Problem 2:

```matlab
% Problem 2 - continuous LQR

% Define system parameters
dt = 0.1;

r = 3;
q = 1;
h = 4;
Q = [q 0; 0 0];
R = r;
A = [0, 1; 0 -1];
B = [0; 1];
tf = 10;
Qf = [0 0;0 h];
tspan = [tf:-dt:0];
x0 = [1;1];


% Reshape final condition as column vector for ode45
Qf_col = Qf(:);
% Solve Riccati ODE
[tV, V_col] = ode45(@(tV,V_col)cont_lqr_riccati(tV,V_col,A,B,Q,R), tspan, Qf_col);
ep_length = size(V_col,1);

% Here, we use Matlab's built in LQR function to confirm our results
[K,S,E] = lqr(A, B, Q, R, 0);

gains = zeros(2, ep_length);
for i=1:ep_length
    gains(:,i) = (1/R)*B.'*reshape(V_col(i,:), [2,2]);
end

% Use solution of Riccati equation to compute gain, and trajectory
[t, x] = ode45(@(t,x)dyn(t,x,A,B,V_col,tV,R), [0:dt:tf], x0);

fig = figure;
plot(x)
legend("x1", "x2")
xlabel("time")
title(sprintf("state trajectory for tf = %s", int2str(tf)))
saveas(fig, sprintf("p2_state_tf_%s.png", int2str(tf)))

fig = figure;
plot(gains');
legend("l1","l2")
xlabel("time")
title(sprintf("gains for tf = %s", int2str(tf)))
saveas(fig, sprintf("p2_gains_tf_%s.png", int2str(tf)))

function dxdt = dyn(t,x,A,B,V_col,tV,R)
    closest_t = interp1(tV,1:length(tV),t,"nearest");
    V_curr = reshape(V_col(closest_t,:), [2,2]);
    K_curr = (1/R)*B.'*V_curr;
    dxdt = A*x + B*-K_curr*x;
end
```

```
% Ricatti equation for continuous LQR
% This is also used for problem 3

function dVdt_col = cont_lqr_riccati(t,V_col,A,B,Q,R)
% Shape V_col into [2,2] matrix
V = reshape(V_col, [2,2]);

dVdt = -1 * (Q - V*B*(1/R)*B.'*V + V*A + A.'*V);

%Reshape dVdt into column vector (4,1)
dVdt_col = dVdt(:);
end
```

Problem 3

```
% Problem 3

% Define system parameters
A = [0 1;1 0];
B = [0; 1];
Q = [3 0;0 3];
R = 1;
Qf = [0 0;0 0];
C = [0 1];

tf = 15;
dt = 0.01;
%%  Part a

% Continuous LQR Riccati equation
% Reshape final condition as column vector for ode45
Qf_col = Qf(:);
% Solve Riccati ODE
[tV, V_col] = ode45(@(tV,V_col)cont_lqr_riccati(tV,V_col,A,B,Q,R), [tf:-dt:0], Qf_col);
ep_length = size(V_col,1);

LQR_gains = zeros(2, ep_length);
% Use solution of Riccati equation to compute gain
for t=1:ep_length
  V_curr = reshape(V_col(t, :),[2,2]);
  L = (1/R) * B.' * V_curr;
  LQR_gains(:,t) = L;
end

% Continuous Kalman filter
sigma0 = eye(2) * 5;
xhat0 = [0, 0];
Sw = [0 0;0 4]; % Process noise covariance
Sv = 0.5; % Measurement noise covariance

[tK, S_col] = ode45(@(tK,S_col)cont_kalman_sigma(tK,S_col,A,C,Sw,Sv), [0:dt:tf], sigma0(:));

% Compute Kalman gain
K_gains = zeros(2, ep_length);
for t=1:ep_length
  S_curr = reshape(S_col(t, :),[2,2]);
  K = S_curr * C.' * (1/Sv);
  K_gains(:,t) = K(:);
end

fig = figure;
plot(K_gains');
legend("L1","L2")
xlabel("time")
title("Kalman gains")
saveas(fig, "p3_Kgains.png")

% Plot gains
fig = figure;
plot(LQR_gains');
legend("L1","L2")
xlabel("time")
title("LQR Gains")
```

```
saveas(fig, "p3_LQRgains.png")

%% Part c

% Simulate closed-loop system

x_0 = [10; -10];
xc_0 = [0; 0];
Sys0 = [x_0; xc_0];

% For now, use steady-state gains (change to time-dependent when this
% converges)

m_noise = randn(size(V_col,1),1) * Sv;
p_noise = randn(size(V_col,1),2);
[t,Sys] = ode45(@(t,Sys)closed_loop(t, Sys, A, B, C, R, V_col, K_gains, tV, tK, m_noise, p_noise), [0:dt:tf], Sys0);

fig = figure;
plot(Sys)
legend("x1", "x2", "xc1", "xc2")
title("True state and state estimates")
saveas(fig, "p3.png")

% Solve coupled ODEs for dxcdt and dxdt
% Sys is a (4,1) column vector that stacks x and xc
function dSysdt = closed_loop(t, Sys, A, B, C, R, V_col, K_gains, tV, tK,m,p)
  x = Sys(1:2);
  xc = Sys(3:4);

  closest_t_ind_V = interp1(tV,1:length(tV),t,"nearest");
  closest_t_ind_K = interp1(tK,1:length(tK),t,"nearest");

  V = reshape(V_col(closest_t_ind_V, :),[2,2]);
  K = reshape(K_gains(:, closest_t_ind_K),[2,1]);

  Ac = A - B*(1/R)*B.'*V - K*C;
  Bc = K;
  Cc = (1/R)*B.'*V;
  Sw = [0 0;0 4]; % Process noise covariance
  dxdt = A*x - B*Cc*xc + Sw*p(closest_t_ind_V,:).';
  dxcdt = Ac*xc + Bc*C*x + m(closest_t_ind_V);

  dSysdt = [dxdt;dxcdt];
end


% sigma ode for continuous time Kalman filter

function dSdt_col = cont_kalman_sigma(t,S_col,A,C,Sw,Sv)
% Shape S_col into [2,2] matrix
S = reshape(S_col, [2,2]);

dSdt = A*S + S*A.' + Sw - S*C.'*Sv*C*S;

%Reshape dVdt into column vector (4,1)
dSdt_col = dSdt(:);
end
```

Problem 5:

cost.m

```
% Put here the cost
c = -1 * y(end);
```

fDyn.m

```
% Put here the dynamics
xDyn = -x.*u + y.*u.*u;
yDyn = x.*u - 3.*y.*u.*u;
```

constraint.m

```
% Put here constraint inequalities
c = 0; % No inequality constraints

% Note that var = [x;y;u]
x = var(1:N+1); y = var(N+2:2*N+2); u = var(2*N+3:3*N+3);
[xDyn,yDyn] = fDyn(x,y,u);
% Computing dynamical constraints via the trapezoidal rule
h = 1.0*T/(1.0*N);
for i = 1:N
   % Provide here dynamical constraints via the trapeziodal formula
   ceq(i) = 0.5 * (xDyn(i) + xDyn(i+1)) * h + x(i) - x(i+1); %x
   ceq(i+N) = 0.5 * (yDyn(i) + yDyn(i+1)) * h + y(i) - y(i+1) ; %y
end

% Put here initial conditions
ceq(1+2*N) = x(1) - x0;
ceq(2+2*N) = y(1) - y0;
```