Math 438 Final:

Using Linear Quadratic Regulator-Trees for Optimal Control with Continuous-Control Cartpole

Math 438, Dr. Barker

Joseph Szendre, Jane Cox, Katie Palmer, Drew Henrichsen

April 24, 2018

Introduction

In this report, we present an application of a variant of the Linear Quadratic Regulator (LQR) Tree algorithm in the Continuous Cart Pole gym environment. In this paper it will be referred to as the LQR-Tree algorithm. The Continuous Cart Pole system is highly nonlinear: the state has two degrees of freedom, and the control has one degree of freedom. Members of the LQR family are good methods for solving problems like the quadcopter ball-in-cup problem—a 16-state system and a similar, albeit more complex, problem. Previously, we have demonstrated the use of regular LQR to linearize the system close to the origin, but for states far from the origin, we have to apply a different type of LQR. For our model, in order to increase the basin of attraction, and thus the robustness of the control policy, we implemented a nonlinear control policy inspired by the LQR-Trees approach.[1] The LQR-Tree algorithm is "certifiably robust to disturbances"[2] and adds additional robustness to states far away from the origin where the linearization provides a poor estimation. So instead of only linearizing the system about one unstable equilibrium point (the origin), the system is linearized for a range of states.

In this report, we will begin by setting up the optimal control problem we hope to solve before setting up the physics required for the problem. After setting up the problem, we will discuss LQR and the LQR-Tree algorithm, then implement that algorithm into our model. Finally, we will examine the results of our model and analyze the optimality of those results.

The Optimal Control Problem

The inverted pendulum and cart-pole problems are classic problems of dynamics and control theory, and some of the first control problems that humans encounter. The balance required for a toddler to first stand on their own comes from the unconscious feedback control system of the human nervous system. That is, treating our feet as the pivot of a pendulum, our nervous system uses sensory input from our eyes, muscles, joints, and inner-ear canal to calculate and exert the control necessary for us to stay upright[3].

Image obtained from https://thumb9.shutterstock.com

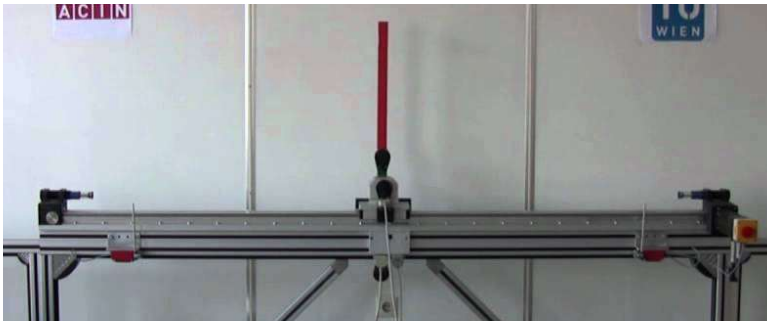
The cart-pole problem assumes an inverted pendulum sits atop a cart that can move horizontally. A pendulum has two rests points; a stable rest point directly under the pivot point

[1] https://danielpiedrahita.wordpress.com/portfolio/cart-pole-control/
[2] Tedrake, R. (2009). LQR-trees: Feedback motion planning on sparse randomized trees. Robotics: Science and Systems V. doi:10.15607/rss.2009.v.003
[3] http://www.gaitposture.com/article/0966-6362(96)82849-9/abstract

of the pendulum, and an unstable rest point directly above it. In the cart-pole problem our goal is to control the movement of the cart so that the pendulum reaches and then stays at the unstable rest point for a given period of time. Although the equations for the forces acting on the cart come from physics, the calculation of the control needed to balance the pendulum is mathematical.

The Physics of the Cart-Pole Problem

We consider a pole suspended above a moving cart of length l and mass m. Our cart has mass M. We let θ represent the angle between the ideal position of the pole (perpendicular to the car), and the actual position. We assume clockwise orientation. From the parameters of the problem, we have the following initial states: mass and length of the pendulum, initial position, angle, approximate change in position, and approximate change in angle.



We now consider the total energy of the system, $L = T - U,$ where T is kinetic energy and U is the potential energy.

The potential energy of the system is determined by the effect of gravity on the pole at any given θ. Then $U = \frac{mgl*cos(\theta)}{2}$.

