Math 7203-Numerical Analysis I: Cart-Pendulum System Simulation and Control

Josh Galloway (galloway.j@husky.neu.edu),
Northeastern University

April 2020

Abstract

The dynamics of the classic cart-pole system were simulated using the Python language's available initial value problem (IVP) ordinary differential equation (ODE) solver. The system was animated to allow for visualization and verification. Following this, the system was linearized around an equilibrium point to allow application of linear control theory; then, a Linear Quadratic Regulator (LQR) was constructed to control the system. The controlled system was simulated and animations produced once again to verify accuracy of the results.

1 Introduction

The classic cart-pole system appears in Figure 1. In this system, a simple pendulum is attached to a horizontally mobile cart. The pendulum swings freely without friction and the cart moves along the x-axis without friction under the influence of an applied force. This system's dynamics can be modelled by a set of autonomous nonlinear differential equations.

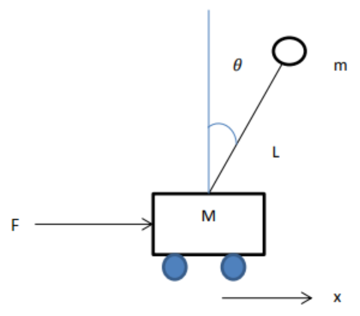

Figure 1: Cart-Pole System

Working from the nonlinear system, we have the goal of designing a controller that will perform the task of balancing the pole in the unstable upward position and moving the cart to a commanded location on the x-axis. In order to achieve this goal, the nonlinear system has to be linearized around the target equilibrium point to allow for application of linear control theory. From here, a controller constructed from the original system's linear approximation by way of calculating a control law that places the system's response eigenvalues (a.k.a poles) such that it will have a stable node at the desired resting place. To minimize the time taken to reach the desired steady-state, an LQR is applied. The LQR is constructed by minimizing a cost function weighted to achieve the desired response.

2 Cart-Pole System of Equations

The nonlinear system describing the dynamics of Figure 1 follow [Ted].

Definitions:
x = position of the cart
$\dot{x}$ = velocity of cart
$\ddot{x}$ = acceleration of cart
θ = angle of pole with respect to vertical
$\dot{\theta}$ = angular velocity of pole
$\ddot{\theta}$ = angular acceleration of pole
$g = 9.81\ \frac{m}{s^2}$, acceleration due to gravity
$M = 1.0\ kg$, mass of the cart
$m = 0.1\ kg$, mass of the pole
$L = 0.5\ m$, length of the pole
F = force applied to the cart N

$$\big(M+m\big)\ddot{x} + mL\ddot{\theta}cos(\theta) - mL\dot{\theta}^2 sin(\theta) = F$$
$$mL\ddot{x}cos(\theta) + mL^2\ddot{\theta} + mgLsin(\theta) = 0$$

$$\implies \ddot{x} = \frac{F + m * sin(\theta)\big(L\dot{\theta}^2 + g * cos(\theta)\big)}{M + m * sin^2(\theta)}$$
$$\ddot{\theta} = \frac{-F * cos(\theta) - mL\dot{\theta}^2 sin(\theta)cos(\theta) - \big(M+m\big)g * sin(\theta)}{L\big(M + m * sin^2(\theta)\big)}$$

Converting to vector form:

$$X = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}, \text{ And, } \dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix}$$

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \frac{F+m*sin(\theta)\left(L\dot{\theta}^2+g*cos(\theta)\right)}{M+m*sin^2(\theta)} \\ \dot{\theta} \\ \frac{-F*cos(\theta)-mL\dot{\theta}^2 sin(\theta)cos(\theta)-\left(M+m\right)g*sin(\theta)}{L\left(M+m*sin^2(\theta)\right)} \end{bmatrix}$$

$$= \begin{bmatrix} \dot{x} \\ \frac{m*sin(\theta)\left(L\dot{\theta}^2+g*cos(\theta)\right)}{M+m*sin^2(\theta)} \\ \dot{\theta} \\ \frac{-mL\dot{\theta}^2 sin(\theta)cos(\theta)-\left(M+m\right)g*sin(\theta)}{L\left(M+m*sin^2(\theta)\right)} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M+m*sin^2(\theta)} \\ 0 \\ \frac{-cos(\theta)}{L\left(M+m*sin^2(\theta)\right)} \end{bmatrix} F$$

3 Simulation of Uncontrolled System

From here the system was simulated using Python's "scipy.integrate.solve_ivp" function starting the initial conditions, $x = 0$, $\dot{x} = 0$, $\theta = 0.1$, and $\dot{\theta} = 0$ with no external force applied to the cart [Figure 2]. The numeric ODE solution method used was 3rd order explicit Runge-Kutta, although scipy's solve_ivp has several other options available including wrappers for methods from ODEPACK [SOL]. This method was chosen because it most closely resembles the implementations we studied in class. The system oscillates in place with no damping as would be expected in the case modelled here with no friction.

