```
%---------------------------------------------------%
% Qual Exam Q5:                                     %
% December 2014 Dr Cobb                 %
% Timothy Coon                                      %
%---------------------------------------------------%
% The problem solved here is given as follows:      %
%---------------------------------------------------%

clear;
close all; clc;

% Geometry
R = 0.5;        % (m) mirror radius
auxdata.f = R/2;        % (m) mirror focal length
auxdata.R1 = 2*R/pi;
auxdata.R2 = R-2*R/pi;
auxdata.rho = norm([auxdata.R1,auxdata.R2]);
rho1 = [-auxdata.R2; auxdata.R1];
rho2 = [auxdata.R1; -auxdata.R2];
phi1 = atan(auxdata.R2/auxdata.R1);
phi2 = atan(auxdata.R1/auxdata.R2);
auxdata.cos_phi1 = cos(phi1);
auxdata.sin_phi1 = sin(phi1);
auxdata.cos_phi2 = cos(phi2);
auxdata.sin_phi2 = sin(phi2);

% Dynamics parameters
auxdata.I = 1;          % (kg-m/s^2) mirror MOI
auxdata.m = 1;          % (kg) mirror mass
auxdata.c = 10;         % (N/m/s) damping coefficient
auxdata.k = 50;         % (N/m) spring rate
auxdata.r0 = 0.5;       % (N) mean of input disturbance
auxdata.sigma = 0.1;    % (N) std of input disturbance

% time boundary conditions
t0 = 0;                       % (s) force start at time = zero
tf = 5;                       % (s) fixed simulation time

% state limits
xmin  = -1e-2; xmax = 1e-2;     % (m) min/max CoM x-pos change
xdmin = -1; xdmax = 1;       % (m/s) min/max CoM x-vel
ymin  = -1e-2; ymax = 1e-2;     % (m) min/max CoM y-pos change
ydmin = -1; ydmax = 1;       % (m/s) min/max CoM y-vel
```

```
amin  = -1e-2; amax = 1e-2;     % (rad) min/max alpha change
admin = -1; admax = 1;       % (rad/s) min/max alpha change rate
uxmin = -1e-1; uxmax = 1e-1;          % (m) x-pos control actuator
uymin = -1e-1; uymax = 1e-1;          % (m) y-pos control actuator

% control limits
uxdmin = -1e-1; uxdmax = 1e-1;          % (m) x-vel control actuator
uydmin = -1e-1; uydmax = 1e-1;          % (m) y-vel control actuator

% state boundary conditions
x0 = 0;                     % (m)
xd0 = 0;                    % (m/s)
y0 = 0;                     % (m)
yd0 = 0;                    % (m/s)
a0 = 0;                     % (rad)
ad0 = 0;                    % (rad/s)
ux0 = 0;                    % (m)
uy0 = 0;                    % (m)

%-------------------------------------------------------------------------%
%----------------------- Setup for Problem Bounds ------------------------%
%-------------------------------------------------------------------------%
iphase = 1;
% time
bounds.phase.initialtime.lower = t0;
bounds.phase.initialtime.upper = t0;
bounds.phase.finaltime.lower = tf;
bounds.phase.finaltime.upper = tf;

% parameters
% bounds.parameter.lower =

% states
% fixed initial state
bounds.phase.initialstate.lower = [x0,xd0,y0,yd0,a0,ad0,ux0,uy0];
bounds.phase.initialstate.upper = [x0,xd0,y0,yd0,a0,ad0,ux0,uy0];
% bounds for states after initial
bounds.phase.state.lower = [xmin,xdmin,ymin,ydmin,amin,admin,uxmin,uymin];
bounds.phase.state.upper = [ymax,admax,ymax,ydmax,amax,admax,uxmax,uymax];
% free final state
bounds.phase.finalstate.lower = [xmin,xdmin,ymin,ydmin,amin,admin,uxmin,uymin];
bounds.phase.finalstate.upper = [ymax,admax,ymax,ydmax,amax,admax,uxmax,uymax];
```

```
% control limits
bounds.phase.control.lower = [uxdmin, uydmin];
bounds.phase.control.upper = [uxdmax, uydmax];

% path constraints
% bounds.phase.path.lower = 1;
% bounds.phase.path.upper = 1;
% The integral constraints are inequalities. Express this by setting the
% lower bound to be the minimum required, then a guess for the upper
% bound which is NOT ARBITRARILY HIGH, MUST BE "CLOSE".
