Assignment 2: Simulation Basics

Optimal Control For Robotics

Assigned: Jan. 23 — Due: Feb. 1

Introduction

In this assignment you will implement Euler's method and use it to simulate two simple dynamical systems. The first system is an ideal passive pendulum, one of the simplest non-linear systems that corresponds to a physical mechanism. The second system is the Lorenz system, which is highly non-linear and generates the famous Lorenz Attractor.

This assignment includes a set of template files to guide you in writing the simulation function, dynamics functions, and top-level code to generate the plots.

- `EulerMethodSimulate.m` — simulate a dynamical system using Euler's method
- `PendulumDynamics.m` — compute the system dynamics for a simple pendulum
- `LorenzDynamics.m` — compute the system dynamics for the Lorenz system
- `prob_03_studentName.m` — run the simulation of the pendulum and make plots
- `prob_04_studentName.m` — run the simulation of the Lorenz system and make plots

You will need to complete each of these files by replacing the `%%%% TODO` markings with correct implementations in each case. There are detailed comments throughout the template code to indicate what each section of the code should do. The template files are described in detail at the end of this assignment, and included as stand-alone Matlab files as well.

Write-Up

You will submit a single (short) write-up for this assignment:

- Header: full name, studentName, date, assignment name and number
- List any other students that you worked with.
- How long did this assignment take you?

Deliverables

- Fully implement all five template files and submit them.
- Submit the plots that you generated for each simulation: `prob_03_studentName.pdf` and `prob_04_studentName.pdf`
- Submit the write-up for the assignment: `hw_02_studentName.txt`
- All code should be clearly written and well documented. Figures should include labels and titles.
- All files should besubmitted through Tufts Trunk (as a single compressed file)

EulerMethodSimulation.m

The key line that you will write in this file will implement Euler's Method for integration, given below, where $\boldsymbol{z}_k$ is the state at the current time step, $\boldsymbol{z}_{k+1}$ is the state at the next time step, h is the duration of the time step, and $\dot{\boldsymbol{z}}_k$ is the time-derivative of the state at the current time step. Notice that this function is general-purpose: it can be used to simulate any dynamical system.

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + h\dot{\boldsymbol{z}}_k \tag{1}$$

```
function [tGrid, zGrid] = EulerMethodSimulation(dynFun, tSpan, zInit, hMax)
% [tGrid, zGrid] = EulerMethodSimulation(dynFun, tSpan, zInit, hMax)
%
% Simulate a dynamical system over a set time span using Euler's method,
% given the initial state and the maximum time step. Use a uniform time grid.
%
% INPUTS:
%    dynFun = a function handle:  dz = dynFun(t, z)
%        IN:  t = [1, nTime] = row vector of time
%        IN:  z = [nState, nTime] = matrix of states corresponding to each time
%        OUT: dz = [nState, nTime] = time-derivative of the state at each point
%    tSpan = [startTime, finalTime] = [1, 2] = time span
%    zInit = [nState, 1] = state of the system at start time
%    hMax = scalar = maximum time step to use for the uniform time grid.
%
% OUTPUTS:
%    tGrid = [1, nTime] = time grid on which the integration was performed
%    zGrid = [nState, nTime] = state at each point in tGrid
%
%

%%%% TODO: Implement this function

end
```

PendulumDynamics.m

The dynamics of a simple pendulum can be modeled as:

$$\ddot{q} = -\sin(q) \tag{2}$$

Euler's Method for simulation, like most simulation methods, requires the dynamical system to be in first-order form. To to this, we will introduce a new variable $\omega = \dot{q}$ to represent angular rate. This allows us to write a second-order equation as a system of two first-order equations:

$$\dot{q} = \omega \tag{3}$$
$$\dot{\omega} = -\sin(q) \tag{4}$$

You will notice a tilde (`~`) in the argument list for this function. This tells Matlab that our function does not use this argument and allows it the compiler to do some optimization to improve run-time speed. We keep the argument here beause simulation functions (such as `EulerMethodSimulate()` and `ode45()`) often require the first argument of the dynamics function to be time.

```
function dz = PendulumDynamics(~, z)
% dz = PendulumDynamics(t, dz)
%
% This function computes the dynamics for a simple pendulum.
% All parameters are set to unity and there is no friction.
%
% INPUTS:
%   t = [1, nTime] = row vector of query times
%   z = [2, nTime] = [angle; rate] = current state of the system
%
% OUTPUTS:
%   dz = [2, nTime] = [rate, accel] = time-derivative at current state
%
% NOTES:
%   q = angle of the pendulum
%   w = angular rate of the pendulum (derivative of angle)
%   dw = angular acceleration of the pendulum (derivative of rate)
%   dynamics:
%       dw = -sin(q)
%

%%%% TODO: Implement this function

end
```

