得平滑了。

为了防止大家后面也遇到这种情况，这里给大家讲解它的用法，除了平滑曲线之外它还可以抑制超调现象，在一定情况下可以使用。

3.8.1 理论知识

什么是跟踪微分器

跟踪微分器主要用于从含噪声的输入信号中提取跟踪信号及其无噪声的微分信号，同时安排过渡过程避免系统超调。其核心设计思想是“通过动态过程生成平滑的过渡轨迹，而非直接使用跳变的输入信号。

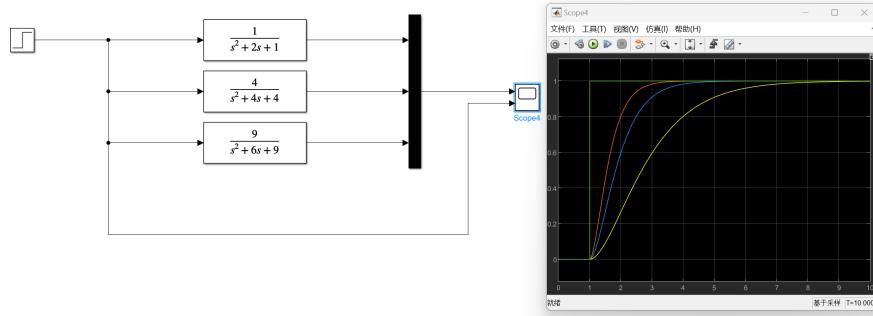
跟踪微分器的传递函数为：

$$G(s) = \frac{r^2}{s^2 + 2rs + r^2}$$

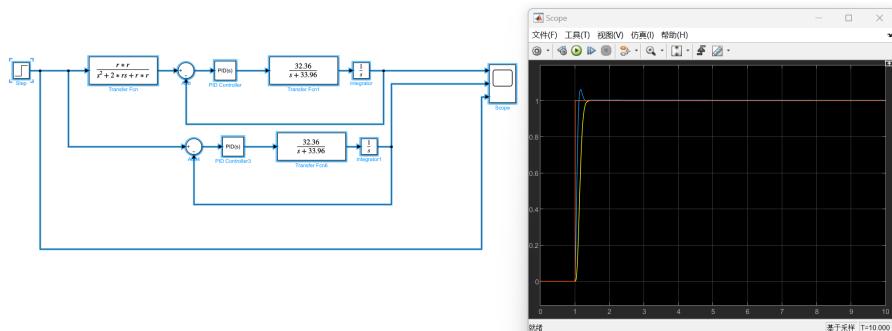
观察发现，其实它是由我们上面推出来过的二阶系统通用系统响应 $G(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}$ ，然后令 $\xi = 1$ 得到的
这是一个临界阻尼系统，我们为什么要用它来作为我们的跟踪微分器的系统呢？

这是因为跟踪微分器需要满足生成平滑过渡轨迹并提取微分信号的目标，需要无超调，且快速反应。如果是欠阻尼($0 < \xi < 1$)则会存在超调、震荡，

如果是过阻尼($\xi > 1$)的话，过渡时间随 ξ 增大显著延长，所以说临界阻尼系统是阶跃响应不产生超调的前提下衰减速度最快的系统。



通过上面的图我们看出，跟踪微分器将一个阶跃信号转化为一条平滑的曲线，并且随着我们的 r 变大，信号跟踪的速度也会越快。这样我们的目的就达到了，将突变的信号转换为平滑的曲线



从传递函数的角度来看，结合PID控制器我们可以得到如上的控制框图（图中 $r=20$ ，PID系数相同），我们的输入信号被跟踪微分器变平滑之后超调的部分就消失了。

