EE3006.01 计算机控制基础

大作业 完整版本

2023 年秋季学期

陈镜舟 PB21061306

允许讨论, 禁止抄袭

说明: 大作业提交时间为: 12 月 22 日. 在 Matlab 或 Python 仿真, 如遇参数没有给出时可自由选择参数值完成仿真.

大作业用到的旋翼无人机参数如下:

质量 (m)	0.03 [kg]	阻力系数 (k_i) $i \in \{x, y, z\}$	4.5×10^{-3} [kg/s]
重力加速度 (g)	9.81 [m/s^2]	阻力矩系数 (k_i) $i \in \{p, q, r\}$	4.5×10^{-4} [kg $\cdot$ m^2/s]
惯性 (I_x)	1.5×10^{-5} [kg $\cdot$ m^2]	惯性 (I_y)	1.5×10^{-5} [kg $\cdot$ m^2]
惯性 (I_z)	3×10^{-5} [kg $\cdot$ m^2]	悬停高度 z_d	2 m
偏航角 (ψ)	$\frac{\pi}{4}$ [rad]		

假设四旋翼无人机在 z_d 高度以偏航角 ψ_d 悬停, 即 $[x, y, z, \psi]^T = [0, 0, z_d, \psi_d]^T$. 此时平移速度为 $[\dot{x}, \dot{y}, \dot{z}]^T = [0, 0, 0]^T$, 旋转角速度为 $[p, q, r]^T = [0, 0, 0]^T$. 此外, 滚转角和俯仰角均为零, 即 $[\phi, \theta]^T = [0, 0]^T$. 升力 u_1 等于四旋翼无人机的重力从而使得旋翼无人机不会下降高度. 滚转、俯仰以及偏航力矩均为零, 即 $u_2 = u_3 = u_4 = 0$.

取状态向量

$$\mathcal{X} = [x, y, \Delta z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \Delta\psi, p, q, r]^T$$

输入向量

$$\mathcal{U} = [\Delta u_1, u_2, u_3, u_4]^T$$

以及输出向量

$$\mathcal{Y} = [x, y, \Delta z, \Delta\psi]^T$$

其中,

$$\Delta z = z - z_d, \quad \Delta\psi = \psi - \psi_d, \quad \Delta u_1 = u_1 - mg$$

四旋翼无人机线性化动力学模型为如下状态空间表达式

$$\dot{\mathcal{X}} = A\mathcal{X} + B\mathcal{U}, \quad \mathcal{Y} = C\mathcal{X}$$

其中

$$A = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -\frac{k_x}{m} & 0 & 0 & g\sin(\psi_d) & g\cos(\psi) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\frac{k_y}{m} & 0 & -g\cos(\psi_d) & g\sin(\psi_d) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{k_z}{m} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{k_p}{I_x} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{k_q}{I_y} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{k_r}{I_z}
\end{bmatrix}$$

$$B = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\frac{1}{m} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & \frac{1}{I_x} & 0 & 0 \\
0 & 0 & \frac{1}{I_y} & 0 \\
0 & 0 & 0 & \frac{1}{I_z}
\end{bmatrix}, \quad C = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}$$

1. 利用 Matlab 或 Python 计算矩阵 A 的特征值, 并判断系统的稳定性.

   解:

$$\lambda = (0, 0, 0, -0.15, -0.15, -0.15, 0, 0, 0, -30, -30, -15)^\top.$$

   特征值实部小于等于 0，故临界稳定。

2. 给出从输入 $\Delta U_1(s)$ 到输出 $\Delta Z(s)$ 的高度控制子系统传递函数 $G_z(s)$, 以及从输入 $U_4(s)$ 到输出 $\Delta\Psi(s)$ 的偏航角控制子系统传递函数 $G_\psi(s)$.

解:

$$\dot{\Delta z} = \dot{z}$$
$$\ddot{z} = -\frac{k_z}{m}\dot{z} + \frac{1}{m}\Delta u_1$$

故

$$s^2\Delta Z = -\frac{k_z}{m}s\Delta Z + \frac{1}{m}\Delta U_1 \Rightarrow G_z = \frac{1}{ms^2 + k_z s} = \frac{1}{0.03s^2 + 4.5\times 10^{-3}s}.$$

同理,

$$\dot{\phi} = r$$
$$\dot{r} = -\frac{k_r}{I_z}r + \frac{1}{I_z}u_4$$

故

$$G_\phi = \frac{1}{I_z s^2 + k_r s} = \frac{1}{3\times 10^{-5}s^2 + 4.5\times 10^{-4}s}.$$

3. 考虑四旋翼无人机的偏航角控制子系统 $G_\psi(s)$. 用 Matlab 或 Python 绘制偏航角的时间响应曲线:

   1) 脉冲输入: $u_4(t) = 3 \times 10^{-5} \cdot \delta(t)$;

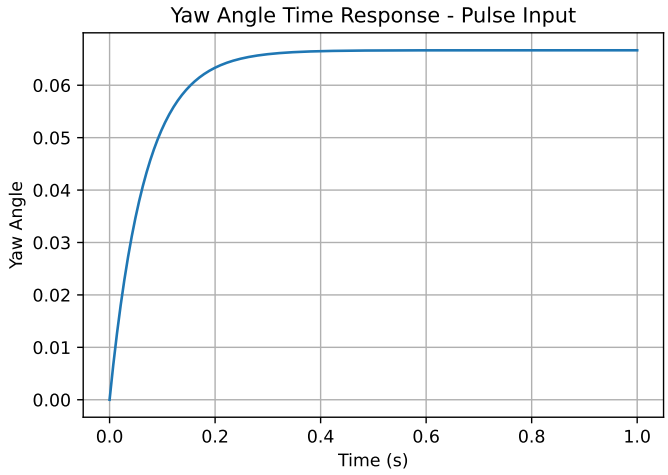
   2) 阶跃输入: $u_4(t) = 3 \times 10^{-5} \cdot 1(t)$;

   3) 正弦输入: $u_4(t) = 3 \times 10^{-5} \sin(t) \cdot 1(t)$.

   根据仿真, 给出偏航角在脉冲输入下的稳态值. 用终值定理验证该稳态值.

