System and Control Theory Home Project

Nahom Welday Weldemikael
Neptun Code: FPXWWD

Wednesday, May 15, 2024

1 Introduction

This project involves the simulation and control of a nonlinear dynamic system described by differential equations. The project is divided into two tasks:

1. Simulation of the system without control inputs and the required plots.
2. Design and implementation of a Variable Structure/Sliding Mode (VS/SM) control strategy to track given nominal trajectories and the required plots.

The implementation is done using Julia programming language and PyPlot for plotting, executed within a Jupyter Notebook environment.

2 Setting Up the Environment

To run the simulations and control implementations, you need to set up your environment with Julia and Jupyter Notebook. Here are the steps to do so:

2.1 Installing Julia

1. Download Julia and install Julia : In this part with the picture provided shows, that the downloaded and installed Julia in my computer.

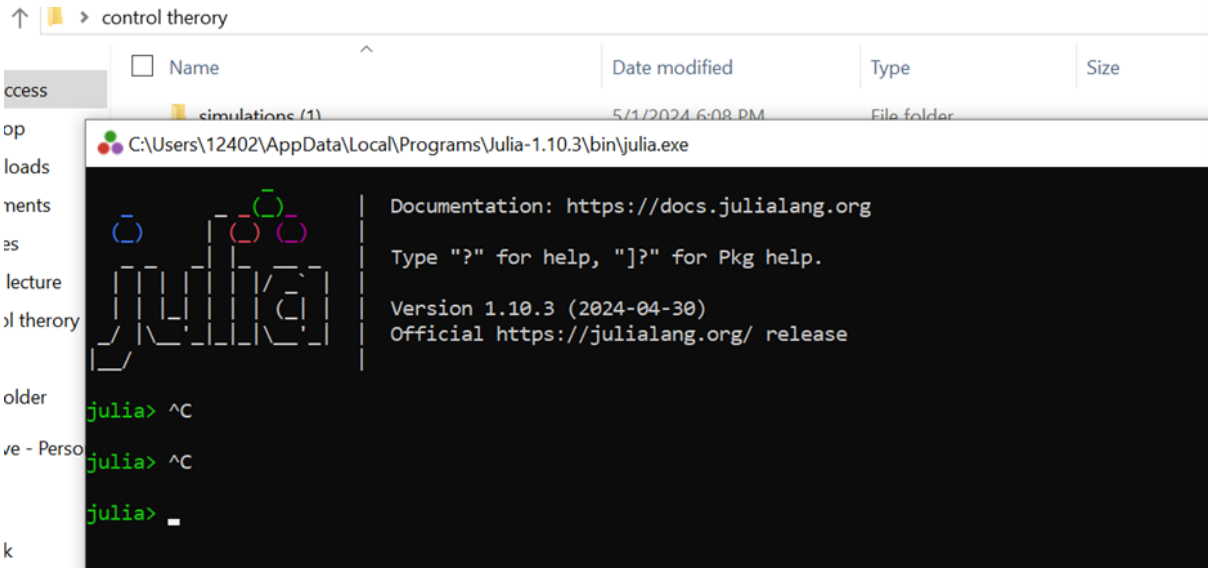


Figure 1: Installed and running Julia software

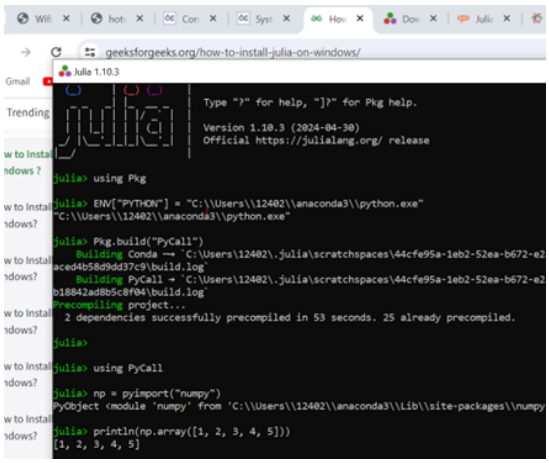
2.2 Installing Jupyter Notebook

1. Install Julia Packages : Open Julia and run the following commands to install the required packages:
2. Install Jupyter: The IJulia package will install Jupyter . You can start Jupyter Notebook by running.
We do the same for using Pkg and Pkg.add("Pyplot").

