Solution for Exercise 2: Dynamic programming

```
TEMPO Summer School on Numerical Optimal Control and Embedded Optimization
University of Freiburg, July 27 - August 7, 2015
Rien Quirynen, Dimitris Kouzoupis and Moritz Diehl
```

Contents

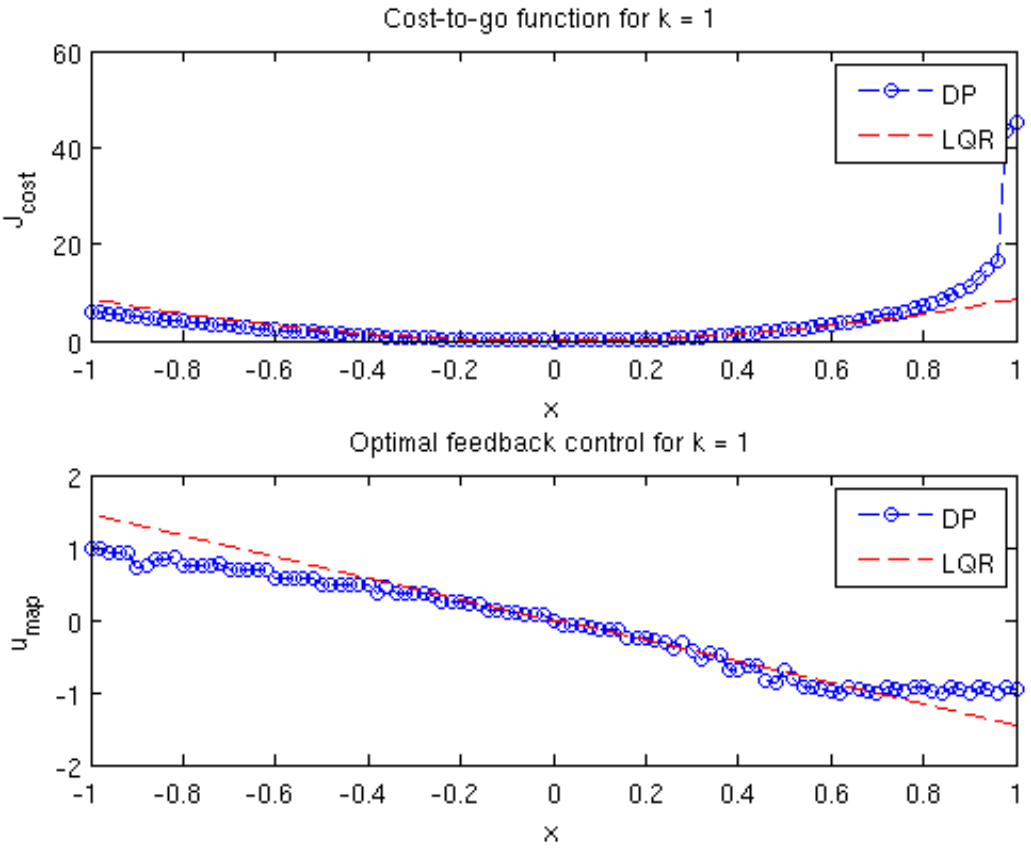
Part 1: LQR design

```
clc;
clear all;
close all;

% linearize
input.Ts = 0.1; input.nSteps = 3;
x_lin = 0; u_lin = 0;
input.x = x_lin; input.u = u_lin;
output = RK4_integrator( @ode, input );
A = output.sensX;
B = output.sensU;

% dlqr
Q = 1; R = 1;
[K,P] = dlqr(A,B,Q,R);
save lqr.mat A B Q R K P
```

Part 2: Dynamic programming

```
N = 20;
Ts = 0.1;
input.Ts = 0.1;
input.nSteps = 1;
input.sens = 0; % no sensitivities are needed here

N_x = 101; % number of discretization points for state x
x_values = linspace(-1,1,N_x);

N_u = 101; % number of discretization points for control u
u_values = linspace(-1,1,N_u);

load lqr.mat A B Q R K P

% initialize the cost-to-go function with the terminal cost
J_cost = P*x_values.^2;
u_map = u_values;
for k = N-1:-1:1
    J_new = inf+J_cost;
    for i = 1:N_x
        x_k = x_values(i);
        input.x = x_k;
        for j = 1:N_u
```

```
            u_k = u_values(j);
            input.u = u_k;
            output = RK4_integrator( @ode, input );
            i_next = project(output.value,x_values);
            if i_next <= 0 || i_next > N_x
                cost = inf;
            else
                cost = x_k^2+u_k^2 + J_cost(i_next);
            end
            if cost < J_new(i)
                J_new(i) = cost;
                u_map(i) = u_k;
            end
        end
    end
    J_cost = J_new;

    figure(1);
    clf;
    subplot(211);
    plot(x_values,J_cost,'--ob'); hold on;
    plot(x_values,P*x_values.^2,'--r');
    xlabel('x'); ylabel('J_{cost}');
    legend('DP', 'LQR')
    title(['Cost-to-go function for k = ' num2str(k)])
    drawnow

    subplot(212);
    plot(x_values,u_map,'--ob'); hold on;
    plot(x_values,-K*x_values,'--r');
    xlabel('x'); ylabel('u_{map}');
    legend('DP', 'LQR')
    title(['Optimal feedback control for k = ' num2str(k)])
    drawnow

end
save DP.mat x_values u_values J_cost u_map
```