还有一个问题就是我们为什么要让曲线变得平滑？这里我引用别人的解释

我们首先从物理上直观地去认识。不考虑偏载力矩、摩擦力矩等非线性因素，现假设某云台的最大加速的加速度和最大减速的加速度相同，记作 a_{max} ，现在想要其以最

快速度不超调的向某个方向转动 x 度，我们应该怎么做？

根据高中物理，我们知道，应该先让云台以 a_{max} 的加速度加速到 $\frac{x}{2}$ 处，然后再让云台以 a_{max} 的加速度减速运动到 x 。这就是一个简单的三角规划，它蕴涵了时间最优控制的思想。

那我们如果直接使用 PID，令这个云台去响应一个阶跃信号会发生什么呢？阶跃信号是一个物理系统无法实现的信号，当执行器性能远超于负载时（即相较于负载惯量和偏载力矩而言，电机输出的电磁转矩非常大，且额定转速满足系统快速性的要求），我们会发现，系统响应阶跃信号也可以响应的很好。

但大部分物理系统的执行器性能都是和实际需求相匹配的，在这种情况下，如果我们想要快速性，令云台刚开始以 a_{max} 的加速度加速，不管怎么调 PID 的参数，其在 $\frac{x}{2}$ 至 x 的区间内，一定无法一直以 a_{max} 的加速度减速，那么就会产生超调。如果我们不想要超调，那刚开始就一定只能用比较小的加速度来加速。这就是传统 PID “快速性” 和 “超调” 之间的矛盾。

所以我们需要对输入信号进行处理，使之变得物理可实现

最后一段话可以这么理解：如果一开始就以 a_{max} 的加速度加速，在 $\frac{x}{2}$ 的时候就无法直接以 a_{max} 减速，因为加速度从 a_{max} 到 $-a_{max}$ 是需要时间的。那么在加速阶段，云台以 a_{max} 加速，积累了较大的动能。在减速阶段，如果无法一直以 a_{max} 减速，意味着系统无法在相同的时间内消耗完这些动能，动能转化为运动，导致云台继续向前运动，直到动能被完全消耗，这就是超调的表现。

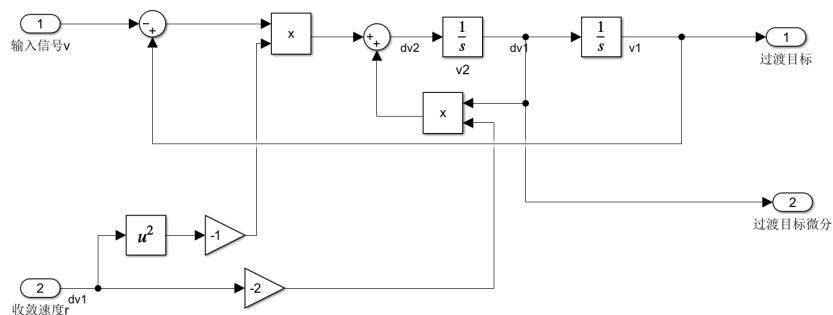
看到这的话，你对跟踪微分器有了一个表层的理解了，下面我会从现代控制的角度去讲解跟踪微分器，虽然现控不在本文档的范围内，但为了大家可以更深刻的理解，我会从这个角度给大家解释，大家只需要懂个大概就行。

由传递函数可以得到其状态空间方程：

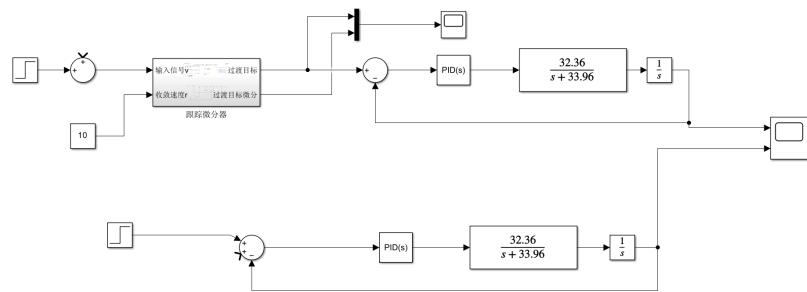
$$\begin{cases} \dot{v}_1 = v_2 \\ \dot{v}_2 = -2rv_2 - r^2(v_1 - v_0) \end{cases}$$

