Question 5

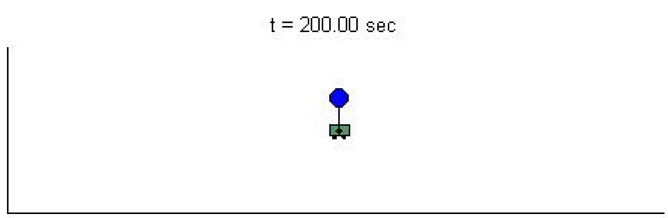
Code

```
function[x_bar,u_bar,l,L] = ilqr_solution(f,linearize_dyn, Q, R, Qf, goal_state, x0, u_bar, num_steps, dt)

% init l,L
n = size(Q,1);
m = size(R,1);

l = zeros(m,num_steps);
L = zeros(m,n,num_steps);

% init x_bar, u_bar_prev
x_bar = zeros(n,num_steps+1);
x_bar(:,1) = x0;
u_bar_prev = 100*ones(m,num_steps); %arbitrary value that will not result in termination

% termination threshold for iLQR
epsilon = 0.001;

% initial forward pass
for t=1:num_steps
    x_bar(:,t+1) = f(x_bar(:,t),u_bar(:,t),dt);
end
x_bar_prev = x_bar;

while norm(u_bar - u_bar_prev) > epsilon
    % we use a termination condition based on updates to the nominal
    % actions being small, but many termination conditions are possible.

    % ----- backward pass

    % We quadratize the terminal cost C_T around the current nominal trajectory
    % C_T(dx,du) = 1/2 dx' * QT * dx + qf' * dx + const

    % the quadratic term QT=Qf, but you will need to compute qf

    % the constant terms in the cost function are only used to compute the
    % value of the function, we can ignore them if we only care about
    % getting our control

    % TODO: compute linear terms in cost function
    qf = Qf' * (x_bar(:,end) - goal_state);

    % initialize value terms at terminal cost
    P = Qf;
    p = qf;

    for t=num_steps:-1:1
        % linearize dynamics
        [A,B,c] = linearize_dyn(x_bar(:,t),u_bar(:,t),dt);

        % TODO: again, only need to compute linear terms in cost function
        q = Q' * (x_bar(:,t) - goal_state);
        r = R' * u_bar(:,t);

        [lt,Lt,P,p] = backward_riccati_recursion(P,p,A,B,Q,q,R,r);
        l(:,t) = lt;
        L(:,:,t) = Lt;
    end

    % ----- forward pass
    u_bar_prev = u_bar; % used to check termination condition

    for t=1:num_steps
        % TODO: implement control update
        dx = x_bar(:,t) - x_bar_prev(:,t);
        du = l(:,t) + L(:,:,t) * dx;
        u_bar(:,t) = u_bar_prev(:,t) + du;
        x_bar(:,t+1) = f(x_bar(:,t),u_bar(:,t),dt);
    end

    x_bar_prev = x_bar; % used to compute dx

end

end
```

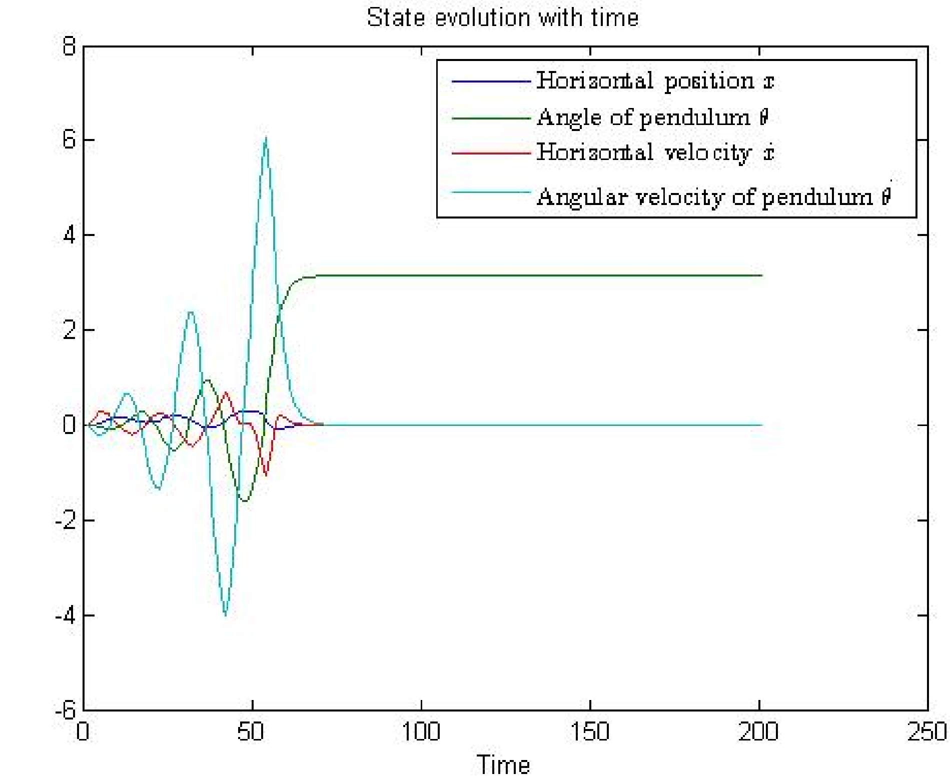
```
function [l,L,P,p] = backward_riccati_recursion(P,p,A,B,Q,q,R,r)
% TODO: write backward riccati recursion step,
% return controller terms l,L and value terms p,P
% refer to lecture 4 slides
n = size(Q,1);
m = size(R,1);
H = zeros(m,n); % no cross term for us
Q_uuk = R + B' * P * B;
Q_xxk = Q + A' * P * A;
Q_uxk = H + B' * P * A;
Q_uk = r + B' * p;
Q_xk = q + A' * p;
L = -Q_uuk\Q_uxk;
l = -Q_uuk\Q_uk;
pnew = Q_xk - L'*Q_uuk*l;
Pnew = Q_xxk - L'*Q_uuk*L;
P = Pnew;
p = pnew;
end
```