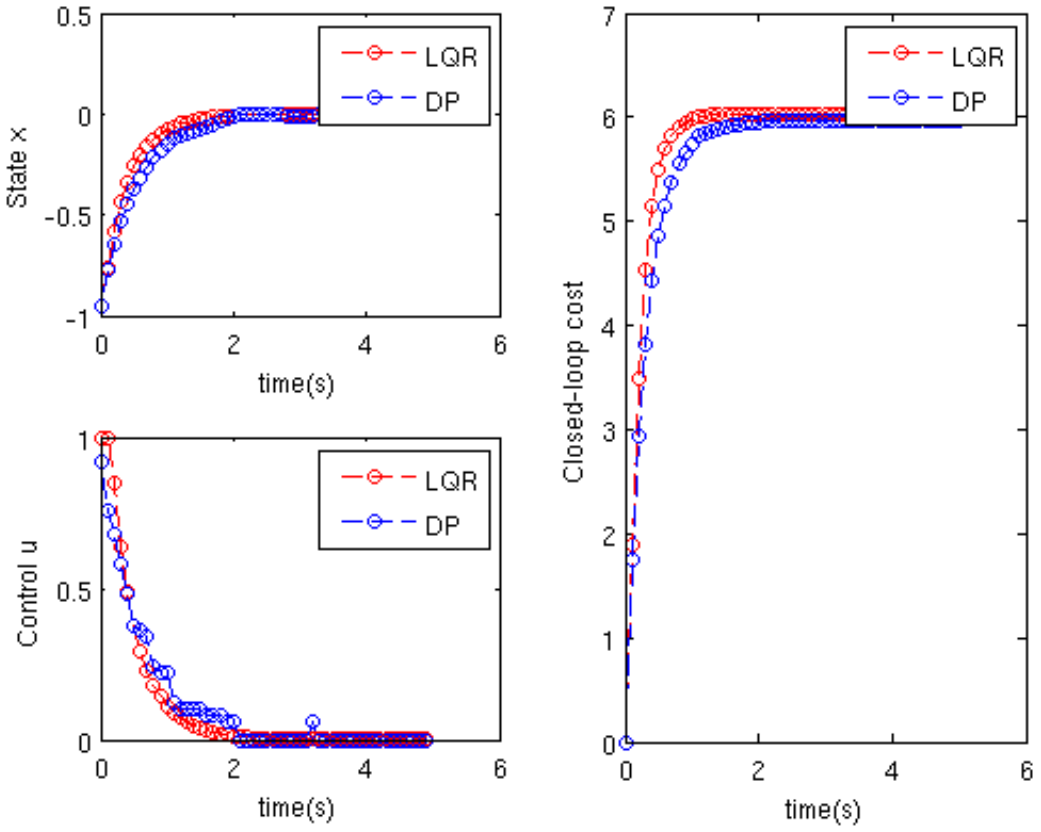


Part 3: Closed-loop simulation

Now let us perform a closed-loop simulation using the DP based controller and compare it to the LQR design

```
input.Ts = 0.1; input.nSteps = 3;
input.sens = 0; % no sensitivities are needed here
x0 = -0.95;

load lqr.mat A B Q R K P
load DP.mat x_values u_values J_cost u_map

% Closed-loop simulation
time = 0;
Tf = 5;
state_LQR = x0;
state_DP = x0;
us_DP = [];
us_LQR = [];
cost_DP = [0];
cost_LQR = [0];
iter = 0;
figure(1);
while time(end) < Tf
    % optimal feedback law from LQR
    u_LQR = min(max(-K*state_LQR(:,end),-1),1); us_LQR = [us_LQR u_LQR];
    cost_LQR = [cost_LQR cost_LQR(end)+u_LQR^2+state_LQR(:,end)^2];

    % apply LQR control to nonlinear system
    input.x = state_LQR(:,end);
    input.u = u_LQR;
    output = RK4_integrator( @ode, input );
    state_LQR(:,end+1) = output.value;
```

```
    % optimal feedback law from DP
    index = project(state_DP(:,end),x_values);
    u_DP = u_map(index); us_DP = [us_DP u_DP];
    cost_DP = [cost_DP cost_DP(end)+u_DP^2+state_DP(:,end)^2];

    % apply DP control to nonlinear system
    input.x = state_DP(:,end);
    input.u = u_DP;
    output = RK4_integrator( @ode, input );
    state_DP(:,end+1) = output.value;

    % next time step and visualize result
    iter = iter+1;
    time(end+1) = iter*Ts;

    % plot results
    clf
    subplot(221);
    plot(time,state_LQR,'--ro'); hold on;
    plot(time,state_DP,'--bo');
    xlabel('time(s)');
    ylabel('State x');
    legend('LQR', 'DP')

    subplot(223);
    plot(time(1:end-1),us_LQR,'--ro'); hold on;
    plot(time(1:end-1),us_DP,'--bo');
    xlabel('time(s)');
    ylabel('Control u');
    legend('LQR', 'DP')

    subplot(2,2,[2 4])
    plot(time,cost_LQR,'--ro'); hold on;
    plot(time,cost_DP,'--bo');
    xlabel('time(s)');
    ylabel('Closed-loop cost');
    legend('LQR', 'DP')

    pause(Ts/2);
end
```



Published with MATLAB® R2014a