The total kinetic energy is more complex. Kinetic energy of the system is determined by both the horizontal and vertical movement of the cart and pole, and the energy of rotation about the pivot point. That is,

$$T = E_{xy} + E_{rot} = \frac{M(x')^2}{2} + \frac{m}{2L}\int_0^L (x + \omega cos(\theta) * \theta)^2 + (-\omega cos(\theta) * \theta)^2 d\omega$$

where ω is the moment about θ. Simplifying further, we arrive at

$$T = \frac{(M+m)(x')^2}{2} + \frac{m(x')\theta' L\cos(\theta)}{2} + \frac{mL^2(\theta')^2}{6}$$

We then calculate the Lagrangian for our control problem, where $L = T - U$ so

$$L = \frac{(M+m)(x')^2}{2} + \frac{m(x')\theta' L\cos(\theta)}{2} + \frac{mL^2(\theta')^2}{6} - \frac{mgl*cos(\theta)}{2}$$

The path of the cart and pole satisfy the Euler-Lagrange differential equations; however, our problem requires the consideration of a nonconservative force F in the x direction. As in the lab, we apply D'Alambert's Principle to obtain F on the right side of the equation:

$$\frac{\partial L}{\partial x} - \frac{d}{dt}\frac{\partial L}{\partial' x} = F$$

$$\frac{\partial L}{\partial \theta} - \frac{d}{dt}\frac{\partial L}{\partial' \theta} = 0$$

We manipulate these equations to obtain θ'' and x'':

$$\theta'' = \frac{9.8 \cdot sin(\theta) - cos(\theta) \cdot \frac{(F + 0.05 \cdot \theta'^2 \cdot sin(\theta)}{(M+m)}}{l \cdot (\frac{4}{3} - 0.1 \cdot \frac{(cos(\theta))^2}{(M+m)}}$$

$$x'' = \frac{F + 0.05 \cdot \theta'^2 \cdot sin(\theta)}{(M+m)} - \frac{0.05 \cdot cos(\theta)}{(M+m)} \cdot \theta''$$

LQR Formulation

Computing the LQ Regulator is vital to finding the optimal control of the linear quadratic (LQ) problem. For the LQ problem, we have a standard linear state and control described by $z' = f(t, z, u) = Az + Bu$ and the cost functional $J = \int_0^\infty x^T Qx + u^T Ru \, dt$. A and B are calculated by evaluating the Frechet derivative of f with respect to z and u and evaluating them at the origin. Note that if Q and R are positive definite, then J is convex. The initial choice of Q and R can be arbitrary, so we will select values that will make Q and R positive definite, thereby making J convex.

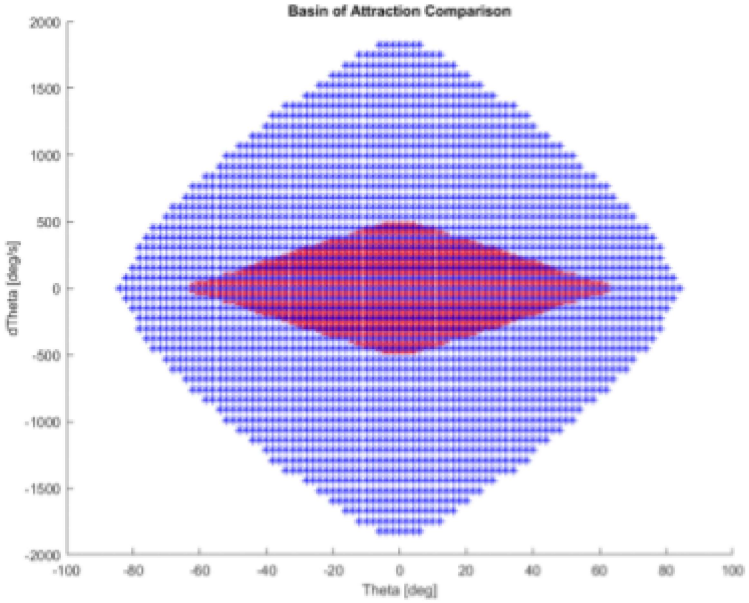
Each entry $Q[i, i]$ controls how dramatically the control policy changes in response to small perturbations in i-th state variable. If other entries of Q are nonzero, then the system responds in proportion to combinations of states.

Entries of the matrix R describe how much to penalize control expenditures in the total cost. In continuous control, $R > 0$ is necessary to ensure that there is a local minima, otherwise arbitrarily high-valued control policies are possible.

