Course EE5104: Adaptive Control System (Part II)

Adaptive Control Project

Sliding Mode Control (CA2)

Lecturer: **Assoc. Prof. Ho Weng Khuen**

Author: XU YIMIAN
Student ID: A0295779Y

National University of Singapore

Apr 10, 2025

Contents

List of Tables

List of Figures

1 Question

Consider the system

$$\dot{x}_1 = ax_1 + bu + d$$
$$\dot{x}_2 = x_1$$

The switching surface is defined by

$$\sigma = c_1 x_1 + c_2 x_2$$

where a, b, c_1, and c_2 are known *a priori*, and d is a bounded disturbance with $|d| \leq d_{\max}$.

(1) Design a variable structure controller such that $x = 0$ is an asymptotically stable solution.

(2) Simulate the dynamic behaviour of the system assuming that:

$$a = 2, \quad b = 1, \quad c_1 = c_2 = 1,$$
$$d = 0.9\sin(628t), \quad d_{\max} = 0.9, \quad \mu = 0.5$$

Consider both the `sign()` and `sat()` functions. For the `sat()` function, let $\varepsilon = 0.01$. Give the phase portrait, plots of states and control signal versus time, and discuss the results.

2 Solution

I adopt a sliding mode control framework due to its robustness against bounded disturbances. As the sliding surface is defined as $\sigma = c_1 * x_1 + c_2 * x_2$, once the system trajectory reaches this surface, it should remain on it and converge to the origin.

To ensure asymptotic stability, we consider the Lyapunov function: $V = \frac{1}{2}\sigma^2$. Taking the time derivative: $\dot{V} = \sigma\dot{\sigma}$. We aim to enforce: $\dot{V} = -\mu|\sigma| \quad \Rightarrow \quad \dot{\sigma} = -\mu \cdot \mathrm{sign}(\sigma)$.

$$\sigma = c_1 x_1 + c_2 x_2 \Rightarrow \dot{\sigma} = (ac_1 + c_2)x_1 + bc_1 u + c_1 d$$

To achieve this, I choose the control law as:

$$u = -\frac{1}{bc_1}\left((ac_1 + c_2)x_1 + \mu \cdot \mathrm{sign}(\sigma)\right)$$

Although the sign-based sliding mode control ensures fast converge in theory, it often induces chattering in practice due to the discontinuous nature of the control law. The control input switches rapidly near the sliding surface, causing high-frequency oscillations that may excite unmodeled dynamics or damage physical actuators. So we introduce the saturation function to mitigate chattering:

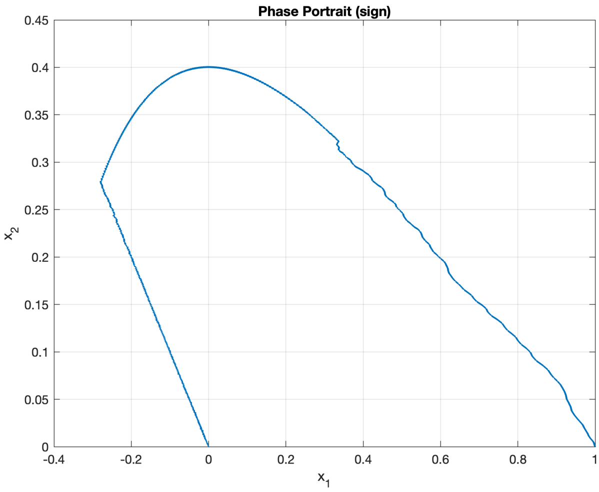
$$\mathrm{sat}\left(\frac{\sigma}{\varepsilon}\right) = \begin{cases} 1 & \sigma > \varepsilon \\ \frac{\sigma}{\varepsilon} & |\sigma| \leq \varepsilon \\ -1 & \sigma < -\varepsilon \end{cases}$$

Here, ε represents the boundary layer thickness. Choosing an appropriate ε balances the trade-off between reducing chattering and maintaining control accuracy. The control law becomes as following:

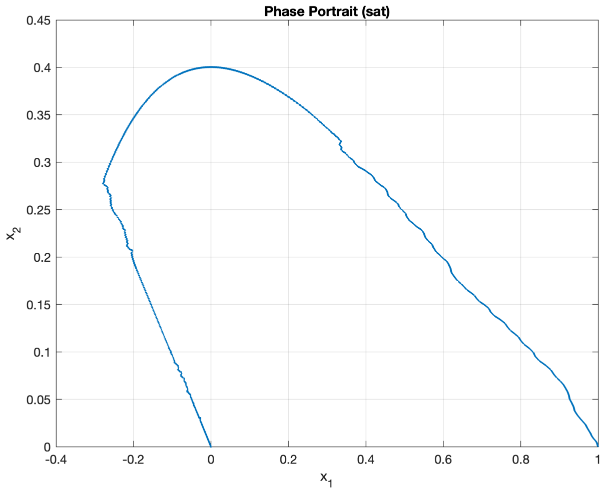
$$u = -\frac{1}{bc_1}\left((ac_1 + c_2)x_1 + \mu \cdot \mathrm{sat}(\frac{\sigma}{\varepsilon})\right)$$

I simulate using MATLAB's `ode45` (code 1), comparing results between the `sign` and `sat` controller (shown in Figure 1).

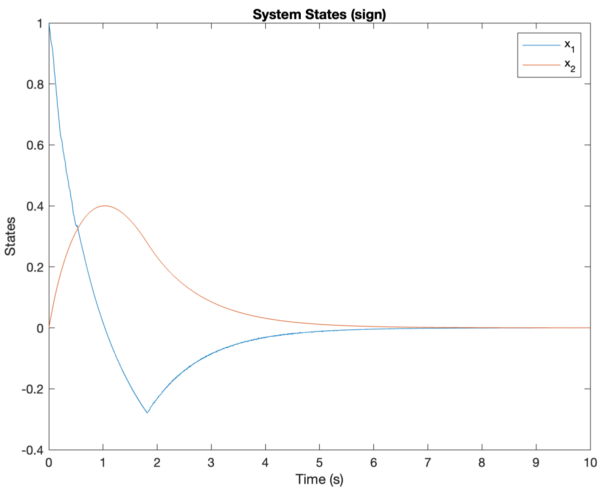
3 Results and Discussion



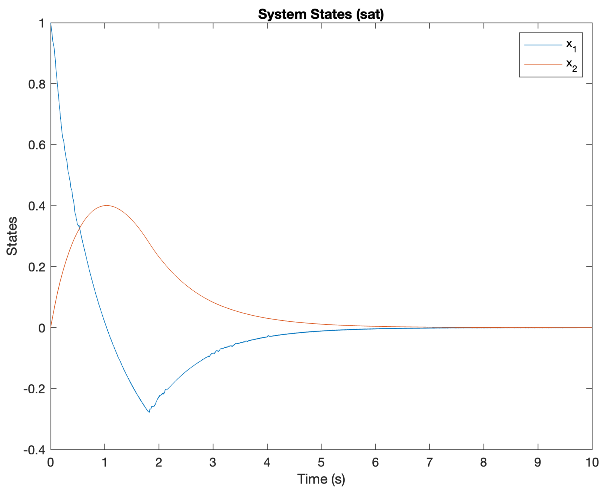
(a) Phase Portrait (sign)



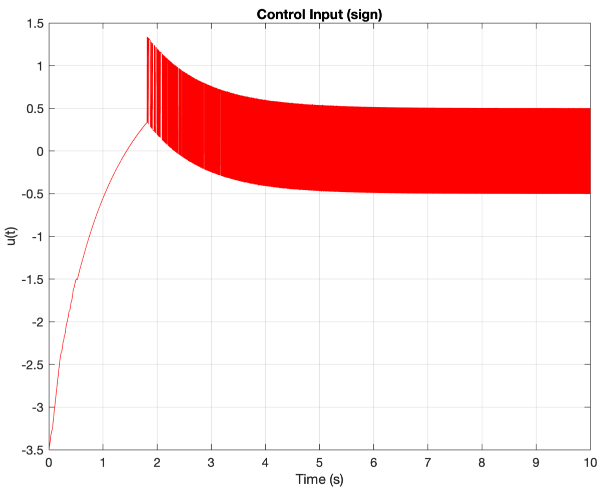
(b) Phase Portrait (sat)