   解:

   1)

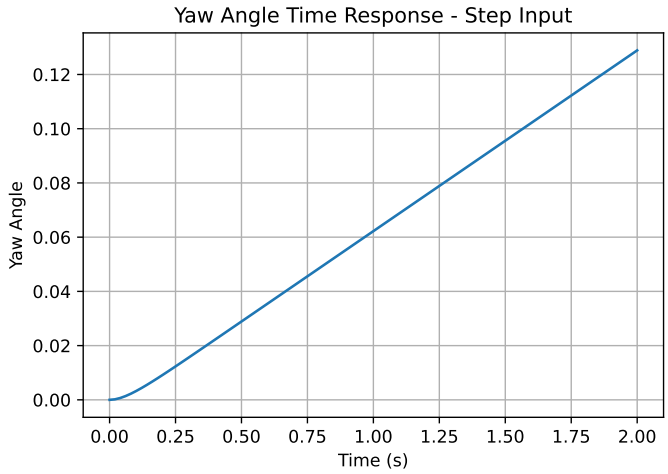


   稳态值 0.0667。

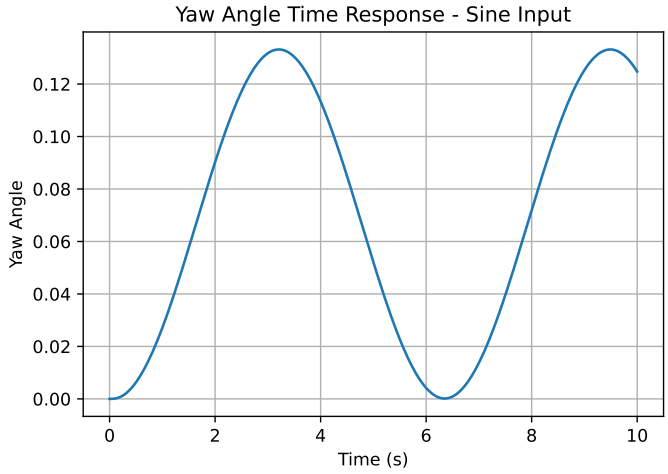
$$\Psi(s) = G_\psi(s)U_4(s) = \frac{1}{3 \times 10^{-5}s^2 + 4.5 \times 10^{-4}s} \cdot 3 \times 10^{-5} = \frac{1}{s^2 + 15s}$$

$$\Rightarrow \lim_{t \to +\infty} f(t) = \lim_{s \to 0^+} s\Psi(s) = \frac{1}{s+15}\bigg|_{s=0} = \frac{1}{15} = 0.0667$$

   2)


Yaw Angle Time Response - Step Input
Yaw Angle
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Time (s)

3)


Yaw Angle Time Response - Sine Input
Yaw Angle
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
2
4
6
8
10
Time (s)

4. 考虑四旋翼无人机的高度控制子系统 $G_z(s)$. 假设选取比例控制器 $\Delta u_1 = -k\Delta z$, $k > 0$. 选取增益 k 使得高度时间响应曲线满足以下要求 (初始条件选为 $\Delta z(0) = 1$, $\dot{z}(0) = 0$):

   1) 高度响应曲线发散;

   2) 高度响应曲线收敛到非零稳态值;

   3) 高度响应曲线收敛到零且没有振荡.

   4) 高度响应曲线经过振荡后收敛到零.

   并用 Matlab 或 Python 绘制高度时间响应曲线:

   解:

$$
\begin{aligned}
&\ddot{\Delta z} = -\frac{k_z}{m}\dot{\Delta z} - \frac{k}{m}\Delta z \\
\Rightarrow &s^2\Delta Z - s\Delta z(0_-) - \dot{\Delta z}(0_-) + \frac{k_z}{m}[s\Delta Z - \Delta z(0_-)] + \frac{k}{m}\Delta Z = 0 \\
\Rightarrow &s^2\Delta Z - s + \frac{k_z}{m}(s\Delta Z - 1) + \frac{k}{m}\Delta Z = 0 \\
\Rightarrow &\Delta Z = \frac{s + \frac{k_z}{m}}{s^2 + \frac{k_z}{m}s + \frac{k}{m}}
\end{aligned}
$$

   计算得阻尼比

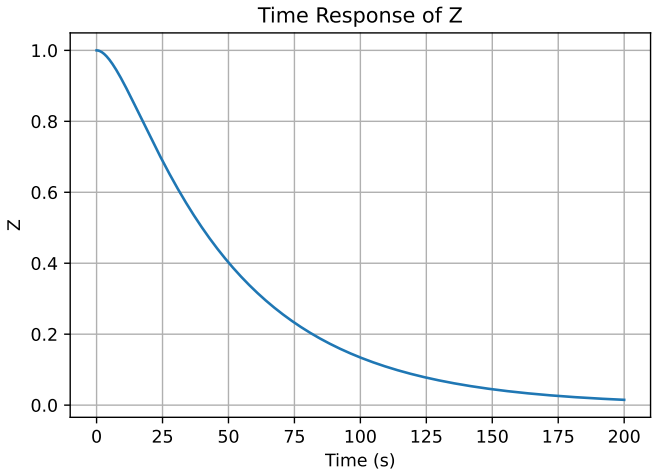
$$
\zeta = \frac{k_z}{2\sqrt{mk}}.
$$

   令 $\zeta = 1$，解得 $k_0 := \frac{k_z^2}{4m}$。

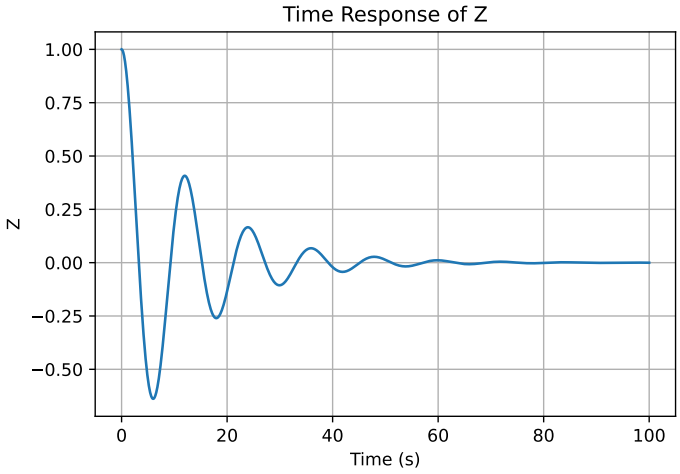
   1) 要求存在实部大于 0 的极点，无法实现。

   2) 无法实现。

   3) 过阻尼或临界阻尼，$0 < k \leq k_0$。取 $k = \frac{1}{2}k_0$ 得



4) 欠阻尼，$k > k_0$。取 $k = 50k_0$ 得



5. 考虑阻力系数为零的四旋翼无人机高度控制子系统 $k_z = 0$. 假设控制输入 Δu_1 存在 T s 时滞 (如从地面站计算的控制指令发送到无人机的通讯延时). 此时, 高度传递函数为 $\Delta(z) = \frac{\mathrm{e}^{-Ts}}{s^2}\Delta U_1(s)$. 用一阶 Pade 近似 $\mathrm{e}^{-Ts} = \frac{2-Ts}{2+Ts}$ 可以将上述传递函数近似为

$$\Delta Z(s) = \frac{2-Ts}{ms^2(2+Ts)}\Delta U_1(s)$$

是否存增益的变化范围使得在如下结构的反馈控制器 $K(s)$ 下高度子系统是稳定的 ($T = 1 \times 10^{-3}$ s)

1) $K(s) = k_p$;

2) $K(s) = k_p + k_d s$.

