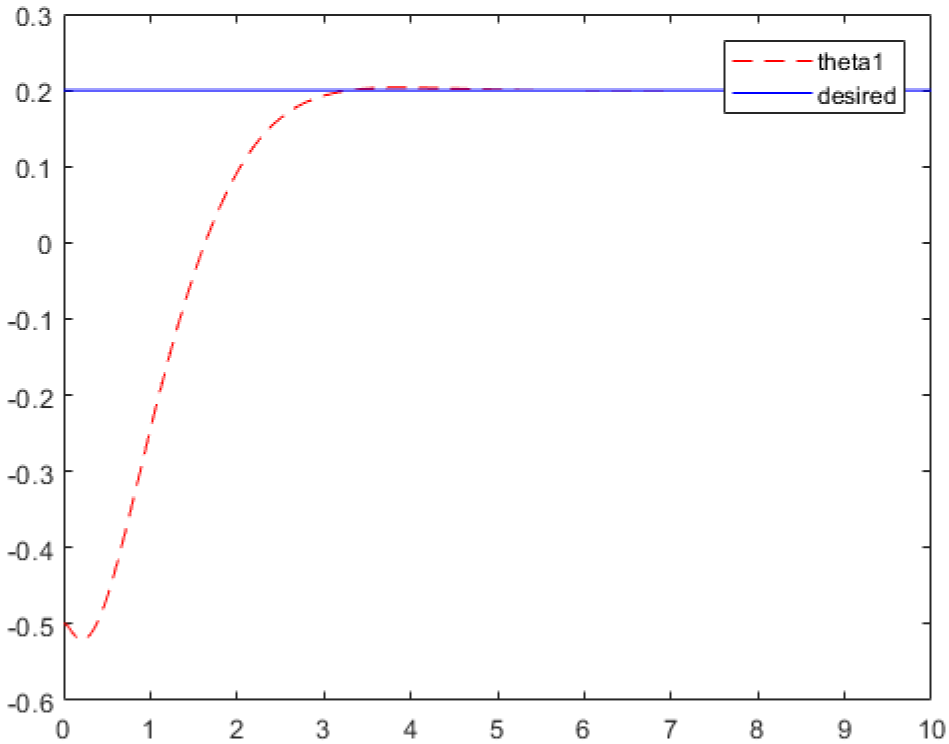
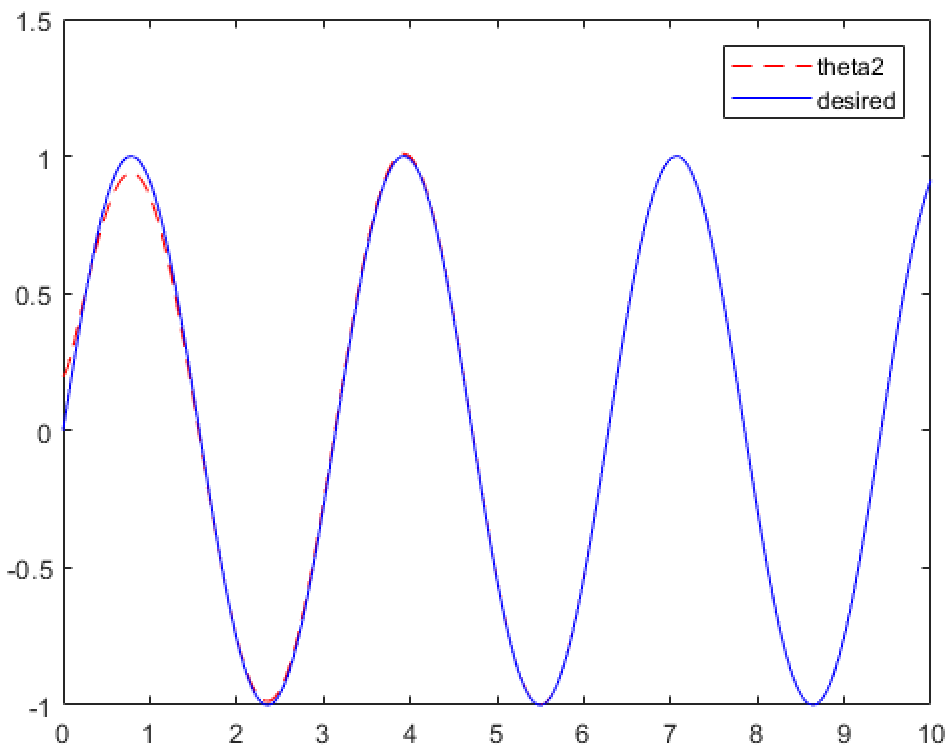
```
function [] = gravity_arm()

clc
clear all;
close all;

% the following parameters for the arm
I1=10;  I2 = 10; m1=5; r1=.5; m2=5; r2=.5; l1=1; l2=1;

% we compute the parameters in the dynamic model
a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
d = I2+ m2*r2^2;

global y;
y = 0;

% initial condition
x0= [-0.5,0.2,0.1,0.1];
%x0= [-0.8, 0.5, 0.1, 0.1];
tf =10;
w =0.2;
beta = 0.1;
g = 9.81;

global torque
torque =[];

global tor;
tor = [0;0];
```