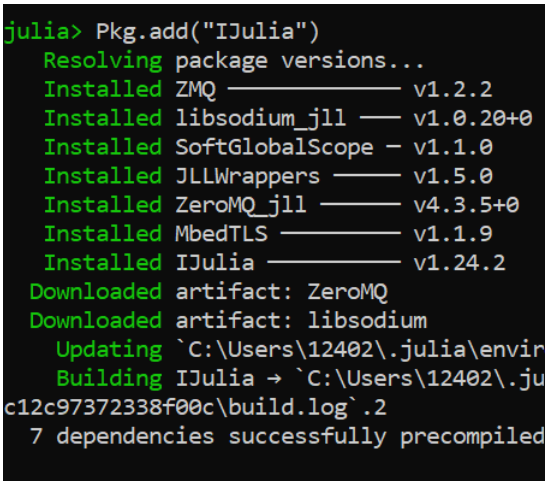
```
using Pkg
Pkg.add("IJulia")

then

using IJulia
notebook()
```



(a) solving Pkg conflicts



(b) IJulia Installation in my Laptop

Figure 2: Julia configuration and installation

3. Verify Installation: Ensure that Jupyter opens in my browser and that you can create a new Julia notebook.

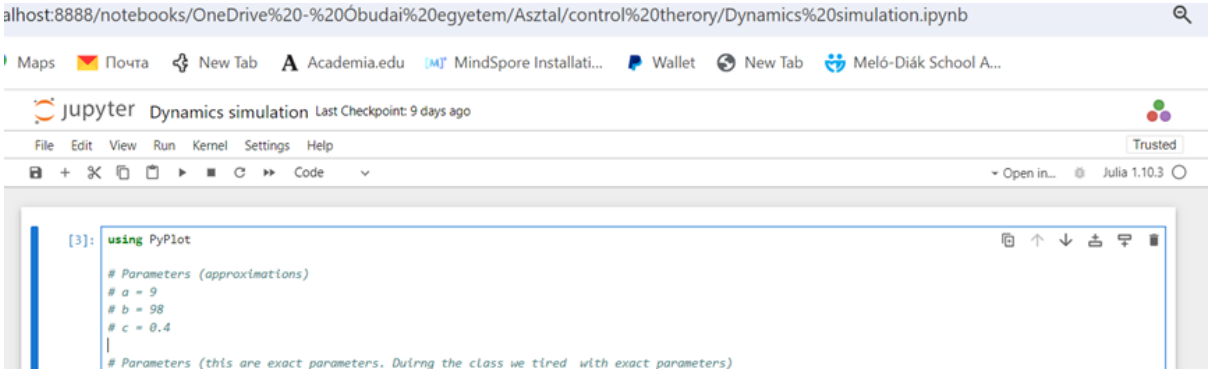


Figure 3: Jupyter notebook IDE for Julia

3 Task 1: Simulation Without Control Inputs

3.1 Problem Setup

The system is described by the following differential equations without control inputs: This simulation involves solving a system of differential equations using Euler's method to observe the behavior of three

variables x, y, and z over time. The system's dynamics are governed by the following differential equations:

$$\begin{aligned}\dot{x} &= ay \\ \dot{y} &= -cx + yz \\ \dot{z} &= b - y^2\end{aligned}$$

where the parameters are given as (using the Exact parameters, as we practised using the exact parameters in the class lessons):

- $a = 10$
- $b = 100$
- $c = 0.3$

The initial conditions for the simulation are:

- $x_0 = 0.1$
- $y_0 = 0.1$
- $z_0 = 0.1$

Simulation Settings:
The simulation is performed using Euler's method with a time step $h = 1 \times 10^{-3}$ over a length of 2×10^4 steps.

$$\begin{aligned}\text{Simulation length:} \quad & L_{\text{Long}} = 20000 \\ \text{Time step:} \quad & h = 0.001 \\ \text{Initial conditions:} \quad & x[1] = 0.1, \quad y[1] = 0.1, \quad z[1] = 0.1\end{aligned}$$

3.2 Design of the Control Problem

3.2.1 Problem Statement

The system is designed to simulate the interaction between the variables x, y, and z over time using a set of differential equations. These equations represent a simplified model of some dynamic system where:

- x is driven by y,
- y is influenced by both x and z,
- z is modified by y.