LorenzDynamics.m

This function implements the equations for the Lorenz system. For a special combination of parameters this system produces a famous fractal known as the Lorenz attractor. The equations for this system are given below, where x, y, and z are the state of the system, and $\sigma = 10$, $\rho = 29$, and $\beta = 8/3$ are system parameters.

$$\dot{x} = \sigma(y - x) \tag{5}$$
$$\dot{y} = x(\rho - z) - y \tag{6}$$
$$\dot{z} = xy - \beta z \tag{7}$$

```
function dState = LorenzDynamics(~, state)
%  dState = LorenzDynamics(time, state)
%
% This function computes the dynamics for the famous 'Lorenz Attractor'
% See the matlab function lorenz.m for a neat demo.
%
% INPUTS:
%   time = [1, nTime] = row vector of query times (unused)
%   state = [3, nTime] = current state of the system
%
% OUTPUTS:
%   dState = [3, nTime] = time-derivative at current state
%
% NOTES:
%   https://en.wikipedia.org/wiki/Lorenz_system
%
% dx = sigma * (y - x)
% dy = x * (rho - z) - y
% dz = x * y - beta * z
%

%%%% TODO: Implement this function

end
```

prob_03_studentName.m

This function sets up and runs a simulation of the simple pendulum system and then generates a plot of the resulting simulation. Let's assume that the pendulum is using SI units. Follow the directions in the template code for simulation and plotting details. When you submit this file you should replace `studentName` with your full name.

```
function prob_03_studentName()
%
% This function computes a simulation of a simple pendulum.
%

%~~~~~~~~~~~~~~~~~  Set up for the simulation  ~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% Duration of 20 seconds, with a maximum time-step of 0.01 seconds.
% Initial angle is randomly selected from [-3, 3] radians
% Initial angular rate is randomly selected from [-1, 1] radians / second
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%

%%%% TODO: Set up for the simulation

%~~~~~~~~~~~~~~~~~~~~~~~  Run the simulation  ~~~~~~~~~~~~~~~~~~~~~~~~~~~%
[t, z] = EulerMethodSimulation(dynFun, tSpan, zInit, hMax);

%~~~~~~~~~~~~~~~~~~  Make plots of the simulation  ~~~~~~~~~~~~~~~~~~~~~~%
% Create a single figure with three sub-plots (three rows, one column)
% The top sub-plot is pendulum angle vs time
% The middle sub-plot is angular rate vs time
% The botom sub-plot is angular acceleration vs time
% All axis should be clearly labeled (including units)
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%

% Set up the figure:
figure(1030); clf;

%%%% TODO: Make plots

end
```

prob_04_studentName.m

This function sets up and runs a simulation of the Lorenz System and then generates a plot of the Lorenz attractor. We will use dimensionless units. Follow the directions in the template code for simulation and plotting details.

```
function prob_04_studentName()
%
% This function computes a simulation of a the Lorenz system, creating
% a visualization of the Lorenz attractor fractal.
%

%~~~~~~~~~~~~~~~~~  Set up for the simulation  ~~~~~~~~~~~~~~~~~~~~~~~~~~%
% Duration of 10 seconds, with a maximum time-step of 0.005 seconds.
% x(0) is randomly selected from [5, 35]
% y(0) is randomly selected from [-30, 5]
% z(0) is randomly selected from [-5, 35]
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%

%%%% TODO:  set up for the simulation

%~~~~~~~~~~~~~~~~~~~~~~~~  Run the simulation  ~~~~~~~~~~~~~~~~~~~~~~~~~~%
[time, state] = EulerMethodSimulation(dynFun, tSpan, stateInit, hMax);


%~~~~~~~~~~~~~~~~  Plot the trajectory in state space ~~~~~~~~~~~~~~~~~~~%
% Use the plot3() command to plot (x(t) vs. y(t) vs. z(t)) on a single plot
% Set the axis commands so that the axis are equal in scale and not visible
% Add a title to the subplot
% The subplot() command places this plot on the right side of the figure.
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%

% Set up the figure
figure(1040); clf;
subplot(2, 3, [3,6]);

%%%% TODO:  plot the Lorenz attractor  (trajectory in state space)

%~~~~~~~~~~~~~~~~  Plot the state as a function of time ~~~~~~~~~~~~~~~~~~%
% The top sub-plot shows all three states as a function of time
% The bottom sub-plot shows the derivative of each state
% Use the legend command to label each curve
% Label both axis and add a title
% The subplot() commands split the left side of the figure into top and bottom.
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%

% Plot the state as a function of time:
h1 = subplot(2, 3, [1, 2]); hold on;

%%%% TODO: plot the state vs time

% Plot the derivative of each state as a function of time:
h2 = subplot(2, 3, [4, 5]); hold on;

%%%% TODO: plot the derivative of each state vs time

end
```