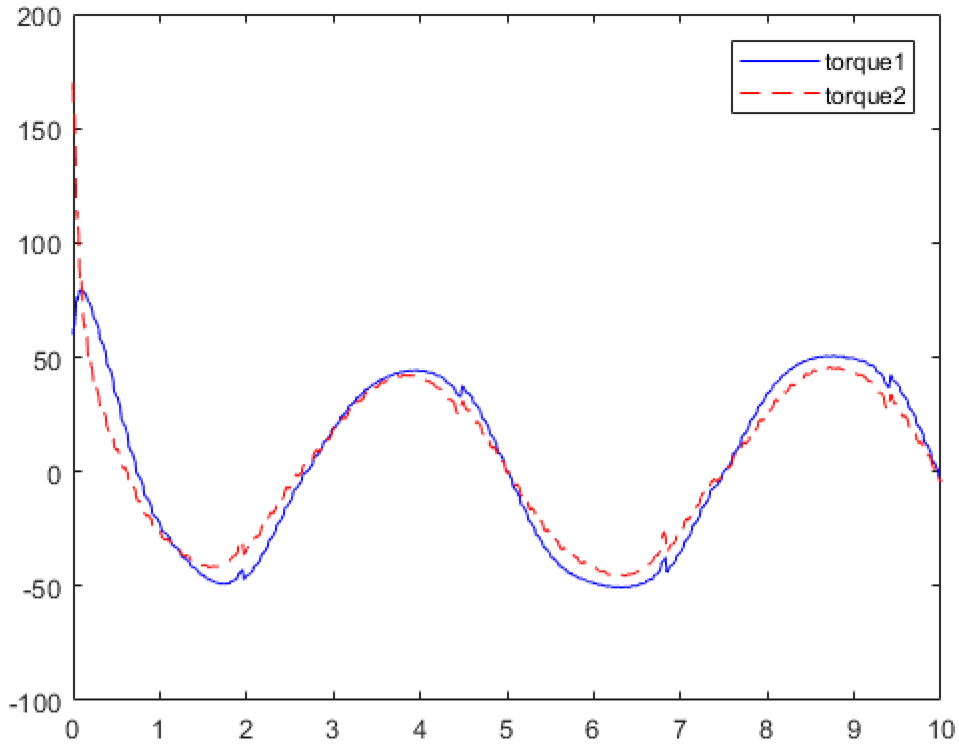
Implement the Augmented PD control.

```
options = odeset('RelTol',1e-4,'AbsTol',[1e-4, 1e-4, 1e-4, 1e-4]);
[T,X] = ode45(@(t,x) Augmented_PD_control(t,x),[0 tf],x0, options);

figure('Name','Theta_1 under Augmented PD control');
plot(T, X(:,1),'r--');
hold on
plot(T, w*ones(size(T,1),1),'b-');
legend('theta1', 'desired')
figure('Name','Theta_2 under Augmented PD control');
plot(T, X(:,2),'r--');
hold on
plot(T, sin(2*T),'b-');
legend('theta2', 'desired')

figure('Name', 'I/p- Augmented PD control')
plot(T, torque(1,1:size(T,1)), 'b-');
hold on
plot(T, torque(2,1:size(T,1)), 'r--');
legend('torque1', 'torque2')
```

```
torque = [];

% The function -Augmented PD control

function dx = Augmented_PD_control(t,x)
theta_d = [w;sin(2*t)];
dtheta_d = [0;2*cos(2*t)];
ddtheta_d = [0;-4*sin(2*t)];
theta = x(1:2,1);
dtheta= x(3:4,1);
des_x = [theta_d ; dtheta_d];


global M C
M = [a+2*b*cos(x(2)), d+b*cos(x(2)); d+b*cos(x(2)), d];
C = [-b*sin(x(2))* x(4), -b*sin(x(2))*(x(3)+ x(4));
 b*sin(x(2))*x(3),0];
invM = inv(M);
invMC= inv(M)*C;

tau = AugPDControl(theta_d, dtheta_d, ddtheta_d, theta, dtheta);
torque = [torque , tau];
dx = zeros(4,1);
dx(1) = x(3);
dx(2) = x(4);
dx(3:4) = -invMC *x(3:4) + invM*tau;

end

function tau = AugPDControl(theta_d, dtheta_d, ddtheta_d, theta,
 dtheta)
global M C
Kp = 100*eye(2);
Kv = 100*eye(2);
e = theta_d - theta;
de = dtheta_d -dtheta;
tau = (Kp*e + Kv*de) + C*dtheta_d + M*ddtheta_d;
end
```