3.2.2 Mathematical Formulation

Here we can see, The system can be mathematically represented as:

$$\begin{aligned}\dot{x} &= a \cdot y \\ \dot{y} &= -c \cdot x + y \cdot z \\ \dot{z} &= b - y^2\end{aligned}$$

Where:

- $\dot{x}$, $\dot{y}$, and $\dot{z}$ are the time derivatives of x, y, and z, respectively.
- The constants a, b, and c are parameters that influence the system's behavior.

3.3 Implementation

Using Euler's method, the differential equations are discredited and iteratively solved over the specified time period. The initial values of values of x, y, and z are set, and the equations are used to compute the values at each subsequent time step.

3.4 Code Implementation

This is Julia cdoe and with explanation of the code.

3.4.1 Parameters

The parameters for the system are defined as follows:

```
# This  are the exact parameters, what we used in class for demonstration.
a = 10
b = 100
c = 0.3

# Parameters (approximations)
# a = 9
# b = 98
# c = 0.4
```

These parameters are used in the differential equations to determine the evolution of the system over time.

3.4.2 Simulation Settings

The simulation length and time step are set as:

```
LONG = Int(2e4)  # Simulation length
l = LONG - 1
h = 1e-3  # Time step
```

- `LONG`: Total number of time steps for the simulation.
- `l`: Number of iterations (one less than `LONG` since indexing starts from 1).
- `h`: Time step for the Euler integration.

3.4.3 Initial Conditions

The state variables x, y, and z are initialized as arrays of zeros and set with initial conditions:

```
x = zeros(LONG)
y = zeros(LONG)
z = zeros(LONG)

x[1] = 0.1
y[1] = 0.1
z[1] = 0.1
```

- x, y, z: Arrays to store the values of the state variables at each time step.
- Initial values for `x[1]`, `y[1]`, and `z[1]` are set to 0.1.

3.4.4 Euler's Method

The system is simulated using Euler's method to approximate the solutions of the differential equations:

```
for i in 1:l
    ẋ = a*y[i]  # Modified differential equation
    ẏ = -c*x[i] + y[i]*z[i]  # Modified differential equation
    ż = b - y[i]^2  # Modified differential equation

    x[i+1] = x[i] + ẋ*h
    y[i+1] = y[i] + ẏ*h
    z[i+1] = z[i] + ż*h
end
```

- For each time step i, the derivatives $\dot{x}$, $\dot{y}$, and $\dot{z}$ are calculated using the current values of $x[i]$, $y[i]$, and $z[i]$.
- The state variables x, y, and z are updated for the next time step using the Euler method formula: $x[i+1] = x[i] + \dot{x} \cdot h$.

3.4.5 Plotting the Results

The results are plotted using PyPlot:

```
using PyPlot

figure(figsize=(10, 8))

subplot(2, 2, 1)
plot3D(x[1:l], y[1:l], z[1:l])
title("3D Plot")

subplot(2, 2, 2)
plot(x[1:l])
title("Time plot for x")

subplot(2, 2, 3)
plot(y[1:l])
title("Time plot for y")

subplot(2, 2, 4)
plot(z[1:l])
title("Time plot for z")

tight_layout()
show()
```

- Subplots are created:
  - The first subplot is a 3D plot of x, y, and z over time.
  - The second subplot is a time series plot of x.
  - The third subplot is a time series plot of y.
  - The fourth subplot is a time series plot of z.
- `show()` displays the plots.

3.5 Result and Explanation

The plots illustrate the behavior of a dynamical system represented by the differential equations specified. The simulation was carried out using Euler's method to observe how the variables x, y, and z evolve over time. The parameters used were both exact and approximate, to study any potential discrepancies