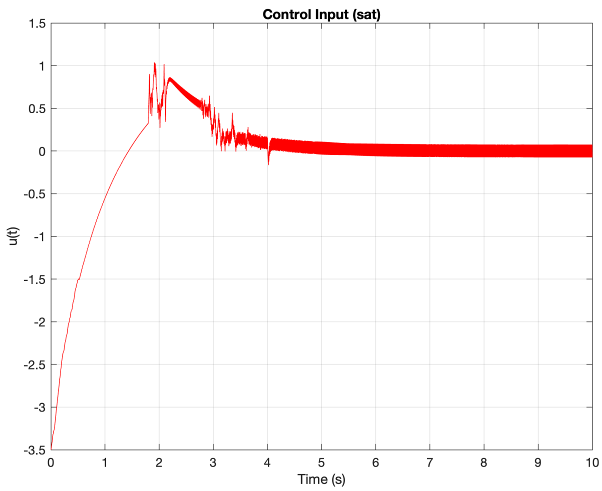
(c) State vs. Time (sign)



(d) State vs. Time (sat)



(e) Control Signal (sign)



(f) Control Signal (sat)

Figure 1: Simulation Results

3.1 Phase Portrait

The trajectories in the phase plane clearly show convergence to the origin under both control laws. The `sign()` function ensures finite-time convergence to the sliding surface while the `sat()` functions smooths the control action near the sliding surface, effectively mitigating chattering while preserving the robustness of SMC.

3.2 State vs. Time

Under both controllers, x_1 and x_2 converge to 0. The `sat()` function causes smoother state evolution due to reduced switching.

3.3 Control Signal vs. Time

The `sign()` controller produces high-frequency switching in the control signal. The `sat()` controller yields a bounded, smoother signal within the boundary layer, making it more feasible for real-world actuators.

3.4 Comparative Analysis of `sign()` and `sat()` Controllers

It was observed that both controllers reached the sliding surface at approximately the same time, indicating that their reaching phases are comparable under the given conditions. However, the `sat()` controller yielded a significantly smoother control signal and state trajectory, with almost no observable chattering.

This demonstrates that while both control laws can effectively enforce sliding motion, the saturation-based controller offers superior performance in practical implementations where high-frequency switching is undesirable.

3.5 Remarks on Reaching Gain and Robustness

In theory, to ensure sliding motion under a bounded disturbance $|d(t)| \leq d_{\max}$, the reaching gain μ must satisfy: $\mu > c_1 d_{\max}$. In our case, $d_{\max} = 0.9$, $c_1 = 1$, and $\mu = 0.5$, so the robustness condition is not satisfied. As a result:

- The system may fail to reach or maintain the sliding surface under the worst-case disturbance.
- Chattering or deviation from the desired surface may occur.
- With a saturation-based controller, the system can still stay within a boundary layer, but exact convergence is not guaranteed (shown in Figure 2).

Variables – x

27117x2 double

	1	2
27106	-2.2566e-05	7.6506e-05
27107	-1.7208e-05	7.6505e-05
27108	-9.7679e-06	7.6504e-05
27109	-3.2713e-07	7.6504e-05
27110	7.3812e-05	7.6515e-05
27111	1.8356e-04	7.6559e-05
27112	3.1533e-04	7.6646e-05
27113	4.5448e-04	7.6781e-05
27114	6.5623e-04	7.7097e-05
27115	7.7731e-04	7.7505e-05
27116	7.6866e-04	7.7946e-05
27117	5.9731e-04	7.8339e-05
27118		

(a) System States (sign)

Variables – x

7577x2 double

	1	2
7566	-2.1569e-04	7.9038e-05
7567	-8.7549e-04	7.8600e-05
7568	-0.0013	7.7708e-05
7569	-0.0015	7.6556e-05
7570	-0.0013	7.5447e-05
7571	-8.7160e-04	7.4590e-05
7572	-2.3395e-04	7.4161e-05
7573	4.4368e-04	7.4242e-05
7574	8.2383e-04	7.4557e-05
7575	0.0011	7.5039e-05
7576	0.0013	7.5638e-05
7577	0.0014	7.6296e-05
7578		

(b) System States (sat)

Figure 2: State Convergence (sign vs. sat)

3.6 On Practical Behavior Under Insufficient Reaching Gain

Although the theoretical robustness condition requires $\mu > c_1 d_{\max}$ to guarantee convergence under bounded disturbances, my simulation shows that the system can still reach the sliding surface when $\mu < c_1 d_{\max}$.