theta1
desired


theta2
desired

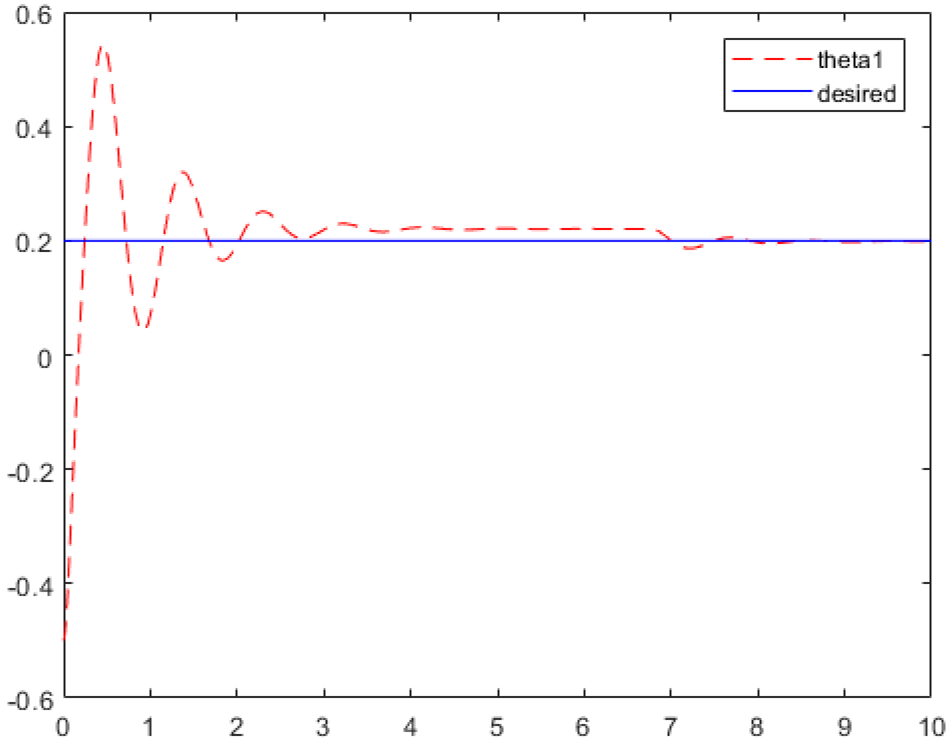


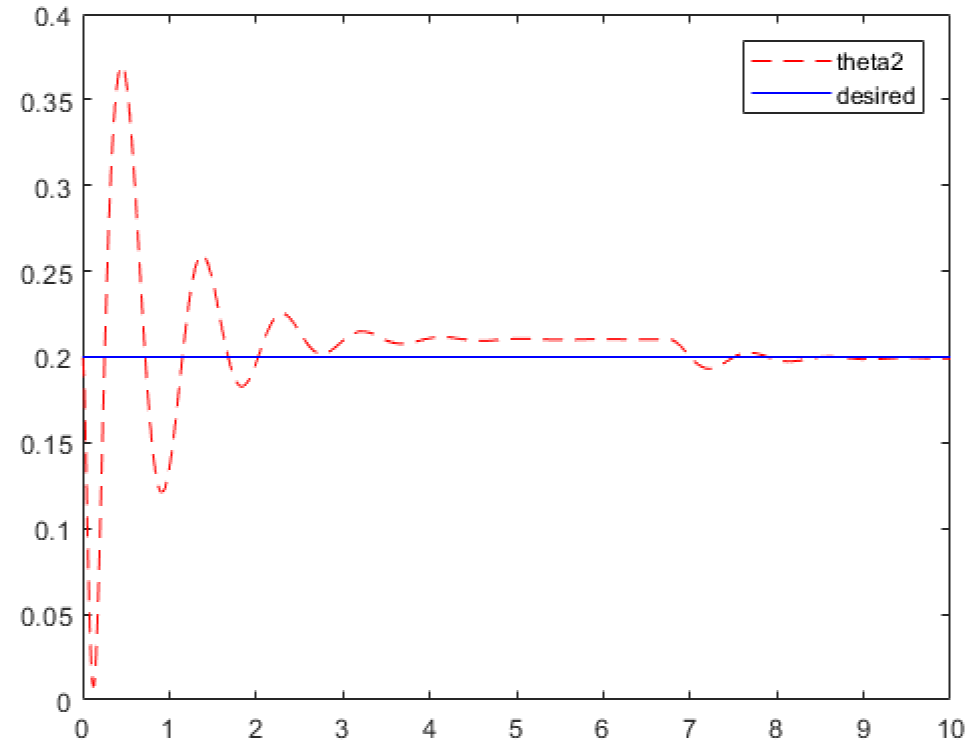
Implement the iterative learning control.

