Problem 1: Closed loop controller tuning

In this exercise, you should tune a PI controller for an *unknown plant* using the Ziegler-Nichols method and Shams' method (Model from Closed-Loop Setpoint Response).

You can obtain more information about the Shams' method here:
http://www.nt.ntnu.no/users/skoge/publications/2012/skogestad-improved-simc-pid/

The process (*unknown plant*) is given to you as a Simulink file. More information about how to use the Simulink model is given in the comments in the file 'runModel.m'.

Tasks

- Obtain the PI controller settings using:
  1. the Ziegler-Nichols tuning rules
  2. Shams' method + SIMC rules

1. ZN tuning

To get P-control, flick the switch as explained in the code

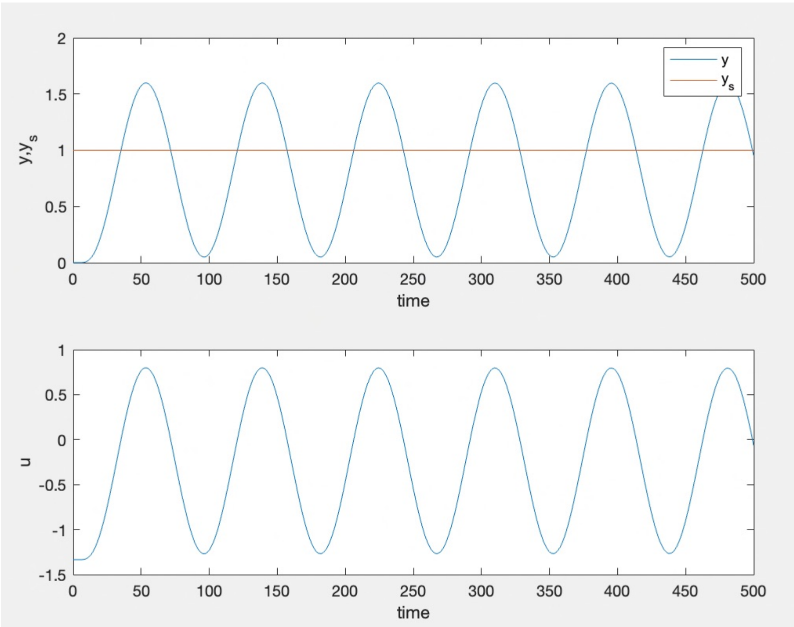
By trial and error, the oscillations are constant when:

$K_c = K_{cu} = -1{,}335$

Table 11.4 Controller Settings based on the Continuous Cycling Method

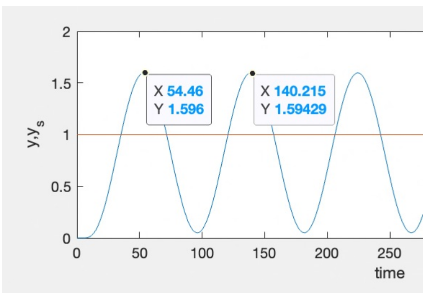
Ziegler-Nichols	K_c	τ_I	τ_D
P	$0.5K_{cu}$	—	—
PI	$0.45K_{cu}$	$P_u/1.2$	—
PID	$0.6K_{cu}$	$P_u/2$	$P_u/8$