This occurs because the disturbance $d(t)$ is a sinusoidal function with zero mean and continuously varying magnitude. While its peak is 0.9, the instantaneous value is often much smaller, allowing the system to reject most of the disturbance.

However, this convergence is not guaranteed in the worst case, and robustness is not ensured. A higher value of μ would theoretically and practically strengthen the controller's ability to reject disturbances.

3.7 Summary

The sliding mode controller shows strong robustness against bounded disturbances and achieves asymptotic stability.

4 Code

Listing 1: MATLAB script

```
% Parameters
a = 2; b = 1;
c1 = 1; c2 = 1;
dmax = 0.9;
mu = 0.5;
epsilon = 0.01;
```

```
tspan = [0, 10];  % time span
x0 = [1.0; 0];  % initial state: [x1, x2]

% Control mode: 'sign' or 'sat'
for mode = 1:2
    clearvars t x u
    switch mode
        case 1
            control_mode = 'sign';
        case 2
            control_mode = 'sat';
        otherwise
            fprintf('Error: Undefined mode!');
    end

    [t, x] = ode45(@(t, x) dynamics(t, x, a, b, c1, c2, mu, dmax, control_mode, epsilon), tspan, x0);

    % Calculate control signal
    for i = 1:length(t)
        x1 = x(i,1);
        x2 = x(i,2);
        sigma = c1*x1 + c2*x2;
        if strcmp(control_mode, 'sign')
            u(i) = -(1/(b * c1))*((a*c1 + c2)*x1 + mu * sign(sigma));
        else
            u(i) = -(1/(b * c1))*((a*c1 + c2)*x1 + mu * sat(sigma/epsilon));
        end
    end

    % --------- Plot Results ---------
    save_path='./image/';
    f=figure;
    plot(x(:,1), x(:,2), 'LineWidth', 1.5);
    xlabel('x_1'); ylabel('x_2'); title(sprintf('Phase Portrait (%s)',control_mode)); grid on;
    exportgraphics(f, [save_path, control_mode, '_PhasePortrait.png'], ...
    'ContentType', 'image', ...
    'Resolution', 300);

    f=figure;
    plot(t, x); legend('x_1','x_2');
    xlabel('Time (s)'); ylabel('States'); title(sprintf('System States (%s)',control_mode));
    exportgraphics(f, [save_path, control_mode, '_SystemStates.png'], ...
    'ContentType', 'image', ...
    'Resolution', 300);

    f=figure;
    plot(t, u, 'r'); xlabel('Time (s)'); ylabel('u(t)');
    title(sprintf('Control Input (%s)',control_mode)); grid on;
    exportgraphics(f, [save_path, control_mode, '_ControlInput.png'], ...
    'ContentType', 'image', ...
    'Resolution', 300);
end
%==================================================================
function dx = dynamics(t, x, a, b, c1, c2, mu, dmax, mode, epsilon)
    x1 = x(1);
    x2 = x(2);
    sigma = c1*x1 + c2*x2;

    % Disturbance
    d = dmax * sin(628 * t);  % bounded, |d| <= 0.9

    % Control Law
    if strcmp(mode, 'sign')
        u = -(1/(b * c1))*((a*c1 + c2)*x1 + mu * sign(sigma));
    else
        u = -(1/(b * c1))*((a*c1 + c2)*x1 + mu * sat(sigma/epsilon));
    end
```

```

    dx1 = a*x1 + b*u + d;
    dx2 = x1;
    dx = [dx1; dx2];
end
%==================================================================
function y = sat(x)
    y = max(-1, min(1, x));
end
```