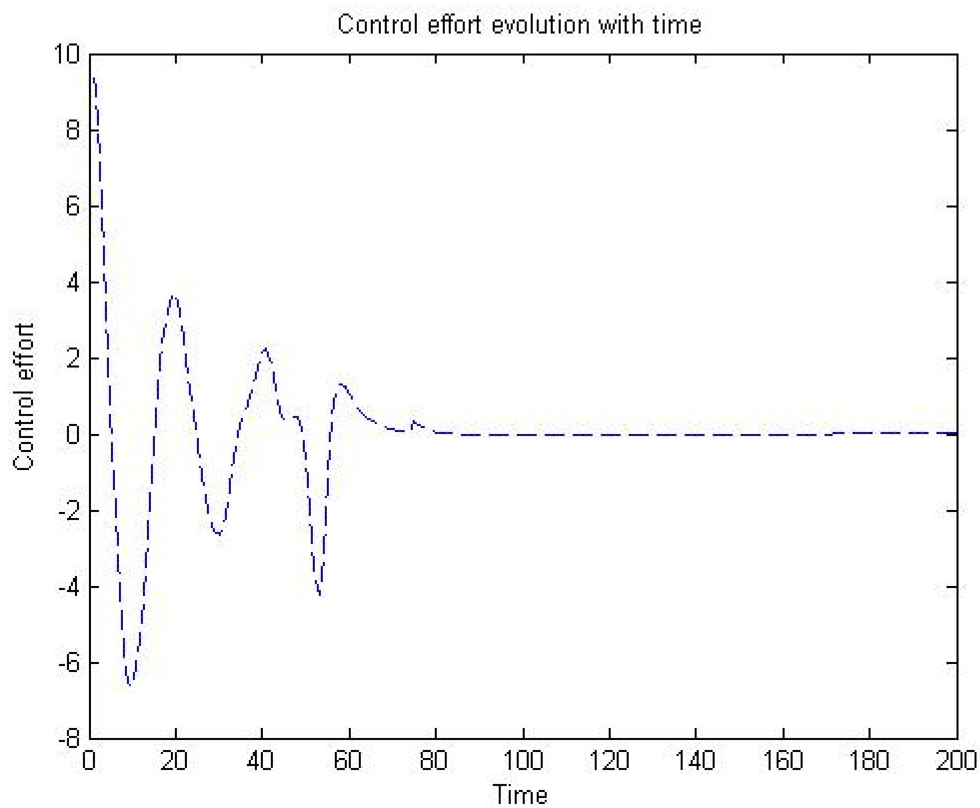
Plots

All the simulations end at this point in the animation

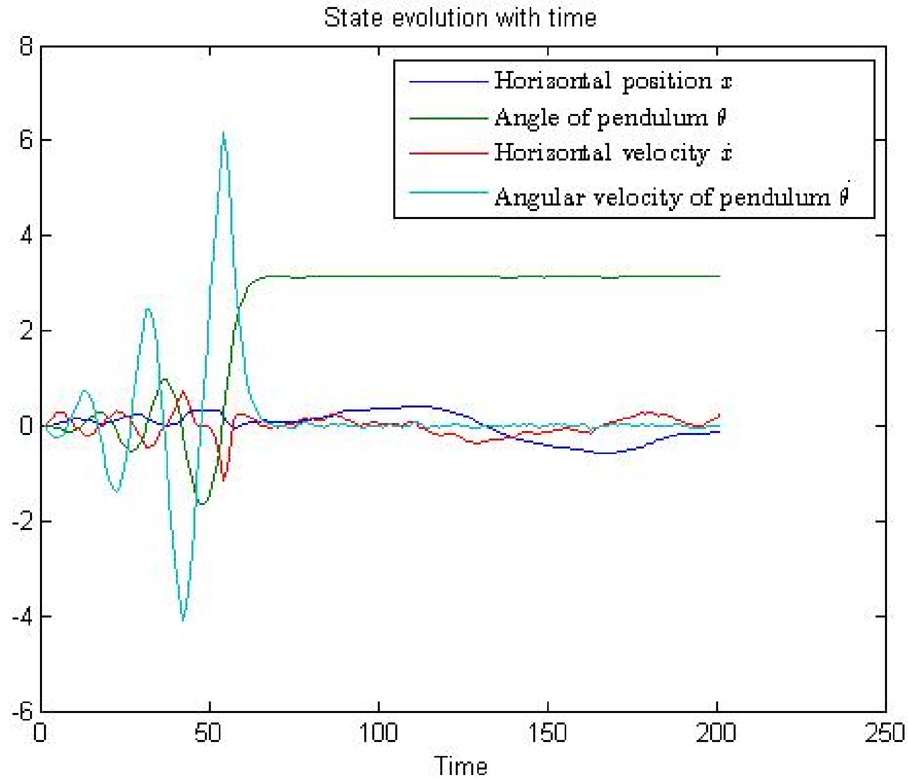


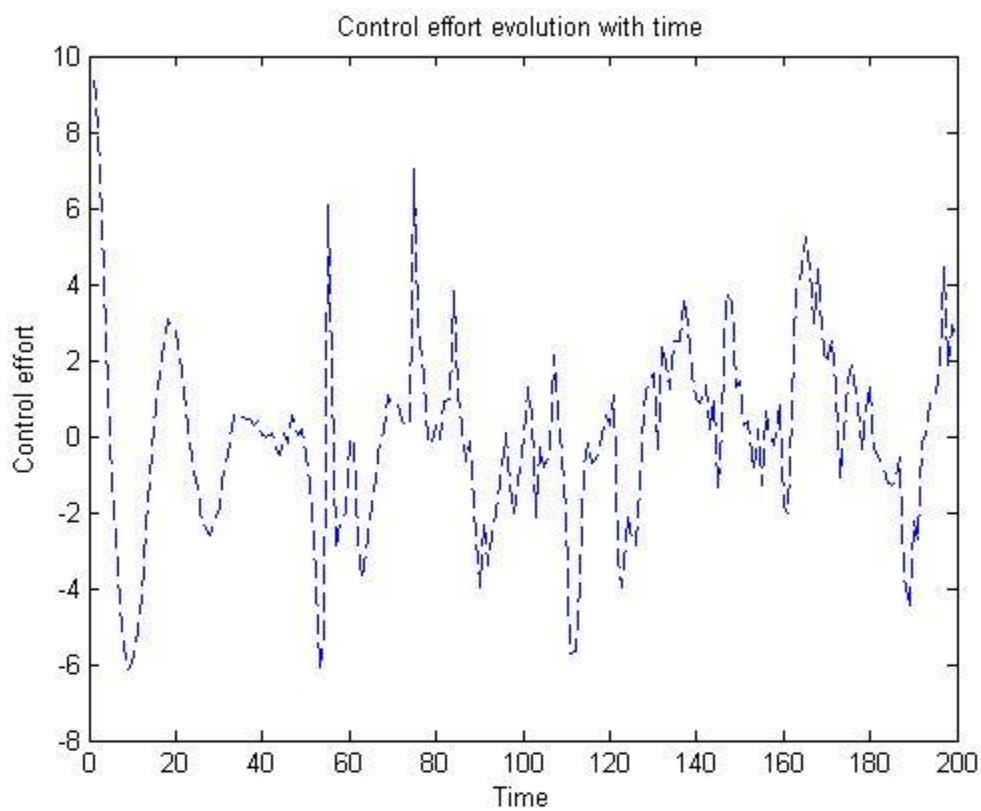
Without Noise





With Noise


State evolution with time
Horisontal position x
Angle of pendulum θ
Horisontal velocity $\dot{x}$
Angular velocity of pendulum $\dot{\theta}$
8
6
4
2
0
-2
-4
-6
0
50
100
150
200
250
Time


Control effort evolution with time
Control effort
10
8
6
4
2
0
-2
-4
-6
-8
0
20
40
60
80
100
120
140
160
180
200
Time