如果存在, 在增益容许范围内选择一个增益，用 Matlab 或 Python 绘制高度时间响应曲线 (初始值选为 $\Delta z(0) = -1$, $\dot{z}(0) = 0$).

解:

1)

$$G_c(s) = \frac{k_p(2-Ts)}{k_p(2-Ts) + ms^2(2+Ts)}$$

特征方程:

$$mTs^3 + 2ms^2 - k_pTs + 2k_p = 0$$

劳斯阵列:

$$\begin{array}{ccc} s^3 & mT & -k_pT \\ s^2 & 2m & 2k_p \\ s & -2k_pT & 0 \\ s^0 & 2k_p & 0 \end{array}$$

当 $k_p \geq 0$ 时无法实现稳定。

2)

$$G_c(s) = \frac{(k_p + k_d s)(2-Ts)}{(k_p + k_d s)(2-Ts) + ms^2(2+Ts)}$$

特征方程

$$mTs^3 + (2m - k_dT)s^2 + (2k_d - k_pT)s + 2k_p = 0$$

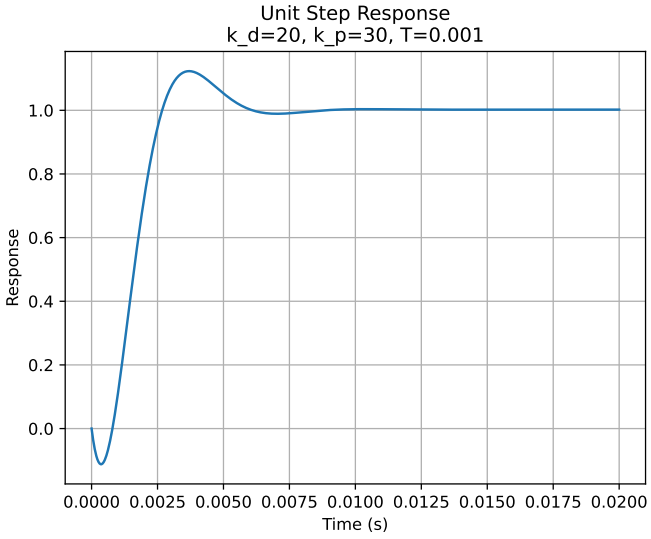
劳斯阵列

$$\begin{array}{ccc} s^3 & mT & 2k_d - Tk_p \\ s^2 & 2m - k_dT & 2k_p \\ s & 2k_d - Tk_p - \frac{2mk_pT}{2m-k_dT} & 0 \\ s^0 & 2k_p & 0 \end{array}$$

解得

$$\begin{cases} k_d < \frac{2m}{T} = 60 \\ k_p < \frac{k_d^2T-2mk_d}{2k_dT^2-2mT} = \frac{2}{T}\frac{k_d^2-2mk_d/T}{k_d-4m/T} = 2000 \times \frac{k_d^2-60k_d}{k_d-120} \\ k_p > 0 \end{cases}$$

取 $k_d = 20, k_p = 30$，不考虑初始值，绘制单位阶跃响应曲线如下图。



6. 考虑阻力系数为零 (即 $k_z = 0$) 的四旋翼无人机高度控制子系统. 假设控制输入 Δu_1 存在 T s 时滞 (如从地面站计算的控制指令发送到无人机的通讯延时). 此时, 高度传递函数为 $\Delta Z(s) = \frac{\mathrm{e}^{-Ts}}{s^2}\Delta U_1(s)$. 用一阶 Pade 近似 $\mathrm{e}^{-Ts} = \frac{2-Ts}{2+Ts}$ 可以将上述传递函数近似为

$$\Delta Z(s) = \frac{2 - Ts}{ms^2(2 + Ts)}\Delta U_1(s)$$

是否存增益的变化范围使得在如下结构的反馈控制器 $K(s)$ 下高度子系统是稳定的 ($T = 1 \times 10^{-3}$ s)

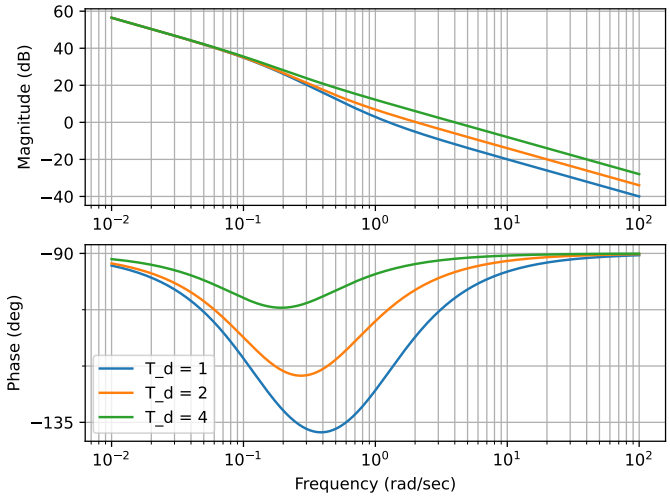
1) $K(s) = k_p$;

2) $K(s) = k_p + k_d s$.

如果存在, 在增益容许范围内选择一个增益，用 Matlab 或 Python 绘制高度时间响应曲线 (初始值选为 $\Delta z(0) = -1$, $\dot{z}(0) = 0$).

7. 考虑带有阻力的四旋翼无人机高度控制子系统 $\Delta Z(s) = \frac{1}{ms(s+\frac{k_z}{m})}\Delta U_1(s)$. 用 PD 控制器对高度控制子系统进行控制, 使得开环传递函数为 $G_{PD}(s) = k_p\frac{T_d s+1}{ms(s+\frac{k_z}{m})}$. 当 $k_p = 0.03$ 时, 分别针对微分时间常数 $T_d = 1, T_d = 2, T_d = 4$ 调用 Matlab 的 bode 指令绘制开环传递函数的伯德图. 当微分时间常数 T_d 增大时, 增益裕度和相位裕度如何变化?