v_0 —目标信号； v_1 —过渡目标； v_2 —过渡目标微分； r —决定跟踪快慢的跟踪因子

由状态空间方程可以得到微分控制器的框图，可以简单的等效为上面所说的跟踪微分器的系统传递函数



为了让读者对控制流程有清晰的认识，下面是整个控制的框图



跟踪微分的核心机制：

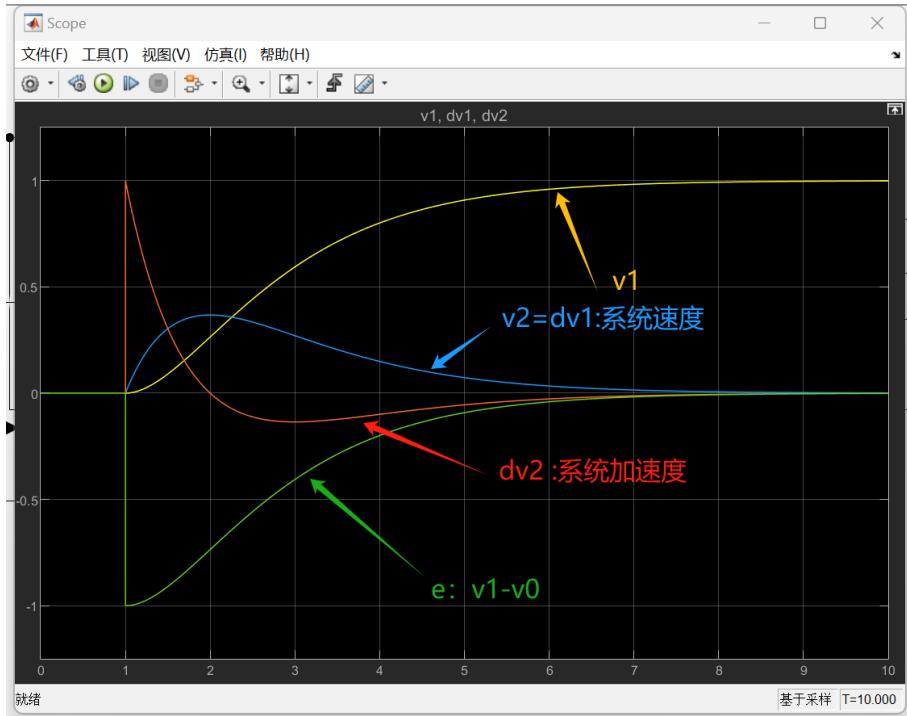
(1) 误差反馈

误差项 $e = v_1 - v_0$, 直接驱动系统的状态变量 v_2

把 $e = v_1 - v_0$ 代入状态空间方程：

$$\dot{v}_2 = -2rv_2 - r^2e$$

- 当 $e > 0$ 时, $-r^2e$ 为负, 驱动系统减速, 使 v_1 向 v_0 靠近。
- 当 $e < 0$ 时, $-r^2e$ 为正, 驱动系统加速, 使 v_1 向 v_0 靠近。



通过上图的例子简单地解释一下，当我们的输入为 1 rad 时，在 0 时刻时，误差最大所以角加速度也是最大，随着速度来到了最大值，随着加速度应该反向了，并且让速度减少。这样子我们的输入的阶跃信号就变成了 v_1 这个平滑的信号

从图中我们可以看到，即使误差 e 一直是负数，系统加速度 v_2 仍然可能由正变负，这是因为 $-k_2 v_2$ 的作用逐渐增强，抑制系统的加速过程， $-k_1 e$ 主导系统的加速或减速趋势，而阻尼项调节加速度的变化速度，二者共同决定了系统的动态行为