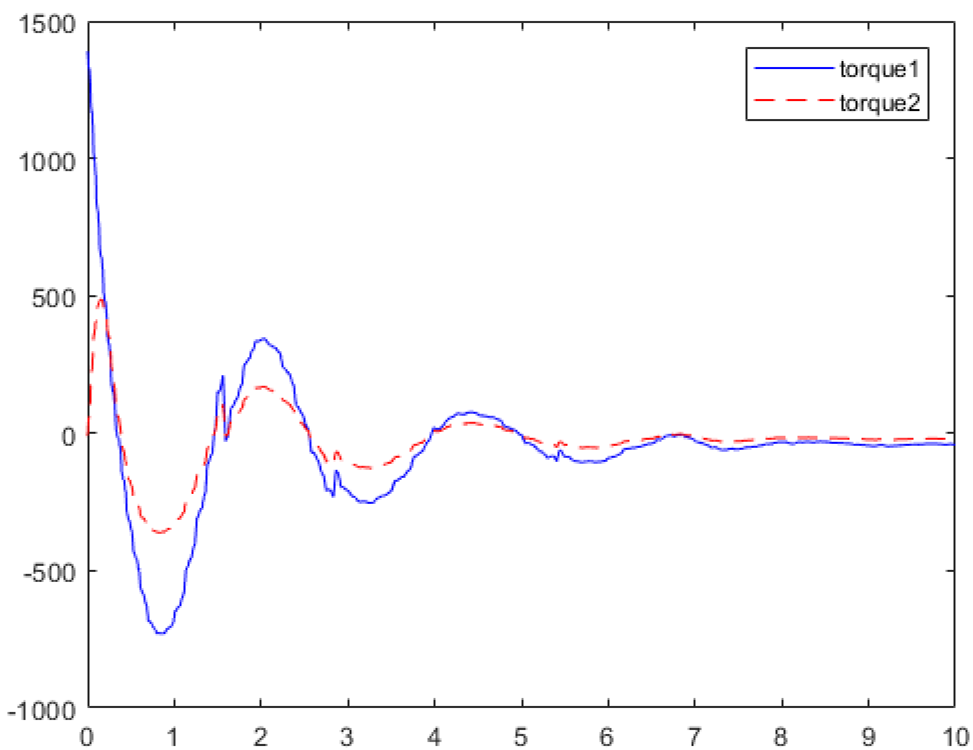
```
options = odeset('RelTol',1e-4,'AbsTol',[1e-4, 1e-4, 1e-4, 1e-4]);
[T,X] = ode45(@(t,x) Iterative_learning_control(t,x),[0 tf],x0,
 options);

figure('Name','Theta_1 under Iterative learning control');
plot(T, X(:,1),'r--');
hold on
plot(T, w*ones(size(T,1),1),'b-');
legend('theta1', 'desired')
figure('Name','Theta_2 under Iterative learning control');
plot(T, X(:,2),'r--');
hold on
plot(T, w*ones(size(T,1),1),'b-');
legend('theta2', 'desired')
%plot(T, sin(2*T),'b-');

figure('Name', 'I/p- Iterative learning control')
plot(T, torque(1,1:size(T,1)), 'b-');
hold on
plot(T, torque(2,1:size(T,1)), 'r--');
legend('torque1', 'torque2')
%torque = [];
```

```
% The function -Iterative learning control

function dx = Iterative_learning_control(t,x)
theta_d = [0.2;0.2];
dtheta_d = [0;0];
ddtheta_d = [0;0];
theta = x(1:2,1);
dtheta= x(3:4,1);


M = [a+2*b*cos(x(2)), d+b*cos(x(2)); d+b*cos(x(2)), d];
C = [-b*sin(x(2))*x(4), -b*sin(x(2))*(x(3)+ x(4));
 b*sin(x(2))*x(3),0];
invM = inv(M);
invMC= inv(M)*C;

% Gravity Matrix
g1=-(m1+m2)*g*l1*sin(x(2))-m2*g*l2*sin(x(1)+ x(2));
g2=-m2*g*l2*sin(x(1)+ x(2));
Gq=[g1;g2];

%global tor
tau_r = IterativeLearningControl(theta_d, dtheta_d, ddtheta_d, theta,
 dtheta,t);
if (sum(isinf(tor)) ~= 0)
    disp('Inf_err')
end