解:



T_d 增大时，增益裕度保持为无穷，相角裕度增大。

8. 当偏航角 $\psi_d = 0$, 无人机在 x 方向的动力学可以通过一个单输入单输出系统描述. 具体地, 可以表达为

$$\frac{\mathrm{d}}{\mathrm{dt}}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ q \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{k_x}{m} & g & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{k_q}{I_y} \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ q \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{I_y} \end{bmatrix} u_3, \quad x = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ q \end{bmatrix}$$

其中, u_3 是由桨叶产生的俯仰力矩. 大多数商业无人机 (包括 Crazyflie) 无法直接获得 u_3, 而是设计一个期望的俯仰角速率 q_{red} 作为输入. 机载控制单元能够很快跟踪 q_{red}, 这样用户可以假定 $q \approx q_{ref}$. 这样, 以 $[x, \dot{x}, \theta]$ 为状态, 以 q 为输入可以进一步简化 x 轴方向的动力学为

$$\frac{\mathrm{d}}{\mathrm{dt}}\begin{bmatrix} x \\ \dot{x} \\ \theta \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{k_x}{m} & g \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} q, \quad x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ \theta \end{bmatrix} \tag{1}$$

1) 写出从输入 q 到输出 x 的 x 轴方向动力学模型的传递函数 $G_x(s)$.

2) 设计如下形式的超前补偿器

$$K(s) = k\frac{Ts+1}{\alpha Ts+1},\ 0.1 \le \alpha < 1$$

使得无人机 x 轴方向动力学子系统至少满足以下要求中的任意三个要求?

(I) 渐近稳定;

(II) 对阶跃参考信号的稳态跟踪误差为零;

(III) 穿越频率不小于 1.5 rad/s;

(IV) 相位裕度不小于 40°.

是否存在超前补偿使得系统满足以上四个要求?

3) 用 Matlab 或 Python 绘制无人机 x 轴方向动力学子系统的单位阶跃响应曲线.

解:

对 $\ddot{x} = -\frac{k_x}{m}\dot{x} + g\theta$ 两侧求导，并代入 $\dot{\theta} = q$，然后进行拉普拉斯变换，得

$$s^3X + \frac{k_x}{m}s^2X = gQ \Rightarrow G_x(s) = \frac{g}{s^3 + \frac{k_x}{m}s^2}.$$

与超前补偿器串联得开环传递函数

$$G_o(s)=\frac{kg(Ts+1)}{s^2(s+\frac{k_x}{m})(\alpha Ts+1)}=\frac{kgTs+kg}{\alpha Ts^4+(\frac{k_x}{m}\alpha T+1)s^3+\frac{k_x}{m}s^2}.$$

这是 2 型系统，对阶跃信号的稳态跟踪误差为 0。

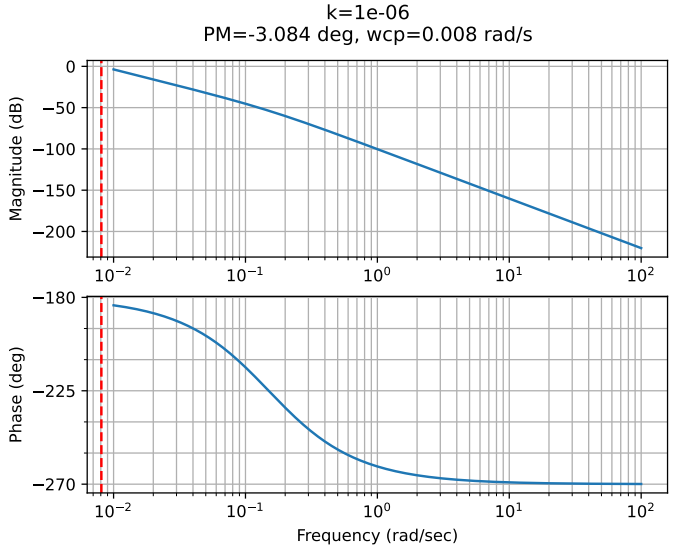
闭环传递函数对应的特征方程为

$$\alpha Ts^4+(\frac{k_x}{m}\alpha T+1)s^3+\frac{k_x}{m}s^2+kgTs+kg=0.$$

劳斯阵列为

s^4	αT	$\frac{k_x}{m}$	kg
s^3	$\frac{k_x}{m}\alpha T+1$	kgT	
s^2	$\frac{k_x}{m}-\frac{\alpha kgT^2}{\frac{k_x}{m}\alpha T+1}$	kg	
s	$kgT-\frac{(\frac{k_x}{m}\alpha T+1)^2kg}{\frac{k_x}{m}(\frac{k_x}{m}\alpha T+1)-\alpha kgT^2}$		
s^0	kg		

考虑相位裕度 $\geq 40°$，取 $k=1\times 10^{-6}$，绘制 Bode 图如下，得到 $kG_x(s)$ 系统的相位裕度 $PM=-3.084°$，穿越频率为 $\omega_c=0.008$ rad/s。



取 $\alpha=0.1$，此时

$$\arcsin\left(\frac{1-\alpha}{1+\alpha}\right)=\arcsin\left(\frac{1-0.1}{1+0.1}\right)\approx 55°.$$

因此

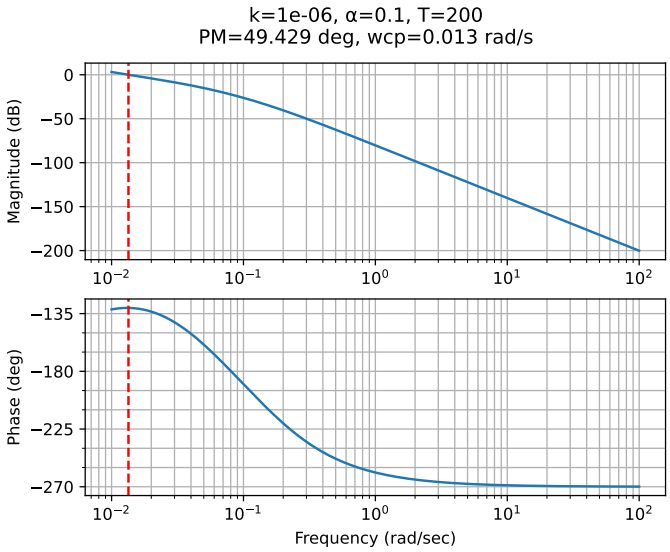
$$\frac{1}{T\sqrt{\alpha}} > \omega_c \Rightarrow T < \frac{1}{\omega_c\sqrt{\alpha}} \approx 395.$$