With the aforementioned construction of Q and R, we've made the problem much simpler. While the continuous version of the LQ problem is extremely complex to solve, the problem is now convex, so there exists a unique global minimum. This unique global minimum, combined with the well-posedness of the problem, allows us to use a discretized version of the cost functional J. We then use the algebraic Riccati equation, given by $PA + A^T P + Q - PBR^{-1}B^T P = 0$, for infinite-horizon optimal control problems in discrete time to solve for a linear regulator. This linear regulator adjusts our control u linearly with respect to the current state. The linear regulator computed from the linearization is arbitrarily close to optimal in sufficiently small neighborhoods of the origin. In nonlinear systems, however, optimality away from the origin drops off in proportion with higher order terms in the Taylor expansion of $f(t, z, u)$ around the origin. The formulation of A and B is further detailed in the *Formulation of our Model* section of this paper.

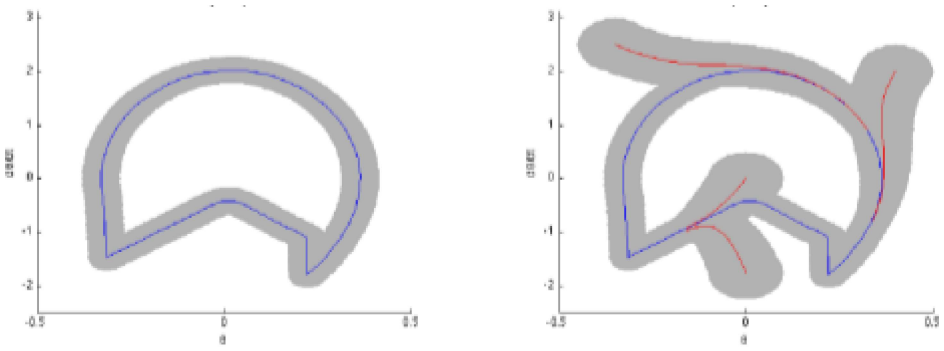
The LQR-Tree Algorithm Formulation

Though LQR worked great for stabilizing the cart-pole system around its unstable equilibrium point (what we have referred to as the origin), it's basin of attraction is limited to points around the equilibrium point. The LQR-Tree algorithm can increase the basin of attraction, as shown in the figure below.



Comparison of the basin of attraction for a regular LQR control policy (red) and a LQR-Trees control policy (blue). The x-axis is theta (degrees) and the y-axis is the derivative of theta (degrees/second)
Image from https://danielpiedrahita.wordpress.com/portfolio/cart-pole-control/

The original LQR-Trees algorithm randomly samples over some bounded state space to build a tree of feasible trajectories. With each random sample, the existing tree grows toward the random sample point, as shown in the figure below.



An illustration of how the LQR-Trees algorithm grows an existing tree to accommodate 4 randomly-sampled nodes. The x-axis is theta, and the y-axis is d(theta)/dt
Image from Tedrake, R. (2009). LQR-trees: Feedback motion planning on sparse randomized trees. Robotics: Science and Systems V. doi:10.15607/rss.2009.v.003

With some additional work, when each new branch (trajectory) is added to the existing tree, we can create a controller with a stable trajectory and then estimate the basin of attraction for the new controller. The goal of this additional calculation is to build a tree that acts as a web of local controllers that takes whatever initial conditions are inputted and pulls them toward the goal. The algorithm

terminates when all initial conditions that are probabilistically capable of reaching goal are in the tree's basin of attraction.[4] The full algorithm, as described by its original authors, is outlined below.

Algorithm 1 LQR-Tree ($\mathbf{x}_G, \mathbf{u}_G, \mathbf{Q}, \mathbf{R}$)