PID is for ideal form



Using the tuning rules in table 11,4 $K_c = 0{,}45 K_{cu}$

The period P_u is read of the plot: $\underline{\underline{K_c = -0{,}60075}}$

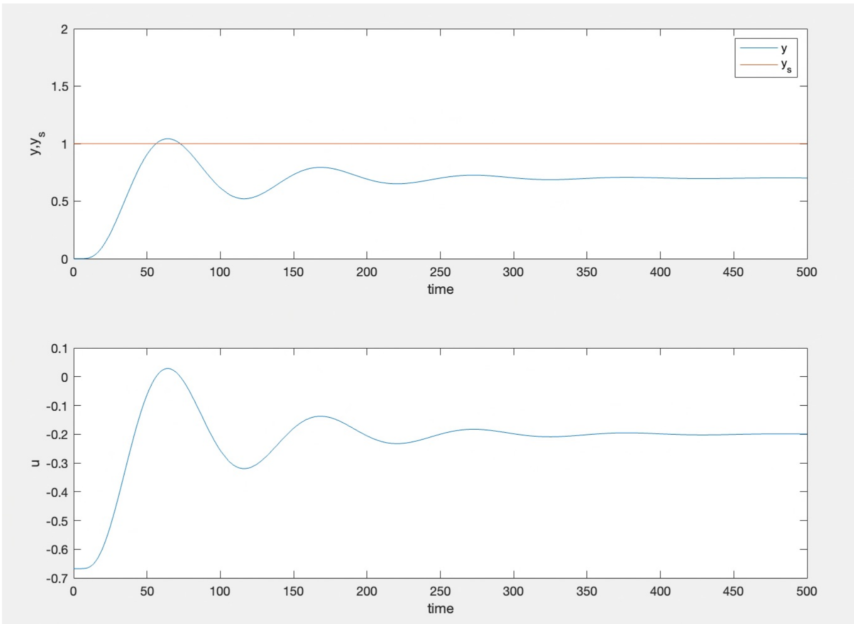


$$P_u \approx 140{,}2 - 54{,}5 = 85{,}7$$

$$\Rightarrow \tau_I = \frac{P_u}{1{,}2} = 71{,}42$$

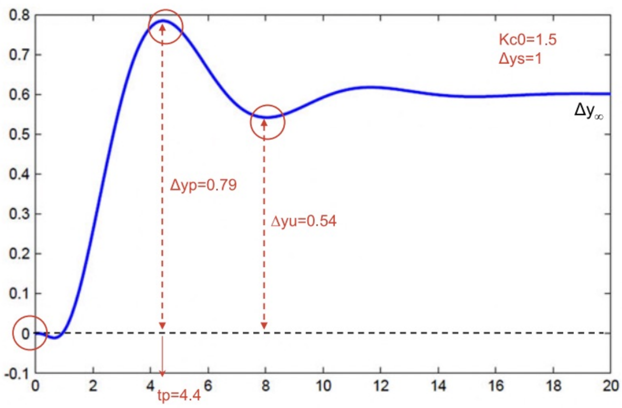
2. Shams method

We use $K_c = \frac{K_{cu}}{2} \approx -0{,}6675$



We then obtain the other necessary values:

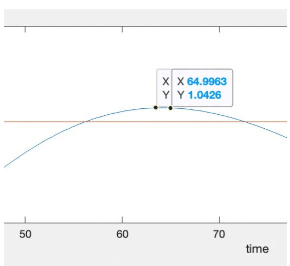
- Controller gain used in experiment, K_{c0}.
- Setpoint change, Δy_s.
- Time from setpoint change to reach first (maximum) peak, t_p.
- Corresponding maximum output change, Δy_p.
- Output change at first undershoot, Δy_u.

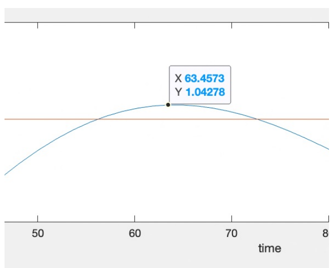


$$K_{c0} = -0{,}6675$$

$$\Delta y_s = 1$$

The peak is approximately halfway between these points:



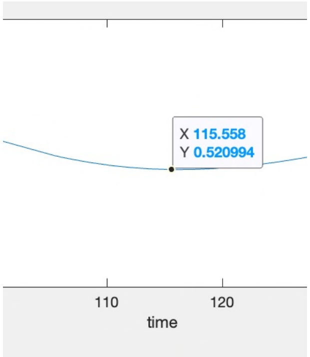


$$\Rightarrow t_p = \frac{63{,}4573 + 64{,}9963}{2} = 64{,}23$$

$$\Rightarrow \Delta y_p = \frac{1{,}04278 + 1{,}0426}{2} = 1{,}043$$

At the first undershoot

$$\Delta y_u = 0{,}521$$



Then, using the provided formulas:

$$\Delta y_\infty = 0.45(\Delta y_p + \Delta y_u). = 0{,}7037$$

- Overshoot, $D = \frac{\Delta y_p - \Delta y_\infty}{\Delta y_\infty}. = 0{,}4820$
- Steady-state offset, $B = |\frac{\Delta y_s - \Delta y_\infty}{\Delta y_\infty}|. = 0{,}4209$

$$A = 1.152D^2 - 1.607D + 1, = 0{,}4931$$

$$r = 2A/B = 2{,}3432$$

We can then use these values to get a 1st order model:

$$k = 1/(K_{c0}B), = -3{,}5597$$

$$\theta = t_p \cdot (0.309 + 0.209e^{-0.61r}), = 23{,}0616$$

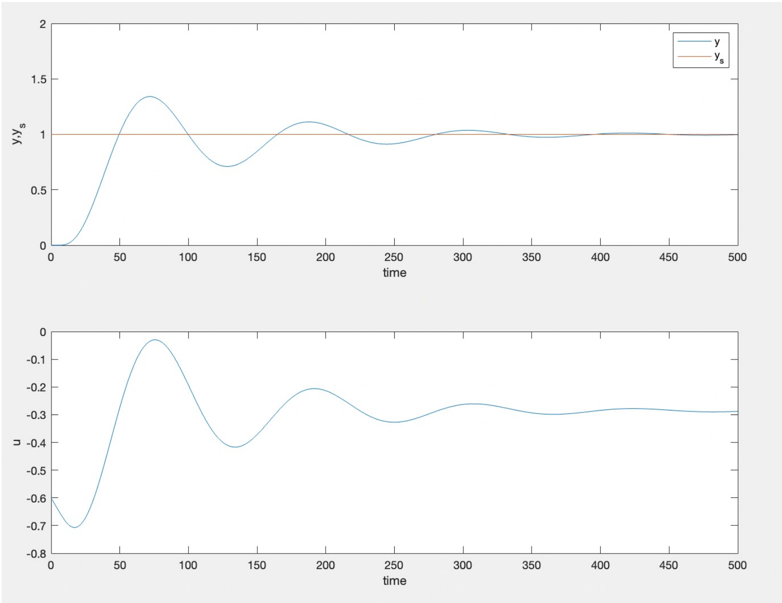
$$\tau_1 = r\theta. = 54{,}0388$$

Then, applying SIMC rules with tight control ($\tau_c = \theta$)

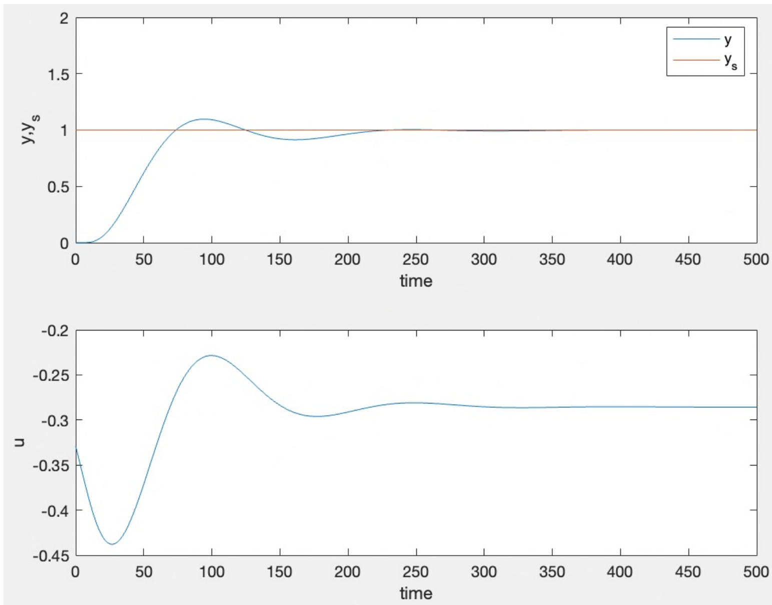
$$K_c = \frac{1}{k}\frac{\tau_1}{\tau_c + \theta} = \frac{1}{-3{,}5597} \cdot \frac{54{,}0388}{2 \cdot 23{,}0616} = \underline{\underline{-0{,}3291}}$$

$$\tau_I = \min\{\tau_1, 4(\tau_c + \theta)\}. = \tau_1 = 54{,}0388$$

- Simulate a step change in the setpoint using a PI controller with the settings you obtained using the Ziegler-Nichols tuning rules.



- Simulate a step change in the setpoint using a PI controller with the settings you obtained using Shams' method + SIMC rules.



- **Compare the results.**

ZN-tuning has a larger overshoot and oscillates more than the Shams+SIMC-tuning does.

Problem 2: Discrete filter

A filter is used to reduce the effect of the noise on the measurements. A typical filter is first-order, as shown in Fig. 1, where the y^m is the noisy measurement, y is the noise-free measurement, and τ_F is the filter time constant. In practice a digital (or discrete) version of the filter is implemented.