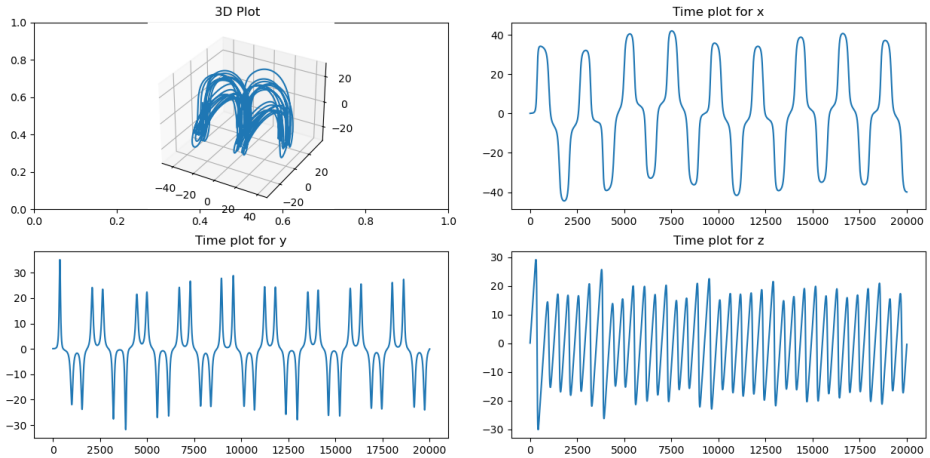


Figure 4: 3D Plot and time plots in X, Y, Z

3.5.1 Explanation of the Results

- **3D Plot with out control input**: The 3D plot shows the trajectory of the system in the three-dimensional space defined by (x, y, z).

  - **Observation:** The trajectory in the 3D plot appears to have a repeating pattern, indicating a form of periodic. This suggests that the system does not diverge to infinity or converge to a fixed point, but rather follows a complex yet bounded path in its state space.

- **Time Plot for X** : This plot shows the value of x over time.
  Observation: The plot for x indicates a periodic behavior with distinct peaks and valleys. The oscillations are regular, suggesting that x exhibits a steady periodic pattern over time.

- **Time Plot for Y** : This plot shows the value of y over time.
  Observation: Similar to x, the variable y also shows periodic behavior. However, the peaks and valleys in y are more noticeable, indicating sharper changes in its value over time. This might be due to its direct interaction with both x and z in the differential equations.

- **Time Plot for Z** : This plot shows the value of z over time.

  observation:The variable z exhibits a higher frequency of oscillations compared to x and y. The plot shows more rapid changes, suggesting that z is more sensitive to the dynamics of the system. The amplitude of the oscillations also appears to decrease slightly over time.

- **Overall result**: The 3D trajectory and time plots show that the system's variables **oscillate periodically**, indicating limit cycle behavior with repetitive cycles. The **periodic behavior** of x ,y, and z demonstrates their **interdependence**, with each variable influencing the others according to the differential equations. The 3D trajectory **shape** and the **varying amplitudes** and **frequencies** in the time plots suggest **complex, nonlinear** interactions.

3.5.2 Using both exact and approximate parameters, the system consistently shows periodic nature

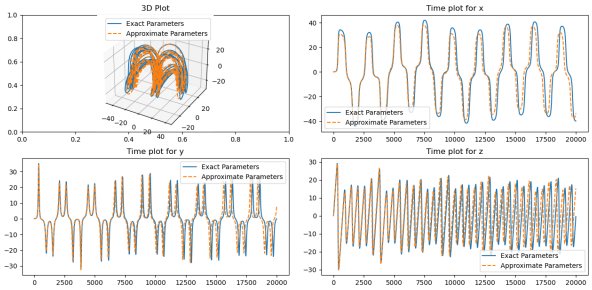


Figure 5: 3D Plot and time plots exact vs approximate variables

4 Task 2: VS/SM Control Implementation

4.1 Problem Setup

The system dynamics are described by the differential equations provided earlier:

- **Mathematical Formulation** Where: - σ, β, ρ are the system parameters. - u_x, u_y, u_z are the control inputs.

$$\dot{x} = \sigma y + u_x$$
$$\dot{y} = -\beta x + yz + u_y$$
$$\dot{z} = \rho - y^2 + u_z$$

Given that the initial conditions for x, y, and z x(0) = 0, y(0) = 0, z(0) =0
The system's behavior is governed by differential equations representing the dynamics of state variables x, y, and z. The goal is to control these state variables to follow a desired trajectory using SMC.

$$q_N = A\sin(\omega t)$$

with parameters:

- $A_1 = 2, \omega_1 = 0.5$
- $A_2 = 3.8, \omega_2 = 0.7$
- $A_3 = 1.5, \omega_3 = 0.9$

4.2 Sliding Mode of Control Design(error Metric(S))

The control strategy involves using a sliding mode controller with a PID-type adjustment. The sliding surfaces are defined as:

$$\begin{aligned} S_x &= e_x + \Lambda e_{\text{int},x} + K e_{\dot{x}} \\ S_y &= e_y + \Lambda e_{\text{int},y} + K e_{\dot{y}} \\ S_z &= e_z + \Lambda e_{\text{int},z} + K e_{\dot{z}} \end{aligned}$$

where Λ and K are control gains, and the errors e_x, e_y, e_z are the differences between the nominal and realized trajectories.

