```
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 =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) - Qf*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) - Q*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

        x = %???
        delx = x - x_bar(:,t);
        u_bar(:,t) = u_bar(:,t) + (l(:,t)+L(:,:,t)*delx);
        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

%Perform math using notation in DP notes where (V=P, v=p)
Suk = r + B'*p;
Suuk = R + B'*P*B;
Suxk = B'*P*A;

Lk = -pinv(Suuk)*Suxk;
lk = -pinv(Suuk)*Suk;
```

```
P = Q + A'*P*A-Lk'*Suuk*Lk;
p = q + A'*p + Suxk'*lk;

%Define our return variables
L = Lk;
l = lk;
end

 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) - Qf*goal_state;

    % initialize value terms at terminal cost
```

```matlab
    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) - Q*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

        x = %???
        delx = x - x_bar(:,t);
        u_bar(:,t) = u_bar(:,t) + (l(:,t)+L(:,:,t)*delx);
        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

%Perform math using notation in DP notes where (V=P, v=p)
Suk = r + B'*p;
Suuk = R + B'*P*B;
Suxk = B'*P*A;

Lk = -pinv(Suuk)*Suxk;
lk = -pinv(Suuk)*Suk;
```

```
P = Q + A'*P*A-Lk'*Suuk*Lk;
p = q + A'*p + Suxk'*lk;

%Define our return variables
L = Lk;
l = lk;
end
```