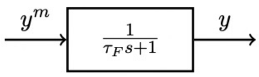


Figure 1: First-order measurement filter

Derive the discrete (digital) expression for the filter for the case where the filter time constant is $\tau_F = 5$ and the sampling time is $\Delta T = 1$.

$$y = \frac{y^m}{\tau_F s+1}$$

$$\tau_F s\, y(s) + y(s) = y_m(s) \quad / \text{Inverse laplace}$$

$$\tau_F \frac{dy}{dt} + y = y_m(t)$$

For discrete/digital approximations: $\frac{dy}{dt} \approx \frac{y_k - y_{k-1}}{\Delta t}$, $y(t) = y_k$

$$\Rightarrow \frac{\tau_F}{\Delta t}(y_k - y_{k-1}) + y_k = y_{m,k}$$

rearranging:

$$y_k\left(\frac{\tau_F}{\Delta t}+1\right) = y_{m,k} + \frac{\tau_F}{\Delta t}\, y_{k-1}$$

$$y_k = \frac{1}{\frac{\tau_F}{\Delta t}+1}\, y_{m,k} + \frac{\frac{\tau_F}{\Delta t}}{\frac{\tau_F}{\Delta t}+1}\, y_{k-1}$$

$$\boxed{\text{Let } \alpha = \frac{1}{\frac{\tau_F}{\Delta t}+1},\quad 1-\alpha = \frac{\frac{\tau_F}{\Delta t}+1}{\frac{\tau_F}{\Delta t}+1} - \frac{1}{\frac{\tau_F}{\Delta t}+1} = \frac{\frac{\tau_F}{\Delta t}}{\frac{\tau_F}{\Delta t}+1}}$$

$$\Rightarrow y_k = \alpha\, y_{m,k} + (1-\alpha)\, y_{k-1}$$

Inserting values in to $\alpha \Rightarrow \alpha = \frac{1}{\frac{5}{1}+1} = \frac{1}{6} \approx 0{,}167$

$$\Rightarrow \underline{\underline{y_k = 0{,}167\, y_{m,k} + 0{,}833\, y_{k-1}}}$$

To understand better why this is called an *exponentially moving average* filter, compute how the predicted output (y_k) depends on the previous 10 inputs:

$$y_k = k_0 y_k^m + k_1 y_{k-1}^m + \ldots + k_9 y_{k-9}^m + k_{10} y_{k-10}. \tag{1}$$

This is, find $k_0, k_1, \ldots, k_{10}$. Note that k_{10} is the weight on the stored output value at $k-10$.

Inserting the expressions for y_{k-i}, we see that:

$$y_k = \alpha\, y_{m,k} + (1-\alpha)\, y_{k-1}$$

$$y_k = \alpha\, y_{m,k} + (1-\alpha)\left[\alpha\, y_{m,k-1} + (1-\alpha)\, y_{k-2}\right]$$

$$y_k = \alpha\, y_{m,k} + \alpha(1-\alpha)\, y_{m,k-1} + (1-\alpha)^2 y_{k-2}$$

$$y_k = \alpha\, y_{m,k} + \alpha(1-\alpha)\, y_{m,k-1} + (1-\alpha)^2\left[\alpha\, y_{m,k-2} + (1-\alpha)\, y_{k-3}\right]$$

$$y_k = \alpha\, y_{m,k} + \alpha(1-\alpha)\, y_{m,k-1} + \alpha(1-\alpha)^2 y_{m,k-2} + (1-\alpha)^3 y_{k-3}$$

We can see a pattern here, where $\underline{\underline{k_i = \alpha\cdot(1-\alpha)^i \text{ for } i \in [0,9]}}$

For the last node, α is replaced by $1-\alpha$ $\Rightarrow$ $\underline{\underline{k_{10} = (1-\alpha)^{10}}}$
(here at k-10)

The values (using $\alpha = \frac{1}{6}$):

k0	0,1667
k1	0,1389
k2	0,1157
k3	0,0965
k4	0,0804
k5	0,0670
k6	0,0558
k7	0,0465
k8	0,0388
k9	0,0323
k10	0,1615
Sum	1

Comment: A common *moving average* filter would simply use the average of the 10 previous inputs, that is,

$$y_k = 0.1y_k^m + 0.1y_{k-1}^m + \ldots + 0.1y_{k-9}^m. \tag{2}$$

Note that this moving average filter has two disadvantages compared to the first-order ("exponentially moving average") filter:

1. It requires that we store many old measurements (9 in this case), whereas the first-order filter only needs to store the previous filtered output, y_{k-1};
2. Its performance is not as good as that of the first-order filter.