4.3 Kinematic Block(with PID feed back)

The control inputs are then computed using these sliding surfaces:

$$\begin{aligned} \dot{x}_{\text{desired}} &= \Lambda e_x + \dot{x}_{\text{nominal}} + K \tanh\left(\frac{S_x}{w}\right) \\ \dot{y}_{\text{desired}} &= \Lambda e_y + \dot{y}_{\text{nominal}} + K \tanh\left(\frac{S_y}{w}\right) \\ \dot{z}_{\text{desired}} &= \Lambda e_z + \dot{z}_{\text{nominal}} + K \tanh\left(\frac{S_z}{w}\right) \end{aligned}$$

The control inputs are then used to update the system state.

4.4 Control Inputs Calculation(Inverse Model)

The control inputs are then computed using these sliding surfaces:

$$\begin{aligned} u_x &= \dot{x}_{\text{des}} - \sigma y \\ u_y &= \dot{y}_{\text{des}} + \beta x - yz \\ u_z &= \dot{z}_{\text{des}} - \rho + y^2 \end{aligned}$$

The control inputs are then used to update the system state.

4.5 The Exact model of the System

This gives the system reaction.That is Updates system states using the system model

$$\begin{aligned} \dot{x} &= \sigma_e y + u_x \\ \dot{y} &= -\beta_e x + yz + u_y \\ \dot{z} &= \rho_e - y^2 + u_z \end{aligned}$$

4.6 System Dynamics Update

The system states are updated using the Euler integration method

$$x[i+1] = x[i] + \delta t\, \dot{x}[i]$$
$$y[i+1] = y[i] + \delta t\, \dot{y}[i]$$
$$z[i+1] = z[i] + \delta t\, \dot{z}[i]$$

4.7 Implementation

This simulation uses sliding mode control with PID adjustments to manage a dynamic system. It includes comparing desired and actual trajectories to evaluate tracking performance, measuring tracking errors to assess control accuracy, visualizing phase space trajectories to analyze stability and response, and examining control signals to understand the control effort needed for trajectory tracking.

4.7.1 Code Implementation with explanation

4.7.2 Control and System Parameters

I found control **parameters** after **extensive experimentation** λ and k and , where I systematically adjusted the parameters and analyzed the resulting graphs. The chosen parameters ensure that the realized trajectories closely match the nominal trajectories, indicating minimal tracking errors. Thus, the parameters for the system are defined as follows:

```
using PyPlot

# Define control parameters
Λ = 900 # Proportional gain (P)
K = 1/300 #Gain for the corrective action, similar to Derivative gain (D)
w = 1   # Boundary width for the tanh function, smoothing control actions

# Define time parameters
δt = 1e-3
LONG = Int(2e4)
l = LONG - 1

# System dynamics parameters
σ_e = 10
ϱ_e = 100
β_e = 0.3

# Inverse dynamics parameters for control calculation
σ_a = 9
β_a = 98
ϱ_a = 0.4

# Nominal trajectory parameters
A1 = 2
ω1 = 0.5
A2 = 3.8
ω2 = 0.7
A3 = 1.5
ω3 = 0.9
```

Control Parameters: Define the parameters for the sliding mode control. Time Parameters: Set the time step δt and total number of simulation steps (LONG). System Model Parameters: Define the constants for the system dynamics. Inverse Model Parameters: Define the constants for the inverse model used in control. Nominal Trajectory Parameters: Define the amplitudes and frequencies for the desired trajectories.

4.7.3 Initialization of Arrays and Variables

The parameters for the system are defined as follows:

```
# Initialization of state and control arrays
x = zeros(LONG)
y = zeros(LONG)
z = zeros(LONG)
ẋ = zeros(LONG)
ẏ = zeros(LONG)
```

```
ż = zeros(LONG)
ẋ^Des = zeros(LONG)
ẏ^Des = zeros(LONG)
ż^Des = zeros(LONG)
x^N = zeros(LONG)
y^N = zeros(LONG)
z^N = zeros(LONG)
ẋ^N = zeros(LONG)
ẏ^N = zeros(LONG)
ż^N = zeros(LONG)
u_x = zeros(LONG)
u_y = zeros(LONG)
u_z = zeros(LONG)
S_x = zeros(LONG)
S_x_p = zeros(LONG)
S_y = zeros(LONG)
S_y_p = zeros(LONG)
S_z = zeros(LONG)
S_z_p = zeros(LONG)
t = zeros(LONG)
e_int_x = 0
e_int_y = 0
e_int_z = 0
e_prev_x = 0
e_prev_y = 0
e_prev_z = 0
```