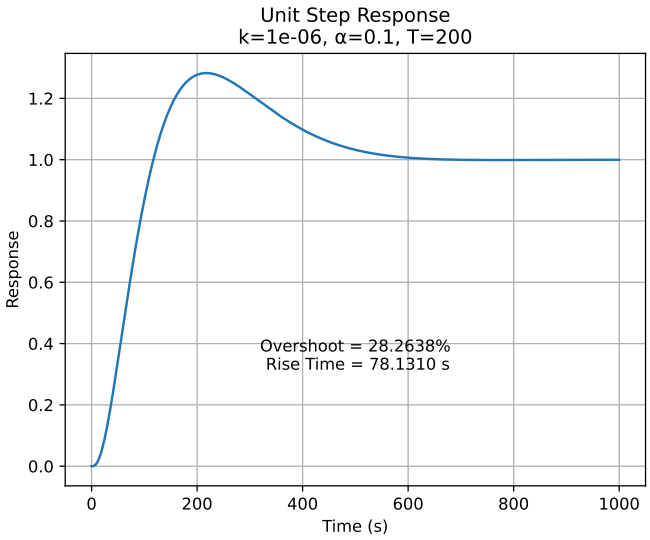
取 $T = 200$。

将

$$k = 1 \times 10^{-6},\ \alpha = 0.1,\ T = 200$$

代入劳斯阵列计算，发现第一列均大于零，因此系统是稳定的，进而可知系统是渐近稳定的。绘制 Bode 图和单位阶跃响应如下。注意到条件中对穿越频率和相位裕度的要求是互斥的，无法同时满足。




Unit Step Response
k=1e-06, α=0.1, T=200
Response
Time (s)
Overshoot = 28.2638%
Rise Time = 78.1310 s
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
200
400
600
800
1000

9. 在实际中, 运动捕捉系统能够提供 Crazyflie 无人机的俯仰角信息. 基于此，我们可以采用级联控制结构实现对 x 轴方向的控制. 如图 1 所示, 内环控制器 (可以简单选择为比例控制) 能够快速跟踪期望的俯仰角 θ_{ref}, 外环控制器通过 θ_{ref} 控制 Crazyflie 无人机 x 轴方向的位置.

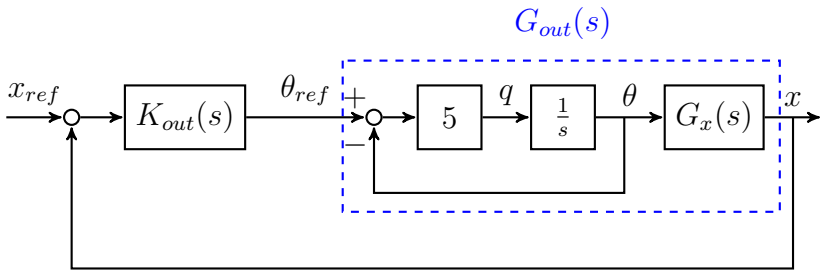


图 1. 级联控制结构

1) 针对无人机 x 轴方向动力学与内环控制组成的系统 $G_{out}(s)$, 设计如下形式的超前补偿器

$$K_{out}(s) = k\frac{Ts+1}{\alpha Ts+1},\ 0.1 \le \alpha < 1$$

使得无人机 x 轴方向动力学子系统满足以下要求

(I) 渐近稳定;

(II) 对阶跃参考信号的稳态跟踪误差为零;

(III) 穿越频率不小于 1.5 rad/s.

2) 用 Matlab 或 Python 绘制无人机 x 轴方向动力学子系统的单位阶跃响应曲线.

解:

由于此处 $G_x(s)$ 的输入为 θ，故取 $G_x(s) = \frac{g}{s^2+\frac{k_x}{m}s}$

$$G_1(s) = \frac{5/s}{1+5/s} = \frac{5}{s+5}$$

$$G_{\text{out}}(s) = G_1(s)G_x(s) = \frac{5}{s+5}\cdot\frac{g}{s^2+\frac{k_x}{m}s} = \frac{5g}{s(s+\frac{k_x}{m})(s+5)}$$

$$\begin{aligned} G_o(s) =& K_{\text{out}}(s)G_{\text{out}}(s) = \frac{5kg(Ts+1)}{s(s+\frac{k_x}{m})(s+5)(\alpha Ts+1)} \\ =& \frac{5kgTs+5kg}{\alpha Ts^4+[(5+\frac{k_x}{m})\alpha T+1]s^3+(5+\frac{k_x}{m}+5\frac{k_x}{m}\alpha T)s^2+5\frac{k_x}{m}s} \end{aligned}$$

特征方程

$$\alpha T s^4 + [(5 + \frac{k_x}{m})\alpha T + 1]s^3 + (5 + \frac{k_x}{m} + 5\frac{k_x}{m}\alpha T)s^2 + (5\frac{k_x}{m} + 5kgT)s + 5kg = 0$$

