PID:17/Shreya Rastogi 10^{th} June, 2020

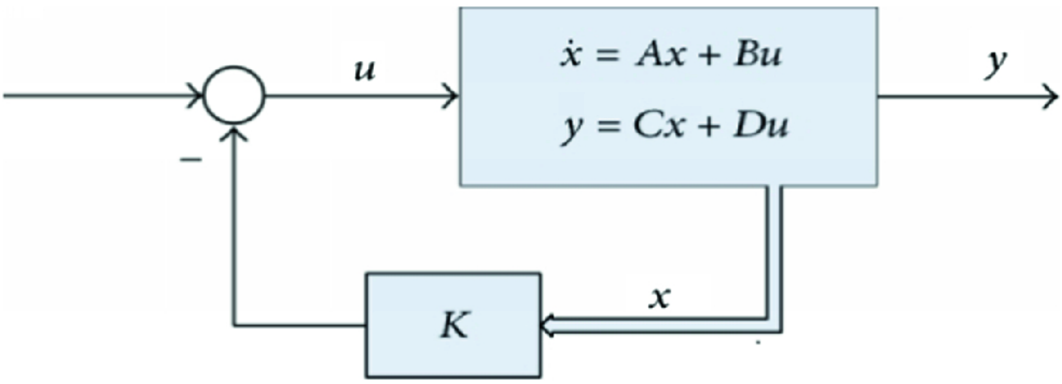
Linear Quadratic Regulator (LQR) in Octave

1 Explanation

LQR is a method in modern control theory that uses state-space approach to analyze systems. Here the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic function.
LQR is a powerful optimisation tool and control algorithm for single or multi-input systems.
The **Q** matrix defines the **weights on the states** while **R** matrix defines the **weights on the control input** in the cost function.

How to implement LQR control in Octave?

Thankfully, implementing LQR control in Octave doesn't require us to solve the Ricatti equation. We have a simple function which is used to calculate the value of the optimal state feedback control gain **K matrix**.



We use the linear quadratic regulation method for determining our state-feedback control gain matrix K. The MATLAB function **lqr** allows us to choose two parameters, R and Q, which will balance the relative importance of the control effort (u) and error (deviation from 0), respectively, in the cost function that we are trying to optimize.

The new state matrix $(A - BK)$ defines the dynamics of the system where $-Kx$ is fed as input and is known as state feedback system. The system stability can be calculated by finding the eigenvalues of the $(A - BK)$ matrix.

lqr in Octave

The code for finding out K matrix of a Cyclebot system.

Cycle__bot.m

```
1;
pkg load control;
clear all;
close all;
m1 = 0.915821;
m2 = 0.5468;
L1 = 0.12;
L2 = 0.20;
I1 = 0.004626;
I2 = 0.002318;
g = 9.81;
a = (m1*(L1**2)) + (m2*(L2**2)) + I1;
b = (m1*L1 + m2*L2)*g;

A = [0 1 0 0;                           % state matrix
     b/a 0 0 0;
     0 0 0 1;
     -b/a 0 0 0];
B = [0;-1/a;0(a+I2)/(a*I2)];            % input matrix
C = eye(4);                             % output matrix
D = [0;0;0;0];                          % feed-forward matrix
Q = [200 0 0 0;
     0 10 0 0;                          % Q matrix of system
     0 0 1 0;
     0 0 0 1];
R = 10;                                 % R parameter
K = lqr(A,B,Q,R)                        % lqr function
```

As seen from the code above we use the lqr function to find out the K matrix.

dlqr function in Octave

When using the lqr function to find the K matrix for real world implementation we need to modify it a little bit. Since the readings coming from the sensors are discrete we need to convert the continuous system into a discrete system. This is achieved by using the **c2d function** in octave. Once the system is converted into a discrete system we then use the **dlqr function** to find the K matrix.

$$sysd = c2d(sysc, Ts)$$

discretizes the continuous-time dynamic system model sysc using zero-order hold on the inputs and a sample time of Ts.

$$A\_d = sysd.A;$$

This is how we convert the continuous A matrix into a discrete one.

k_values_lqr.m

```
1;
close all;
clear all;
pkg load control;
km= 0.27785508333; ke= 0.38197186342;
Mp= 0.830; Ip= .002163283;
l= .03025; r= 0.0325; Res= 4; Mw= 0.046;
Iw= .0000485875; g= 9.81;
beeta= (2*Mw + (2*Iw/(r**2)) + Mp);
alphaa= (Ip*beeta + 2*Mp*(l**2)*(Mw+ Iw/(r**2)));

% A matrix of the Biped bot
A = [0 1 0 0;
    0 (2*km*ke*(Mp*l*r-Ip-Mp*(l**2)))/(Res*(r**2)*alphaa) ((Mp**2)*g*(l**2))/alphaa 0;
    0 0 0 1;
    0 (2*km*ke*(r*beeta-Mp*l))/(Res*(r**2)*alphaa) (Mp*g*l*beeta)/alphaa 0];
% B matrix of the Biped bot
B = [0;
    (2*km*(Ip+Mp*(l**2)-Mp*l*r))/(Res*r*alphaa);
    0;
    (2*km*(Mp*l-r*beeta))/(Res*r*alphaa)];

C = eye(4);                    % C matrix
D = [0;0;0;0];                 % D matrix
Ts = 1/100;                    % sample time
sys_s = ss(A,B,C,D);           % ss function for state space representation
```

k_values_lqr.m

```
sys_d = c2d(sys_s,Ts,'zoh');  %{ c2d discretizes the continuous-time dynamic
                                 % system model sysc using zero-order hold
                                 % on the inputs and a sample time of Ts % }

A_d = sys_d.A;                 % A_d is the discrete A matrix
B_d = sys_d.B;                 % B_d is the discrete B matrix
Q = [1e5 0 0 0;
    0 8 0 0;
    0 0 12 0;
    0 0 0 1e5];
R = 100;
K = dlqr(A_d,B_d,Q,R)           % dlqr is used to find K

Ac = [(A_d-B_d*K)];             % (A - BK)
Bc = [B_d];
Cc = [sys_d.C];                 % Cc is the discrete C matrix
Dc = [sys_d.D];                 % Dc is the discrete D matrix
x_initial = [0,0,0.0872665,0]; % initial point
x_set = [1;2;2;1];              % end point
sys_cl = ss(Ac,Bc,Cc,Dc,Ts);
t = 0:0.01:20;
[y,t,x] = initial(sys_cl,x_initial,t);

for i= 1:size(t)(1)
  u(i) = -K*x(i,:)';
endfor
```

Above is the implementation and use of dlqr function in finding the value of K and using the LQR controller to control a real world system of a self-balancing robot.