State Variables: Arrays to store the state variables (x, y, z) and their derivatives $(\dot{x}, \dot{y}, \dot{z})$. **Desired Trajectories:** Arrays to store the desired state derivatives $(\dot{x}^{\text{des}}, \dot{y}^{\text{des}}, \dot{z}^{\text{des}})$ and nominal trajectories $(x^{\text{N}}, y^{\text{N}}, z^{\text{N}})$.

Control Inputs: Arrays to store control inputs (u_x, u_y, u_z). **Sliding Mode Control:** Arrays to store sliding variables (S_x, S_y, S_z) and their previous values. **Time Array:** Array to store time steps. **Error Integrals:** Variables to store the integrated error for each state. **Error Derivatives:** to store derivated error for each state

4.7.4 Initial Conditions

Initial State Values: Set initial values for the state variables and their derivatives to zero according to the problem instruction.

```
x[1]=0
y[1]=0
z[1]=0

ẋ[1]=0
ẏ[1]=0
ż[1]=0
```

4.7.5 Simulation loop

- Time Update: Update the current time step.
- Nominal Trajectories: Compute the nominal trajectories and their derivatives at the current time step.
- Error Calculation: Calculate the tracking errors for each state variable.
- Error Integration: Integrate the tracking errors.
- Sliding Mode Control: Compute the sliding variables based on integrated, derivative and current errors. (PID error metric)
- Desired Derivatives: Compute the desired state derivatives using the sliding mode control law.
- Inverse Dynamics Model: Calculate the control inputs required to achieve the desired derivatives.
- State Update: Update the state derivatives using the system model equations.
- State Integration: Integrate the state derivatives to obtain the new state values.

```
for i = 1:l
    t[i] = i * δt  # Time update
    # Calculate nominal trajectories
    x^N[i] = A1 * sin(ω1 * t[i])
    y^N[i] = A2 * sin(ω2 * t[i])
    z^N[i] = A3 * sin(ω3 * t[i])
    ẋ^N[i] = A1 * ω1 * cos(ω1 * t[i])
    ẏ^N[i] = A2 * ω2 * cos(ω2 * t[i])
    ż^N[i] = A3 * ω3 * cos(ω3 * t[i])

    # Calculate current errors
    e_x = x^N[i] - x[i]
    e_y = y^N[i] - y[i]
    e_z = z^N[i] - z[i]

    # Integrate error for the Integral component
    e_int_x += δt * e_x
    e_int_y += δt * e_y
    e_int_z += δt * e_z

    # Derivative of error for the Derivative component
    e_dot_x = (e_x - e_prev_x) / δt
    e_dot_y = (e_y - e_prev_y) / δt
    e_dot_z = (e_z - e_prev_z) / δt
    e_prev_x = e_x
    e_prev_y = e_y
    e_prev_z = e_z

    # Sliding mode control computation with PID error metrics
    S_x[i] =  e_x + Λ * e_int_x + K * e_dot_x  # Adding PID components
    S_y[i] =  e_y + Λ * e_int_y + K * e_dot_y
    S_z[i] =  e_z + Λ * e_int_z + K * e_dot_z



    # Compute control inputs using sliding mode dynamics
    ẋ^Des[i] = Λ * e_x + ẋ^N[i] + K * tanh(S_x[i] / w)
    ẏ^Des[i] = Λ * e_y + ẏ^N[i] + K * tanh(S_y[i] / w)
    ż^Des[i] = Λ * e_z + ż^N[i] + K * tanh(S_z[i] / w)

   # Dynamics inverse model
    u_x[i] = ẋ^Des[i] - σ_a * y[i]
    u_y[i] = ẏ^Des[i] + β_a * x[i] - y[i] * z[i]
    u_z[i] = ż^Des[i] - ϱ_a + y[i]^2

 # Dynamics Update system states using the system model
    ẋ[i] = σ_e * y[i] + u_x[i]
    ẏ[i] = -β_e * x[i] + y[i] * z[i] + u_y[i]
    ż[i] = ϱ_e - y[i]^2 + u_z[i]

    x[i+1] = x[i] + δt * ẋ[i]
    y[i+1] = y[i] + δt * ẏ[i]
    z[i+1] = z[i] + δt * ż[i]
end
```