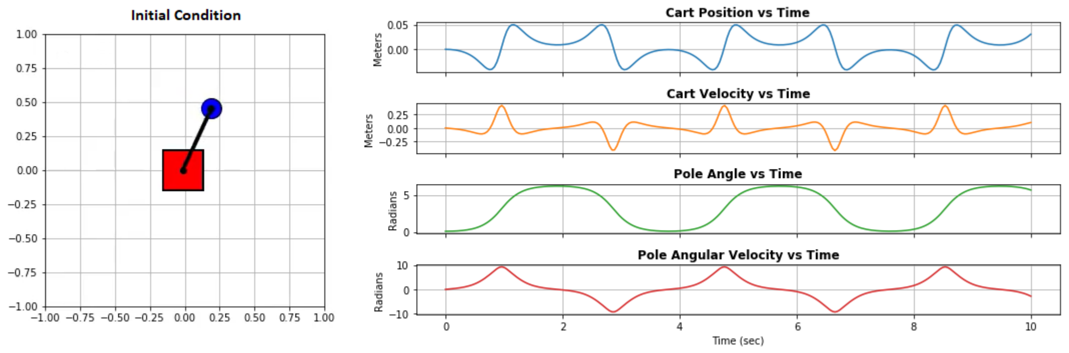

Figure 2: Cart-Pole System No Applied Force

4 Linearization For Controller Design

In order to apply linear control theory the system must be linearized. This is done by computing the Jacobian evaluated at the desired equilibrium point. Here, that point will be $x = 0$, $\dot{x} = 0$, $\theta = 0$, and $\dot{\theta} = 0$ such that the cart is at rest at the origin and the pole is balanced inverted directly above it.

Building the Jacobian,

$$\frac{\partial \dot{x}}{\partial x} = 0, \frac{\partial \dot{x}}{\partial \dot{x}} = 1, \frac{\partial \dot{x}}{\partial \theta} = 0, \frac{\partial \dot{x}}{\partial \dot{\theta}} = 0$$

$$\begin{aligned}
&\frac{\partial \ddot{x}}{\partial x} = 0, \frac{\partial \ddot{x}}{\partial \dot{x}} = 0,\\
&\frac{\partial \ddot{x}}{\partial \theta} = \frac{-m\Big(\big(mL\dot{\theta}^2 * cos(\theta) + gm + 2Mg\big) * sin^2(\theta) + 2F * sin(\theta)cos(\theta) - ML\dot{\theta}^2 * cos(\theta) - Mg\Big)}{(M + m * sin^2(\theta))^2},\\
&\frac{\partial \ddot{x}}{\partial \dot{\theta}} = \frac{2mL * sin(\theta) * \dot{\theta}}{M + m * sin^2(\theta)}
\end{aligned}$$

$$\frac{\partial \dot{\theta}}{\partial x} = 0, \frac{\partial \dot{\theta}}{\partial \dot{x}} = 0, \frac{\partial \dot{\theta}}{\partial \theta} = 0, \frac{\partial \dot{\theta}}{\partial \dot{\theta}} = 1$$

$$\begin{aligned}
&\frac{\partial \ddot{\theta}}{\partial x} = 0, \frac{\partial \ddot{\theta}}{\partial \dot{x}} = 0,\\
&\frac{\partial \ddot{\theta}}{\partial \theta} = \frac{2m * cos(\theta) * sin(\theta)((Lm\dot{\theta}^2 * cos(\theta) + gm + Mg) * sin(\theta) + F * cos(\theta))}{L(M + m * sin^2(\theta))^2} -\\
&\quad \frac{-Lm\dot{\theta}^2 * sin^2(\theta) - F * sin(\theta) + *cos(\theta)(Lm\dot{\theta}^2 * cos(\theta) + gm + Mg)}{L(M + m * sin^2(\theta))},\\
&\frac{\partial \ddot{\theta}}{\partial \dot{\theta}} = -\frac{2m * cos(\theta) * sin(\theta) * \dot{\theta}}{M + m * sin^2(\theta)}
\end{aligned}$$

And then linearizing around the equilibrium point,

$$X = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Gives the linearized state equation:

$$\dot{X} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-g(M+m)}{LM} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{-1}{LM} \end{bmatrix} F = AX + Bu$$

5 Linear Quadratic Controller Design

From the linearized system, we wish to build a controller that applies a force to the cart to both keep the pole as close to vertical as possible while moving the cart to the desired position and then, have the pole be balanced directly over the cart at steady state. From above, $\dot{X} = AX + Bu$, so we need to design a u such that the desired response is achieved.