1: $[\mathbf{A}, \mathbf{B}] \Leftarrow$ linearization of $\mathbf{f}(\mathbf{x}, \mathbf{u})$ around $\mathbf{x}_G, \mathbf{u}_G$
2: $[\mathbf{K}, \mathbf{S}] \Leftarrow \text{LQR}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$
3: $\rho_c \Leftarrow \max \rho > 0$ s.t. $J(\mathbf{x}) + \mathbf{m}^T(\mathbf{x})\mathbf{Hm}(\mathbf{x})(\rho\text{-}J(\mathbf{x})) < 0$
4: T. init($\{\mathbf{x}_g, \mathbf{u}_g, \mathbf{S}, \mathbf{K}, \rho_c, NULL\}$)
5: **for** $\mathbf{k} = \mathbf{1}$ to K **do**
6: $\mathbf{x}_{rand} \Leftarrow$ random uniform sample, reject samples already in the basin of attraction of any tree branch; if no samples are found, then FINISH
7: $\mathbf{x}_{near}$ from cost-to-go distance metric described as $J(\mathbf{x}, t_f) = t_f + .5\mathbf{d}^T(\mathbf{x}, t_f)\mathbf{P}^{-1}(t_f)\mathbf{d}(\mathbf{x}, t_f)$
8: $\mathbf{u}_{tape} \Leftarrow \mathbf{u}(t) = \text{-}\mathbf{R}^{-1}\mathbf{B}^T e^{\mathbf{A}^T(t_f\text{-}t)}\mathbf{P}^{-1}(t_f)\mathbf{d}(\mathbf{x}(t_0), t_f)$; The extend operation.
9: **for each u in** $\mathbf{u}_{tape}$ **do**
10: $\mathbf{x} \Leftarrow$ Integrate backwards from $\mathbf{x}_{near}$ with action $\mathbf{u}$
11: $[\mathbf{K}, \mathbf{S}]$ from LQR derivation where $\mathbf{u}(t) = \text{-}\mathbf{K}(t)\mathbf{x}(t)$ and $-\mathbf{S}' = \mathbf{Q} - \mathbf{SBR}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{SA} + \mathbf{A}^T\mathbf{S}$
12: $\rho_c \Leftarrow \begin{cases} \sum_{m=0}^{N_p} \beta_{km}(t\text{-}t_k)^m \\ \rho_f, \ t=t_f \end{cases}$
13: $i \Leftarrow$ pointer to node containing $\mathbf{x}_{near}$
14: T. add $-$ node($\mathbf{x}, \mathbf{u}, \mathbf{S}, \mathbf{K}, \rho_c, i$)
15: $\mathbf{x}_{near} \Leftarrow \mathbf{x}$
16: **end for**
17: **end for**

The LQR-Tree algorithm "uses locally optimal linear feedback control policies to stabilize planned trajectories".[5]

The LQR-Tree algorithm is initially formulated similarly to the regular LQR problem, however the two problems function very differently in real time. While the LQR problem relies on a linearization close to the origin to calculate the control, the LQR-Tree algorithm calculates linearizations around a series of control points. For each point, the LQR method is used to determine an A_i and B_i:

$$A_i = \frac{df}{dx}\Big|_{(x_i,0)}, \ B_i = \frac{df}{du}\Big|_{(u_i,0)}$$

The algebraic Riccati equations are then solved for every i, and at time t when the cart reaches one of the control-points, an optimization algorithm depending on x_i is generated.

$$R_i = LQR(A_i, B_i)$$

Our algorithm is similar in that at any control point x_i, our control u relies on R_i, rather than on an R based solely on the point (0,0). However, we do not pick our control points to be outside of the

[4] Tedrake, R. (2009). LQR-trees: Feedback motion planning on sparse randomized trees. Robotics: Science and Systems V. doi:10.15607/rss.2009.v.003
[5] Tedrake, R. (2009). LQR-trees: Feedback motion planning on sparse randomized trees. Robotics: Science and Systems V. doi:10.15607/rss.2009.v.003

existing basin of attraction as used in the backward integration from different values of u. We instead neglect that calculation and just choose control points that surround the origin. That is, as the pendulum moves the linearization is calculated for its trajectory at time t rather than its initial position and trajectory at time $t = 0$ This allows the cart to reach the unstable resting point much quicker than a normal LQR method.

Formulation of Our Model

To obtain the linearization of the system, we make two simplifying assumptions. Using our equations for x" and θ" found previously, we linearize about a series of random control points, $(\theta, \theta') = (r_i, s_i)$ for all $i = |S|$, where S is our set of control conditions for the problem. We assume that our pole is of even density and height, so our center of mass lies at $.5 \cdot l$, where l is the length of the pole. We then write our linearizations as first order systems.