4.7.6 Plotting the Results

Plot the results: Nominal and Realized Trajectories, Tracking Errors, Phase Spaces, Control Signals using PyPlot

```
using PyPlot

# Plotting nominal and realized trajectories
figure(1)
grid(true)
title("Nominal and Realized Trajectory")
plot(t[1:l], x^N[1:l], label="Nominal x")
plot(t[1:l], y^N[1:l], label="Nominal y")
plot(t[1:l], z^N[1:l], label="Nominal z")
plot(t[1:l], x[1:l], "--", label="Realized x")
plot(t[1:l], y[1:l], "--", label="Realized y")
plot(t[1:l], z[1:l], "--", label="Realized z")
legend()

# Plotting tracking errors for x, y, z
figure(2)
grid(true)
title("Tracking Error")
plot(t[1:l], x^N[1:l] - x[1:l], label="Error x")
plot(t[1:l], y^N[1:l] - y[1:l], label="Error y")
```

```
plot(t[1:l], z^N[1:l] - z[1:l], label="Error z")
legend()

# Plotting phase space trajectories
figure(3,figsize=(34, 10))
grid(true)
title("Phase Space")
plot(ẋ^N[1:l], x^N[1:l], label="Nominal Phase x")
plot(ẏ^N[1:l], y^N[1:l], label="Nominal Phase y")
plot(ż^N[1:l], z^N[1:l], label="Nominal Phase z")
plot(ẋ[1:l], x[1:l], "--", label="Realized Phase x")
plot(ẏ[1:l], y[1:l], "--", label="Realized Phase y")
plot(ż[1:l], z[1:l], "--", label="Realized Phase z")
legend()

# Plotting control signals
figure(4)
grid(true)
title("Control Signals")
plot(t[1:l], u_x[1:l], label="Control u_x")
plot(t[1:l], u_y[1:l], label="Control u_y")
plot(t[1:l], u_z[1:l], label="Control u_z")
legend()

show()
```

- Figure 1: Plot nominal and realized trajectories of x, y, and z to compare the desired and actual paths.
- Figure 2: Plot the tracking errors for x, y, and z to visualize how well the system tracks the desired trajectories.
- Figure 3: Plot the phase space trajectories for x, y, and z to analyze the system dynamics in state space.
- Figure 4: Plot the control signals (u_x, u_y, u_z) to understand the control effort required to track the desired trajectories.

4.8 Results and Explanation

These plots collectively demonstrate that the control system effectively tracks the nominal trajectories with minimal errors. The control signals are successfully driving the system, and the phase space trajectories indicate consistent dynamic behavior. While the tracking errors are generally small, the largest deviations are observed in the y dimension

4.8.1 Nominal and Realized Trajectory

This plot compares the nominal (planned) trajectories with the realized (actual) trajectories for the system.

- **Nominal Trajectories (solid lines)**: Represent the planned paths for x, y, and z.
- **Realized Trajectories (dashed lines)**: Depict the actual paths taken by the system.

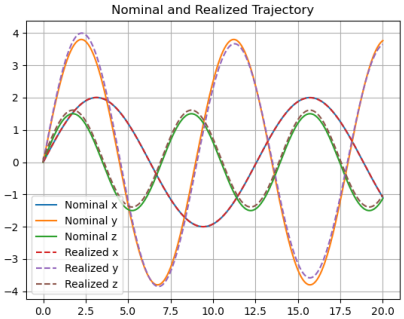


Figure 6: Nominal and Realized trajectory

Observation from the plot as we can see the close alignment of the solid and dashed lines indicates effective tracking performance, with the realized trajectory closely following the nominal one.

4.8.2 Control Signals

This plot shows the control inputs u_x, u_y, and u_z over time.

- **Control u_x (blue)**: Exhibits a sinusoidal pattern with a higher amplitude, indicating a significant role in driving the system.
- **Control u_y (orange)**: Displays smaller amplitude oscillations, suggesting a more stable component with less influence.
- **Control u_z (green)**: Starts with the largest negative amplitude and then oscillates distinctly from u_x and u_y, potentially serving a compensatory or stabilizing function.