To do this we take, $u = -KX$, where K is the matrix that will perform the control. To make this process optimal, two weighting matrices are built Q and R. Q is a diagonal matrix with its entries corresponding to the cost of deviating each state variable from the desired value. Similarly, R is a penalty for moving the manipulated variable (Force in this case). From these inputs a cost function is constructed as follows:

$$J = \int_0^\infty (X^T Q X + u^T R u)dt$$

This is an infinite-horizon linear quadratic regulator. The solution of this optimization problem is non-trivial. A publicly available solution was implemented using functions available in Python's "scipy.linalg" library as found at [Mue]. Using,

$$Q = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R = 0.01$$

This weights varying from the desired x the highest, the desired θ next highest and values deviation from velocity targets the least. Choosing R small values arriving at the desired state without regard for the amount of force applied to the cart. Solving the cost function J produces

$$K \approx \begin{bmatrix} -100 & -66.45 & -222.25 & 50.47 \end{bmatrix}$$

6 Simulation of LQR Controlled System

The system was, again, simulated using the Python 3rd order Runge-Kutta implementation. This time the initial conditions, $x = -0.75$, $\dot{x} = 0$, $\theta = 0.1$, and $\dot{\theta} = 0$ with the desired steady-state of $x = 0$, $\dot{x} = 0$, $\theta = 0$, and $\dot{\theta} = 0$. The system's response is graphed in [Figure 3].

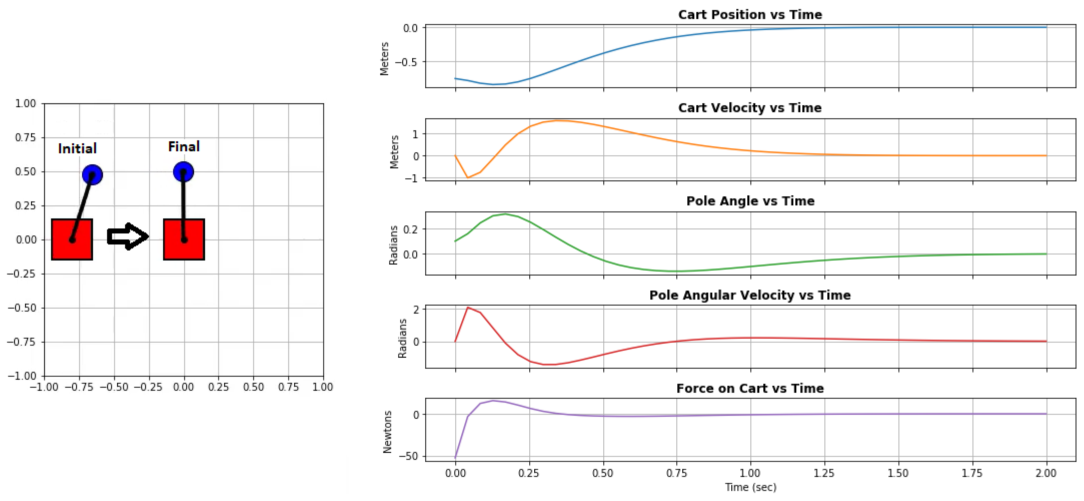

Figure 3: Cart-Pole System LQR Control

7 Conclusion

The nonlinear model was successfully simulated using the available Runge-Kutta solver from the scipy library. Following this, the system was linearized and an LQR controller designed, applied and the response simulated. The resulting simulation moved the cart to the desired spot at the origin with the pole balanced directly above in approximately 2 seconds. Overall, the simulation was reasonably performant considering the choice of implementation language. Both simulations required slightly less than 30ms to complete on laptop with an Intel i7-8650U CPU @ 1.90 GHz processor [Table 1].

Simulation	**Number of Simulations Timed**	**Length of Simulated Time Frame**	**Simulation Computing Time (mean $\pm std.$)**
Uncontrolled System	70	10s	23.9 ms $\pm 468\mu s$
LQR Controlled System	70	10s	28.9 ms $\pm 670\mu s$

Table 1: Timed Simulations

References

[Mue] M. W. Mueller. Lqr controllers with python. Retrieved from http://www.mwm.im/lqr-controllers-with-python/.

[SOL] scipy.integrate.solve _ivp. Retrieved from https://docs.scipy.org/ doc/scipy/ reference/ generated/ scipy.integrate.solve_ivp.htmlscipy.integrate.solve_ivp.

[Ted] R. Tedrake. The acrobot and cart-pole. Retrieved from https://ocw.mit.edu/courses/ electrical-engineering-and-computer-science/ 6-832-underactuated-robotics-spring-2009/readings/MIT6_832s09_read_ch03.pdf.