Taking the linearizations about (r_i, s_i), we obtain that $A_i = \frac{df}{dx_i}|_{(x_i,0)}$ the usual Jacobian of $f(x, u)$ evaluated at the origin. An example of such a matrix is given in the appendix. The control-system vector is

$$B_i = \frac{df}{du_i}|_{(u_i,0)} = [0, \frac{1}{(M+m)}, 0, \frac{-2}{(l * 1.1 * (\frac{4}{3} + 1.1)}]$$

By taking the derivative at 0 with respect to the control we see the effect that control has on the system at exactly 0.

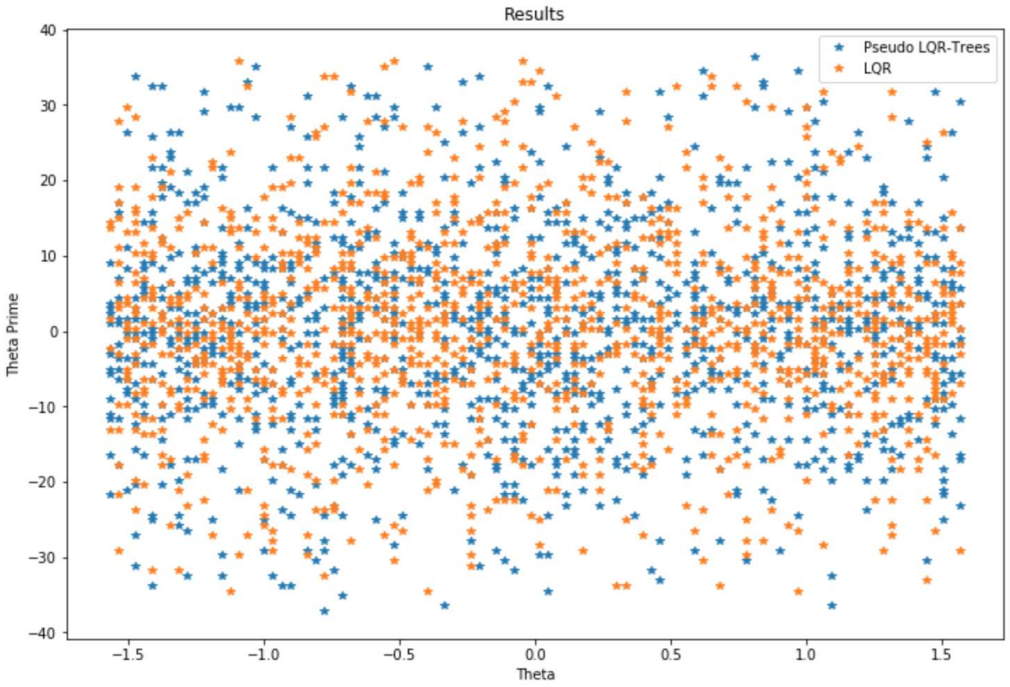
We choose Q_i = diag([.01, 2048., 1. ,1.]), R_i = [.3]. We chose these values by first evaluating different Q_i values and holding R_i constant, and then chose R_i to be the smallest value that maintained stability in the discretization. Q_{i_1} is extremely small as we're only interested in the last three values. Q_{i_2} is optimal when extremely large because the regulator is good at stabilizing the pendulum while leaving a larger translational velocity. R_i was then chosen to be the smallest possible value while preserving stability.

Results

Using the code given, we reached an average of 89.72 iterations before bringing the system to within epsilon of .01 from the origin, as compared to our previous return with regular LQR, which averaged 107.0 iterations. However, it appears that our automatic differentiation function returned a different

linearization than before, because with the same settings it obtained 91.3 iterations on average before convergence with only one control point, the origin.

We suspect there may be a possible flaw in our model. The fact that using LQR-Trees only marginally improved speed is possibly an indicator of this. Looking at the graph below, we see that while the average rate of convergence may be less for LQR-Trees, the median is not; that is, the results from both algorithms resemble each other to a worrying degree, as seen in the following graph showing the convergences resulting from a grid search across θ *in* $[-\frac{\pi}{2}, \frac{\pi}{2}]$, and θ' *in* $[-100, 100]$ radians/sec.



An anomaly that is worth noting is that it appears that our basin of attraction remained exactly the same for both the LQR and LQR-Tree algorithms instead of increasing with LQR-Trees. We checked the code for implementation errors but only become more certain of its veracity. However, our program is built off of our original LQR code. While the gradients are now taken programmatically, it is still possible that it contains an error that did not limit LQR implementation but strongly affected our LQR-Trees implementation. Maybe this result isn't statistically significant: in that case, we would need to ask why our model didn't do better than the original LQR algorithm.