bounds.phase.integral.lower = [0];
bounds.phase.integral.upper = [5];

%-------------------------------------------------------------------------%
%---------------------- Provide Guess of Solution ------------------------%
%-------------------------------------------------------------------------%
guess.phase.time    = [t0; tf];
guess.phase.state   = [[x0,xd0,y0,yd0,a0,ad0,ux0,uy0];[x0,xd0,y0,yd0,a0,ad0,ux0,uy0]];
guess.phase.control = [[0; 0],[0; 0]];
% the integral guess is to be a row vector (1xnd)
guess.phase.integral = [0];

%-------------------------------------------------------------------------%
%------------- Assemble Information into Problem Structure ---------------%
%-------------------------------------------------------------------------%
setup.name = 'Q5P2-Problem';
setup.functions.continuous = @Q5P2_Continuous;
setup.functions.endpoint = @Q5P2_Endpoint;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;

setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'sparseNaN';

setup.scales.method = 'none';

setup.method = 'RPM-Differentiation';

setup.mesh.method = 'hp-PattersonRao';
setup.mesh.tolerance = 1e-2;
n_cols = 20; n_mesh = 20;
```

```
setup.mesh.phase.fraction = (1/n_mesh)*ones(1,n_mesh);
setup.mesh.phase.colpoints = n_cols*ones(1,n_mesh);
setup.nlp.solver = 'ipopt';
setup.nlp.ipoptoptions.linear_solver = 'mumps';
setup.nlp.ipoptoptions.tolerance = 1e-2;
setup.displaylevel = 2;

%-------------------------------------------------------------------------%
%------------------------- Solve Problem Using GPOPS2 --------------------%
%-------------------------------------------------------------------------%
output = gpops2(setup);
output.result.nlptime
solution = output.result.solution;
GPOPS2_ExitFlags(output)

%% ----------------------------------------------------------------------%
%-------------------------------- Plot Solution --------------------------%
%-------------------------------------------------------------------------%

t      = solution.phase.time;
x      = solution.phase.state(:,1);
xd     = solution.phase.state(:,2);
y      = solution.phase.state(:,3);
yd     = solution.phase.state(:,4);
alpha  = solution.phase.state(:,5);
alphad = solution.phase.state(:,6);
Ux     = solution.phase.state(:,7);
Uy     = solution.phase.state(:,8);
Uxd    = solution.phase.control(:,1);
Uyd    = solution.phase.control(:,2);


% mirror vertex position
x2_nom = rho2(1);
y2_nom = rho2(2);
x2  = x2_nom + x + auxdata.rho*alpha*auxdata.cos_phi2;
y2  = y2_nom + y + auxdata.rho*alpha*auxdata.sin_phi2;

% focal point position
xfoc_nom = x2_nom;
yfoc_nom = y2_nom + auxdata.f;
xfoc = x2 + auxdata.f*sin(alpha);
yfoc = y2 + auxdata.f*cos(alpha);
```

```
figure()
% suptitle('Motion Plots')
subplot(421)
plot(x,y)
title('CoM Motion')
axis equal
subplot(422)
plot(xfoc,yfoc)
% hold on
% tc = linspace(0,2*pi,50);
% xc_m = meanRadius*cos(tc)+xfoc_nom;
% yc_m = meanRadius*sin(tc)+yfoc_nom;
% plot(xc_m,yc_m)
% xc_std1 = (meanRadius - stdRadius)*cos(tc) + xfoc_nom;
% yc_std1 = (meanRadius - stdRadius)*sin(tc) + yfoc_nom;
% xc_std2 = (meanRadius + stdRadius)*cos(tc) + xfoc_nom;
% yc_std2 = (meanRadius + stdRadius)*sin(tc) + yfoc_nom;
% plot(xc_std1,yc_std1,'--k',xc_std2,yc_std2,'--k')
% hold off
% xlim([xfoc_nom-2*meanRadius, xfoc_nom+2*meanRadius]);
% ylim([yfoc_nom-2*meanRadius, yfoc_nom+2*meanRadius]);
axis equal
% legend('Pos','Mean','Std')
title('Focal Point Motion')
subplot(423)
plot(t,x,t,y)
legend('x','y')
title('CoM Linear Disp')
subplot(424)
plot(t,alpha)
title('Angular Disp')
subplot(425)
plot(t,Ux,t,Uy)
legend('Ux','Uy')
title('Actuator Length')
subplot(426)
plot(t,Uxd,t,Uyd)
legend('Uxd','Uyd')
title('Actuator Velocity')
subplot(4,2,7:8)
plot(t,sin(10*t),t,sin(5*t))
legend('Dx','Dy')
```

```
title('Disturbance Force')


% arrow setup
% Start = [x,y];
% Stop  = [xfoc,yfoc];
%
% figure()
% arrow(Start,Stop,'Length',0.3,'BaseAngle',15,'TipAngle',25)
% axis square; axis equal
```