torque = [torque , tau_r];

dx = zeros(4,1);
dx(1) = x(3);
dx(2) = x(4);
dx(3:4) = -invMC*x(3:4) + invM*tau_r - invM*Gq  ;
if (sum(isinf(dx)) ~= 0)
    disp('Inf_err')
end

end

function tau_ret = IterativeLearningControl(theta_d, dtheta_d,
 ddtheta_d, theta, dtheta,t)
% global tor
Kp = 200 * eye(2);
Kv = 10 * eye(2);
e = theta - theta_d;
de = dtheta - dtheta_d;
if t==0
    tor=[0;0];
end
tau_ret = ((1/beta)*(-Kp*e - Kv*de)) + tor;
if norm(dtheta)<0.0001
    tor = tau_ret;
end
```

```
end
```




theta2
desired
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0 1 2 3 4 5 6 7 8 9 10


torque1
torque2
1500
1000
500
0
-500
-1000
0 1 2 3 4 5 6 7 8 9 10

```
end
```

Published with MATLAB® R2017b

Explanation:-

The dynamic Model of the system

$$\cancel{M(\vec{q},\dot{\vec{q}})\ddot{\vec{q}}}$$

$$M(\vec{q})\ddot{\vec{q}} + V(\vec{q},\dot{\vec{q}})\dot{\vec{q}} + G(\vec{q}) = \tau$$

for system to converge

K_p gain (proportional gain) shall be high enough.

i.e., $\lambda_{min}(K_p) \geq \alpha$

where

$$\left\| \frac{\partial G(\vec{q})}{\partial(\vec{q})} \right\| \leq \alpha$$

Minimum eigen value for K_p shall be greater than or equal to the partial derivative of Gravity term w.r.t the joint angles.

Hence, I seperately computed the α value and delibrately chose K_p value higher in my code.