(2) 速度调节

微分信号 v_2 反映跟踪信号的变化速度。通过 $-2rv_2$ 项，系统能够调节过渡过程的速度，避免超调或振荡。

这也是我们不用一阶系统的原因，虽然一阶系统一定程度上也可以平滑信

号。但是他不存在我们上面所说的对系统靠近目标值的调节。同时我们的跟踪微分器还有一定的滤波作用，这个就让大家matlab自己动手仿真一下了，用matlab看看不同r值的伯德图的样子。

线性跟踪微分器

3.8.2 代码实现

上面讲了这么多，其实跟踪微分器的代码实现并不难，并且我们战队的代码中已经有移植好跟踪微分器了，我们只需要知道怎么去使用就好了

```
/*
 * 函数名: void ADRC_TD(LADRC_NUM *LADRC_TYPE1, float Expect)
 * 函数说明: LADRC跟踪微分部分
 * @param[in] 入口参数，期望值Expect(v0)输出值v1,v2
 * @par 修改日志
 * WangShun于2022-05-28创建
 */
void LADRC_TD(LADRC_NUM *LADRC_TYPE1, float Expect)
{
    float fh = -LADRC_TYPE1->r * LADRC_TYPE1->r * (LADRC_TYPE1->v1 - Expect) - 2 * LADRC_TYPE1->r * LADRC_TYPE1->v2;
    LADRC_TYPE1->v1 += LADRC_TYPE1->v2 * LADRC_TYPE1->h;
    LADRC_TYPE1->v2 += fh * LADRC_TYPE1->h;
}
```

在ladrc.c中可以找到上面的代码，这和我们上面的状态空间方程是一样的，至于这里为什么要乘一个积分步长，其实这就是采样时间，和我上面写前馈控制器将传递函数化成差分方程是差不多的，但是留意一下这里是将状态空间方程化成差分方程。大致的过程都是差不多的，我这里就不推导了。

```

LADRC_NUM LADRC_PIT =
{
    .r = 30,           //速度因子
    .h = 0.002,        //积分步长
};
LADRC_NUM LADRC_YAW =
{
    .r = 30,           //速度因子
    .h = 0.002,        //积分步长
};

/*控制pit,yaw速度*/
float yaw_spd=0.01;
static void gimbal_speed_ctrl(int16_t pit_ref_spd, int16_t yaw_ref_spd)
{
    //鼠标往左yaw_ref_spd为正
    // km.pit_v = pit_ref_spd * 0.015;
    //km.pit_v = -pit_ref_spd * 0.015;

    LADRC_TD(&LADRC_PIT,-pit_ref_spd * 0.007);
    LADRC_TD(&LADRC_YAW,-yaw_ref_spd * 0.007);

    km.pit_v = LADRC_PIT.v1;
    km.yaw_v = LADRC_YAW.v1;
}

```

我们只需要像上面一样定义好结构体的数值，其他的数值可以先不用管，比如观测器输出和观测器带宽等，这需要更多的现控知识，一时之间很难在这份文档里面说完。如果有需要的同学可以自己去研究。

关于r的调参大家只需要根据实际状况去调就好了，毕竟只有一个参数还是很好调的。

在这里我建议看完可以先看郭宝龙的信号与系统的工程信号与系统第三章（189—192集）对状态空间方程有一个大致的概念再去看dr.can的现控。学完现控的知识后，里面说的观测器什么的就自然而然的就理解了。

若有错误的地方可以在github上联系作者和作者讨论。或者说是发QQ邮箱到3103942481@qq.com,或谷歌邮箱Amos.lab.oss@gmail.com。出门右转还有一些关于matlab的代码笔记方便大家使用。后面其实还有重力补偿、滤波器、卡尔曼滤波器等知识点因为用的少就没有提及到，有机会的话我还会进行补充，没有机会的话就留给学弟学妹们补充了。

2025.2.18

历时一个月