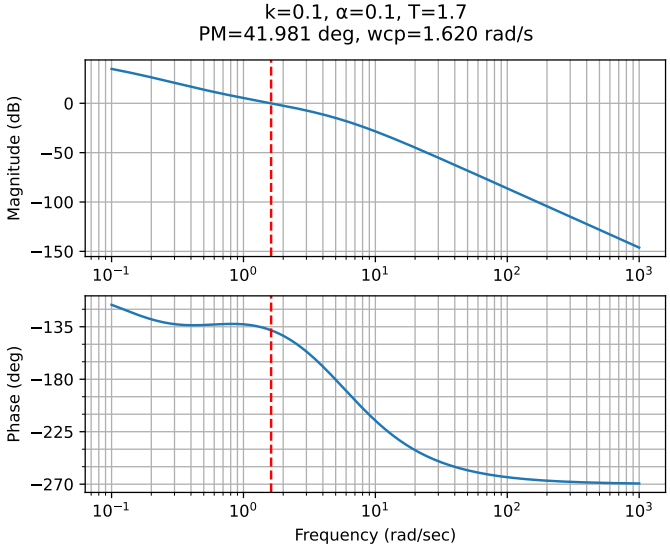
劳斯阵列

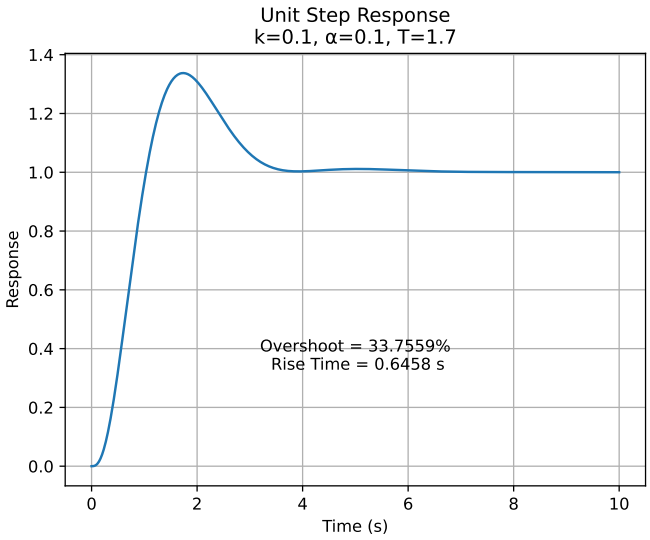
$$\begin{array}{cccc}
s^4 & \alpha T & 5 + \frac{k_x}{m} + 5\frac{k_x}{m}\alpha T & 5kg \\
s^3 & (5 + \frac{k_x}{m})\alpha T + 1 & 5\frac{k_x}{m} + 5kgT & \\
s^2 & 5 + \frac{k_x}{m} + 5\frac{k_x}{m}\alpha T - \frac{\alpha T(5\frac{k_x}{m} + 5kgT)}{(5+\frac{k_x}{m})\alpha T+1} & 5kg & \\
s & 5\frac{k_x}{m} + 5kgT - \frac{5kg[(5+\frac{k_x}{m})\alpha T+1]}{5+\frac{k_x}{m}+5\frac{k_x}{m}\alpha T-\frac{\alpha T(5\frac{k_x}{m}+5kgT)}{(5+\frac{k_x}{m})\alpha T+1}} & & \\
s^0 & 5kg & &
\end{array}$$

G_o 是 1 型系统，对阶跃参考信号的稳态跟踪误差为零。令 $|G_o(s)| = 1$ 得

$$T = \sqrt{\frac{\omega_c^2(\omega_c^2 + (k_x/m)^2)(\omega_c^2 + 25) - (5gk)^2}{(5gk\omega_c)^2 - \omega_c^4\alpha^2(\omega_c^2 + (k_x/m)^2)(\omega_c^2 + 25)}}$$

取 $\omega_c = 1.6, \alpha = 0.1, k = 0.1$，由上式计算得 $T \approx 1.7$。计算得劳斯阵列第一列均大于 0，因此系统是稳定的，进而可知系统是渐近稳定的。绘制 Bode 图和单位阶跃响应曲线如下。




Unit Step Response
k=0.1, α=0.1, T=1.7
Response
Time (s)
Overshoot = 33.7559%
Rise Time = 0.6458 s
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
2
4
6
8
10

10. 考虑 8 中的状态空间系统 (1).

1) 判断系统的能控性.
2) 寻找反馈控制律 $q = -K[x, \dot{x}, \theta]^T$ 使得闭环系统的极点配置为 $-0.7081, -0.5210 \pm j1.068$.
3) 设计反馈控制 $u = -Kx + Nx_{ref}$ 使得系统对单位阶跃参考输入 $x_{ref}(t) = \mathbf{1}(t)$ 响应的稳态误差为零.
4) 用 Matlab 或 Python 绘制无人机 x 轴方向动力学子系统的单位阶跃响应曲线.
5) 从超调量和上升时间等角度, 比较本部分控制与 8 和 9 中控制得到的闭环系统性能.

解:

1)

$$\mathcal{C} = [B, AB, A^2B] = \begin{bmatrix} 0 & 0 & g \\ 0 & g & -g\frac{k_x}{m} \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 9.81 \\ 0 & 9.81 & -1.4715 \\ 1 & 0 & 0 \end{bmatrix}$$

行满秩，故能控。

2)

$$\mathcal{C}^{-1} = \begin{bmatrix} 0 & 0 & 1 \\ 0.01529052 & 0.1019368 & 0 \\ 0.1019368 & 0 & 0 \end{bmatrix}$$

$$\alpha_c(s) = s^3 + 1.7501s^2 + 2.1498s + 0.9998372$$

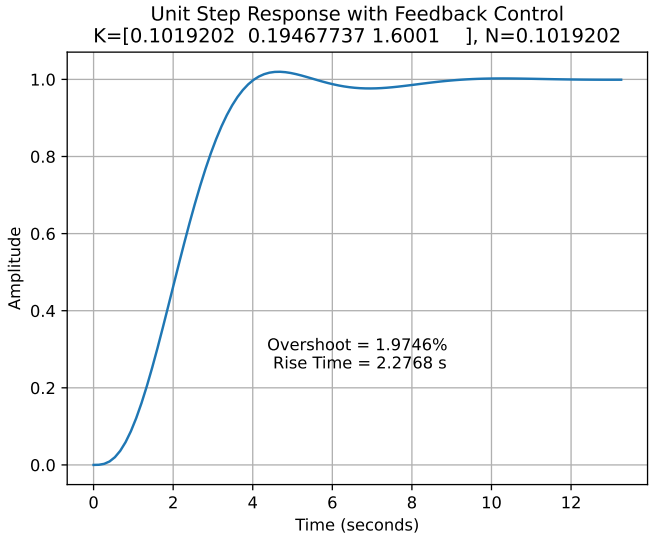
$$\alpha_c(A) = \begin{bmatrix} 0.9998372 & 1.909785 & 15.696981 \\ 0 & 0.71336945 & 18.73499085 \\ 0 & 0 & 0.9998372 \end{bmatrix}$$

$$K = [0, 0, 1]\mathcal{C}^{-1}\alpha_c(A) = [0.1019202, 0.19467737, 1.6001]$$

3)

$$N = -\frac{1}{C(A - BK)^{-1}B} = 0.1019202$$

4) 使用上述 K 和 N 得到 $A' = A - BK, B' = BN$，绘制单位阶跃曲线如下图。



5) 本部分控制的超调量远小于 8 和 9 中的控制，上升时间远小于 8 中的控制，但略大于 9 中的控制。整体来说本部分控制的性能最好。

代码

1.

```python
import numpy as np

m = 0.03
k_x = 4.5e-3
k_y = 4.5e-3
k_z = 4.5e-3
k_p = 4.5e-4
k_q = 4.5e-4
k_r = 4.5e-4
g = 9.81
psi_d = np.pi / 4
I_x = 1.5e-5
I_y = 1.5e-5
I_z = 3e-5
z_d = 2

A = np.array([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
              [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
              [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
              [0, 0, 0, -k_x / m, 0, 0, g * np.sin(psi_d), g * np.cos(
                                        psi_d), 0, 0, 0, 0],
              [0, 0, 0, 0, -k_y / m, 0, -g * np.cos(psi_d), g * np.sin(
                                        psi_d), 0, 0, 0, 0],
              [0, 0, 0, 0, 0, -k_z / m, 0, 0, 0, 0, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
              [0, 0, 0, 0, 0, 0, 0, 0, 0, -k_p / I_x, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -k_q / I_y, 0],
              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -k_r / I_z]])

eigenvalues = np.linalg.eigvals(A)
eigenvalues
```