However, in the process of asking why this could be the case, we found reasons why our model didn't improve the size of the basin of attraction. However, as we investigated this finding we noticed that using linearizations about points further away from the origin as though we were linearizing about the

origin results in our estimates being overamplified. It's as if we were assuming the derivative of x^2 at 0 was its derivative at 5 and interpolating from 0 to 5, which would result in an estimate of 50. While LQR will still attempt to move the state towards the origin it is doing so at a rate that is actually too fast for the given R value. We further hypothesize that the following could have factored into our model having the same basin of attraction. An important step of that algorithm is to use the regular LQR method to find our matrix R_t. As R_t is then used to calculate the control, a misstep there would have profound consequences on the rest of the algorithm implementation. Another possibility for this strange result is that we assumed that our different linearizations for the LQR-Tree algorithm were about the origin, which could potentially cause the estimation to overshoot our $f(t, x, u)$; a more asymmetrical approach may yield different results. We conclude that a refinement of how we apply LQR linearization about a non-fixed point while the system is in motion is needed.

Analysis of Optimality

Although the initial LQR algorithm works passably well for a system with initial states close to the origin, the LQR-Tree algorithm, when run correctly, allows us greater range of initial conditions, and is much more optimal for that range. This allows us to overcome the problems of using linearizations for nonlinear systems by using a number of points about which linearizations and optimal control policies are computed. Thus we achieve near optimality in almost any region desired. Rather than rely on the control policy calculated by LQR near the origin, LQR-Trees calculates the control policy associated with the point closest to the current state. This calculation is then used by the controller at each time step. Although the LQR algorithm suffers from instability caused by discretization. LQR-Trees overcomes this problem by continuously adjusting the control based on which initial conditions the pole is closest to. Although instability still occurs, this severely limits it. It must be noted, however, that for an initial condition near $\theta = 0$, LQR-Trees will not be optimal, as algorithm takes extra time needed to linearize about several other initial conditions that will not affect the final solution. In such a case, we expect the LQR or another such algorithm to be the most optimal.

Conclusion

We were unable to replicate the results shown earlier, but also did no worse. The version of the LQR-Tree algorithm we used may not be the fastest, but are still faster than other optimization methods like dynamic programming and can be used in real time applications. For cases where the pole is close to the origin, the feedback controllers used are almost identical to the one in the LQR method

(and actually are identical when closer than any other control points). Where the LQR-Tree algorithm will truly shine is in its application of expanding the basin of attraction to arbitrarily large regions, so long as they remain controllable. The ability to balance the pole from diverse areas effectively and efficiently has far-reaching consequences for technology such as hoverboards. For further research, we need to be able to formulate LQR in such a way that your linearization is about a point different from that of the origin. That point will most often not be a fixed point, and will therefore add a bias to our linear estimate. This would be very helpful as we optimize the implementation of the LQR-Trees algorithm, both in a simulated setting as well as on a physical cart-pole system.

Appendix

$$Q := \begin{bmatrix} 1.000e\text{-}02 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 2.048e+03 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 1.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.000e+00 \end{bmatrix}$$

$$R := [[0.3]]$$

$$A_0 := \begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & 0. & -0.717 & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 0. & 15.776 & 0. \end{bmatrix}$$

$$B_0 := \begin{bmatrix} 0. \\ 0.97560972 \\ 0. \\ -1.4634\,1467 \end{bmatrix}$$

$$Reg_0 := \begin{bmatrix} 4.53298014e+00 & 3.39544873e+00 & 6.33365077e+00 & 2.33470185e+00 \\ 3.39544873e+00 & 1.53787788e+03 & 2.86869662e+03 & 1.05745491e+03 \\ 6.33365077e+00 & 2.86869662e+03 & 5.44163425e+03 & 2.00525661e+03 \\ 2.33470185e+00 & 1.05745491e+03 & 2.00525661e+03 & 7.39152586e+02 \end{bmatrix}$$

$$Feedback\ gain\ vector\ K_0 := [1.02469508e\text{-}02\ 2.82900342e+01\ 1.66686573e+02\ 4.36259489e+01]$$