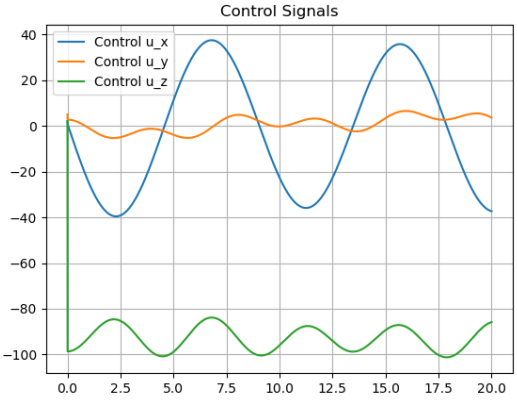


Figure 7: Control Signal

4.8.3 Phase Space

The phase space plot shows the system states' trajectories in phase space (state vs. state derivatives).

- **Nominal Phase (solid lines)**: These lines represent the expected phase paths for x, y, and z.
- **Realized Phase (dashed lines)**: These lines illustrate the actual paths taken by the system in the phase space.

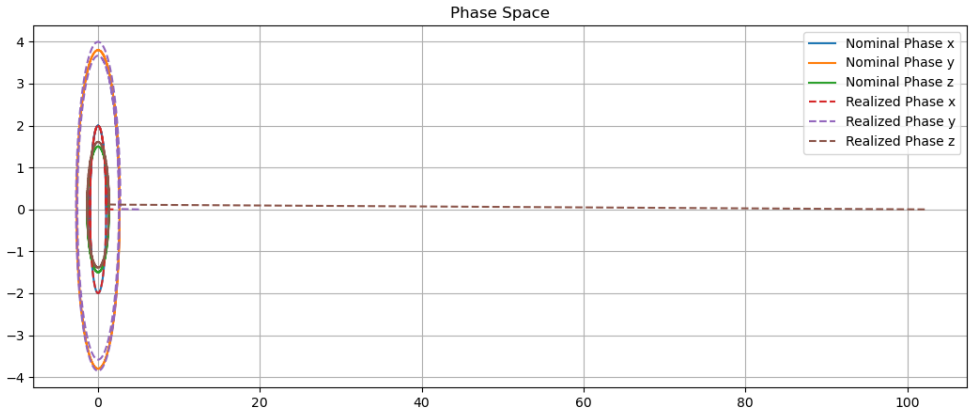


Figure 8: Phase space

Plot observation The close correspondence between the realized and nominal phases suggests that the system's dynamic behavior and state transitions are as expected. We also can see that, realized z starts from other values in the x- direction**(long horizontal dashed line)**, it should have started from origin,possibly **it could be due to initial values**, I remember in the **class demonstration** we had such kind of situation for Lorenz and it was fixed by trying different initial conditions. Thus, I have tried with different initial conditions, but I can still see the issue.

4.8.4 Tracking Error

This plot displays the error between the nominal and realized trajectories over time for x, y, and z.

- **Error in x (blue)**: Shows small oscillations around zero, indicating minimal tracking errors.
- **Error in y (orange)**: Exhibits larger oscillations, indicating higher deviations in this dimension.
- **Error in z (green)**: Remains close to zero, suggesting negligible tracking errors in this dimension.

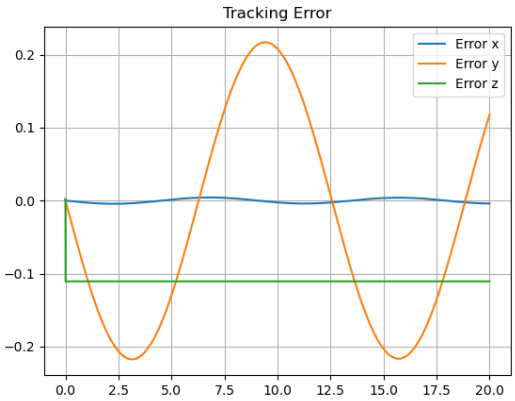


Figure 9: Julia code for VS/SM control implementation

5 Conclusion

The project successfully demonstrates the simulation of a nonlinear dynamic system and the implementation of a sliding mode control strategy with PID-type adjustments. The control strategy effectively tracks the nominal trajectories, as evidenced by the tracking errors and phase space plots. The control inputs are calculated and applied at each time step to maintain the desired system behavior.The figures provide a clear visualization of the system's performance both in the absence and presence of control inputs. The use of Julia and Jupyter Notebook provides an efficient and interactive environment for implementing and analyzing the system and control strategies.