2.

```python
import numpy as np
import control
import matplotlib.pyplot as plt

# System parameters
I_z = 3e-5
k_r = 4.5e-4

# Define the transfer function
num = [1]
den = [I_z, k_r, 0]
sys = control.TransferFunction(num, den)

# Define the time vector
t1 = np.linspace(0, 1, 1000)
t2 = np.linspace(0, 1, 1000)
t3 = np.linspace(0, 10, 1000)

# Define the input signals
u_sine = 3e-5 * np.sin(t3) * np.heaviside(t3, 1)

# Compute the time response
t_pulse, y_pulse = control.impulse_response(sys, T=t1, X0=0)
t_step, y_step = control.step_response(sys, T=2, X0=0)
t_sine, y_sine = control.forced_response(sys, T=t3, U=u_sine)
y_pulse = y_pulse * 3e-5
y_step = y_step * 3e-5

# Plot the time response - Pulse Input
plt.figure(figsize=(6, 4))
plt.plot(t_pulse, y_pulse)
plt.title('Yaw Angle Time Response - Pulse Input')
plt.xlabel('Time (s)')
plt.ylabel('Yaw Angle')
plt.grid(True)
plt.savefig('./images/3/yaw_angle_pulse.pdf')
plt.show()

```

```
# Plot the time response - Step Input
plt.figure(figsize=(6, 4))
plt.plot(t_step, y_step)
plt.title('Yaw Angle Time Response - Step Input')
plt.xlabel('Time (s)')
plt.ylabel('Yaw Angle')
plt.grid(True)
plt.savefig('./images/3/yaw_angle_step.pdf')
plt.show()

# Plot the time response - Sine Input
plt.figure(figsize=(6, 4))
plt.plot(t_sine, y_sine)
plt.title('Yaw Angle Time Response - Sine Input')
plt.xlabel('Time (s)')
plt.ylabel('Yaw Angle')
plt.grid(True)
plt.savefig('./images/3/yaw_angle_sine.pdf')
plt.show()
```

3.

4.

```
import numpy as np
import control
import matplotlib.pyplot as plt

# System parameters
k_z = 4.5e-3
m = 0.03
k_0=k_z**2/(4*m)
k = k_0*0.5
# k = k_0*50

# Define the transfer function
num = [1, k_z / m]
```

```
den = [1, k_z / m, k / m]
sys = control.TransferFunction(num, den)

# Define the time vector
t = np.linspace(0, 200, 10000)

# Compute the time response
t, y = control.impulse_response(sys, T=t)

# Plot the time response
plt.figure(figsize=(6, 4))
plt.plot(t, y)
plt.title('Time Response of Z')
plt.xlabel('Time (s)')
plt.ylabel('Z')
plt.grid(True)

# Automatically set y-axis limits
height = np.max(y) - np.min(y)
alpha=0.05
plt.ylim(np.min(y)-height*alpha, np.max(y)+height*alpha)
plt.savefig('./images/4/3.pdf')
plt.show()
```

5.

```
import numpy as np
import control
import matplotlib.pyplot as plt

# System parameters
k_d = 40
k_p = 30
T = 0.001
m = 0.03

# Define the transfer function
num = [-k_d * T, (2 * k_d - k_p * T), 2 * k_p]
```

```
den = [m * T, (2 * m - k_d * T), (2 * k_d - k_p * T), 2 * k_p]
sys = control.TransferFunction(num, den)

# Define the initial conditions
z0 = -1
z_dot0 = 0

# Reshape the initial conditions
X0 = np.array([[z0], [z_dot0], [0]])

# # Compute the zero-input response
# t, y = control.initial_response(sys, T=t, X0=X0)

# # Plot the response curve
# plt.figure(figsize=(6, 4))
# plt.plot(t, y)
# plt.title('Zero-Input Response')
# plt.xlabel('Time (s)')
# plt.ylabel('Response')
# plt.grid(True)
# plt.show()

t, y = control.step_response(sys,T=np.linspace(0, 0.02, 1000))
plt.plot(t, y)
plt.title(f'Unit Step Response\nk_d={k_d}, k_p={k_p}, T={T}')
plt.xlabel('Time (s)')
plt.ylabel('Response')
plt.grid(True)
plt.savefig(f'./images/5/step_response_kd={k_d}_kp={k_p}_T={T}.pdf')
plt.show()
```

6.

7.

```
import numpy as np
import control
```

```
import matplotlib.pyplot as plt

k_p = 0.03
T_d_values = [1, 2, 4]
m = 0.03
k_z = 4.5e-3

for T_d in T_d_values:
    sys = control.TransferFunction([k_p * T_d, k_p], [m, k_z, 0])
    # control.bode_plot(sys, dB=True)
    control.bode(sys, dB=True, omega_limits=(1e-2, 1e2))

plt.legend(['T_d = 1', 'T_d = 2', 'T_d = 4'])
plt.savefig('./images/7/bode_plot.pdf')
plt.show()
```

8.

```
# 原系统乘以k
import numpy as np
import control
import matplotlib.pyplot as plt

# System parameters
g = 9.81
k_x = 4.5e-3
m = 0.03
k = 1e-6
# k = 1

# Define the transfer function
num = [g * k]
den = [1, k_x / m, 0, 0]
sys = control.TransferFunction(num, den)

# Calculate the system's margins
gm, pm, wcg, wcp = control.margin(sys)
print(f"Phase Margin: {pm} degrees")
```

```
print(f"Crossover Frequency: {wcp} rad/s")

# Bode plot
control.bode(sys, dB=True, omega_limits=(1e-2, 1e2))
for ax in plt.gcf().get_axes():
    ax.axvline(x=wcp, color='r', linestyle='--', label='Crossover
                                        Frequency')
plt.suptitle(f"k={k}\nPM={pm:.3f} deg, wcp={wcp:.3f} rad/s")
plt.grid(True)
plt.savefig(f'./images/8/bode_k={k}.pdf')
plt.show()
```

```
# 增加了超前补偿器的系统
import numpy as np
import control
import matplotlib.pyplot as plt

# System parameters
g = 9.81
k_x = 4.5e-3
m = 0.03
k = 1e-6
alpha = 0.1
T = 200

# Define the transfer function
num = [k * g * T, k * g]
den = [alpha * T, (k_x * alpha * T / m + 1), k_x / m, 0, 0]
sys = control.TransferFunction(num, den)

# Calculate the system's margins
gm, pm, wcg, wcp = control.margin(sys)
print(f"Phase Margin: {pm} degrees")
print(f"Crossover Frequency: {wcp} rad/s")

# Bode plot
control.bode(sys, dB=True, omega_limits=(1e-2, 1e2))
for ax in plt.gcf().get_axes():
```

```
    ax.axvline(x=wcp, color='r', linestyle='--', label='Crossover
                                        Frequency')
plt.suptitle(f"k={k},  ={alpha}, T={T}\nPM={pm:.3f} deg, wcp={wcp:.3f}
                                        rad/s")
plt.grid(True)
plt.savefig(f'./images/8/bode_k={k}_alpha={alpha}_T={T}.pdf')
plt.show()

# Create the closed-loop system
closed_loop_sys = control.feedback(sys)
info = control.step_info(closed_loop_sys)
overshoot = info['Overshoot']
rise_time = info['RiseTime']

t, y = control.step_response(closed_loop_sys,T=np.linspace(0, 1000, 1000)
                                        )
plt.plot(t, y)
plt.title(f'Unit Step Response\nk={k},  ={alpha}, T={T}')
plt.xlabel('Time (s)')
plt.ylabel('Response')
plt.grid(True)
plt.text(0.5, 0.3, f'Overshoot = {overshoot:.4f}%\n Rise Time = {
                                        rise_time:.4f} s', ha='center', va
                                        ='center', transform=plt.gca().
                                        transAxes)
plt.savefig(f'./images/8/step_response_k={k}_alpha={alpha}_T={T}.pdf')
plt.show()
```

9.

```
# 增加了超前补偿器的系统
import numpy as np
import control
import matplotlib.pyplot as plt

# System parameters
g = 9.81
k_x = 4.5e-3
```

```
m = 0.03

k = 0.1
alpha = 0.1
T = 1.7

# Define the transfer function
num = [5 * g]
den = [1, 5 + k_x / m, 5 * k_x / m, 0]

num=[5*g*k*T, 5*k*g]
den=[alpha*T, 1+alpha*T*(5+k_x/m), 5*k_x*alpha*T/m+5+k_x/m, 5*k_x/m, 0]
sys = control.TransferFunction(num, den)

# Calculate the system's margins
gm, pm, wcg, wcp = control.margin(sys)
print(f"Phase Margin: {pm} degrees")
print(f"Crossover Frequency: {wcp} rad/s")

# Plot the Bode plot of the closed-loop system
control.bode(sys, dB=True, omega_limits=(1e-1, 1e3))
# plt.axvline(x=wcp, color='r', linestyle='--', label='Crossover
                                        Frequency')
for ax in plt.gcf().get_axes():
    ax.axvline(x=wcp, color='r', linestyle='--', label='Crossover
                                        Frequency')
# plt.legend()
plt.suptitle(f"k={k},  ={alpha}, T={T}\nPM={pm:.3f} deg, wcp={wcp:.3f}
                                        rad/s")
plt.grid(True)
plt.savefig(f'./images/9/bode_k={k}_alpha={alpha}_T={T}.pdf')
plt.show()

# Create the closed-loop system
closed_loop_sys = control.feedback(sys)
info = control.step_info(closed_loop_sys)
overshoot = info['Overshoot']
rise_time = info['RiseTime']

```

```
t, y = control.step_response(closed_loop_sys,T=np.linspace(0, 10, 1000))
plt.plot(t, y)
plt.title(f'Unit Step Response\nk={k},  ={alpha}, T={T}')
plt.xlabel('Time (s)')
plt.ylabel('Response')
plt.grid(True)
plt.text(0.5, 0.3, f'Overshoot = {overshoot:.4f}%\n Rise Time = {
                                        rise_time:.4f} s', ha='center', va
                                        ='center', transform=plt.gca().
                                        transAxes)
plt.savefig(f'./images/9/step_response_k={k}_alpha={alpha}_T={T}.pdf')
plt.show()
```

10.

```
# 方法1
import numpy as np
g = 9.81
k_x = 4.5e-3
m = 0.03

A = np.array([[0, 1, 0], [0, -k_x/m, g], [0, 0, 0]])
B = np.array([0, 0, 1])
AB = np.dot(A, B)
A2B = np.dot(np.dot(A, A), B)
Cl = np.column_stack((B, AB, A2B))
Cl_inv = np.linalg.inv(Cl)
print("Cl:")
print(Cl)
print("Inverse of Cl:")
print(Cl_inv)

alpha_A = A@A@A+1.7501*A@A+2.1498*A+0.9998372*np.eye(3)
print("alpha_A:")
print(alpha_A)

K = np.dot(np.dot(np.array([0, 0, 1]), np.linalg.inv(Cl)), alpha_A)
K = np.reshape(K, (1, 3))
```

```
print("K:")
print(K)
```

```
# 方法2
import numpy as np
import control
# 已知状态空间对应的矩阵A,B,寻找反馈控制率使得闭环系统的极点配置为~$-0.
                                                    7081,-0.5210\pm \jmath1.068$
g = 9.81
k_x = 4.5e-3
m = 0.03

A = np.array([[0, 1, 0], [0, -k_x/m, g], [0, 0, 0]])
B = np.array([0, 0, 1])

# Reshape A and B to be 2-dimensional arrays
A = np.reshape(A, (3, 3))
B = np.reshape(B, (3, 1))

desired_poles = [-0.7081, -0.5210 + 1.068j, -0.5210 - 1.068j]

K = control.place(A, B, desired_poles)

print("Feedback Control Gain (K):")
print(K)
```

```
import numpy as np
import matplotlib.pyplot as plt
import control

g = 9.81
k_x = 4.5e-3
m = 0.03

A = np.array([[0, 1, 0], [0, -k_x/m, g], [0, 0, 0]])
B = np.array([[0], [0], [1]])
C = np.array([[1, 0, 0]])
D = np.array([[0]])
# K = np.array([[0.1019249, 0.19468809, 1.6001]])

```

```
N=-1/np.dot(np.dot(C,np.linalg.inv(A-np.dot(B,K))),B)
print(N)

feedback_A = A - np.dot(B, K)
feedback_B = np.dot(B, N)

feedback_sys2 = control.ss(feedback_A, feedback_B, C, D)

# 使用step_info()函数获取单位阶跃响应的性能指标
info = control.step_info(feedback_sys2)
overshoot = info['Overshoot']
rise_time = info['RiseTime']
# print(f'Overshoot = {overshoot:.4f}%')
# print(f'Rise Time = {rise_time:.4f} s')

t, y = control.step_response(feedback_sys2)
plt.plot(t, y)
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.title(f'Unit Step Response with Feedback Control\nK={K[0]}, N={N[0][0
                                        ]:.7f}')
plt.grid(True)
plt.text(0.5, 0.3, f'Overshoot = {overshoot:.4f}%\n Rise Time = {
                                        rise_time:.4f} s', ha='center', va
                                        ='center', transform=plt.gca().
                                        transAxes)
plt.savefig(f'./images/10/3step_response.pdf')
plt.show()

poles = control.pole(feedback_sys2)
print('Poles:', poles)
```