EML 6934-Optimal Control Course Project

Submitted to: Dr.Anil V. Rao, MAE Department, University of Florida

Submitted by: Prajwal Gowdru Shanthamurthy

December 10, 2019

0.1 Elementary Optimal Control Problem

Linear Tangent Steering Problem

Minimize the cost functional : $J = t_f$ subject to the dynamic constraints:

$$\begin{aligned}
\dot{x_1} &= x_3 \\
\dot{x_2} &= x_4 \\
\dot{x_3} &= a\cos(u) \\
\dot{x_4} &= a\sin(u)
\end{aligned}$$

and the boundary conditions:

$$\begin{aligned}
x_1(0) &= 0 & \qquad x_1(t_f) &= \text{Free} \\
x_2(0) &= 0 & \qquad x_2(t_f) &= 5 \\
x_3(0) &= 0 & \qquad x_3(t_f) &= 45 \\
x_4(0) &= 0 & \qquad x_4(t_f) &= 0
\end{aligned}$$

Solution using Indirect Shooting:

Lagrange Cost, $L = 0$

Mayer Cost, $M = t_f$

Let the state vector $\boldsymbol{X} \triangleq \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ and $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}) \triangleq \begin{bmatrix} x_3 \\ x_4 \\ a\cos(u) \\ a\sin(u) \end{bmatrix}$

Let the costate vector be $\boldsymbol{\Lambda} \triangleq \begin{bmatrix} \lambda_{x_1} \\ \lambda_{x_2} \\ \lambda_{x_3} \\ \lambda_{x_4} \end{bmatrix}$. So, the Hamiltonian, H, is defined as $H = L + \boldsymbol{\lambda^T f}$.

Using first-order optimality conditions:

$$\begin{aligned}
\frac{\partial H}{\partial u} = a\lambda_{x_4}\cos(u) - a\lambda_{x_3} sin(u) &= 0 \\
\Rightarrow \tan(u) &= \frac{\lambda_{x_4}}{\lambda_{x_3}}
\end{aligned}$$

So, u must be computed as $u = atan2(\lambda_{x_4}, \lambda_{x_3})$ or $u = atan2(-\lambda_{x_4}, -\lambda_{x_3})$ depending on whether the cost is being minimized or maximized. For our solution :

$$u = atan2(-\lambda_{x_4}, -\lambda_{x_3})$$

The costate dynamics is given by:

$$\dot{\mathbf{\Lambda}} = -\left(\frac{\partial H}{\partial \boldsymbol{X}}\right)^T = \begin{bmatrix} 0 \\ 0 \\ -\lambda_{x_1} \\ -\lambda_{x_2} \end{bmatrix}$$

Transversality Conditions:

t_f is free.

$$\Rightarrow \frac{\partial M}{\partial t_f} - \boldsymbol{\nu^T} \frac{\partial \boldsymbol{b}}{\partial t_f} + H(t_f) = 0$$

where $\boldsymbol{b} = \mathbf{0}$ is the set of boundary conditions. This yields us

$$H(t_f) = -1$$

The hamiltonian does not depend on time explicitly. Therefore $H(t) \equiv -1$.

$x_1(t_f)$ is free.

$$\Rightarrow \frac{\partial M}{\partial x_1(t_f)} + \boldsymbol{\nu^T} \frac{\partial \boldsymbol{b}}{\partial x_1(t_f)} - \lambda_{x_1}(t_f) = 0$$

which yields us:

$$\lambda_{x_1}(t_f) = 0$$

The rest of the transversality conditions do not apply to this problem.

Two-Point Boundary Value Problem:

So, using all these, the two-point boundary value problem to solve would be:

For $\boldsymbol{P} \triangleq \begin{bmatrix} \boldsymbol{X} \\ \mathbf{\Lambda} \end{bmatrix}$, subject to the dynamics:

$$\dot{\boldsymbol{P}} = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{X}) \\ -\left(\frac{\partial H}{\partial \boldsymbol{X}}\right)^T \end{bmatrix}$$

and the **boundary conditions** as given by the boundary conditions on the states given in the problem statement along with the transversality conditions on the costates arising out of the first-order optimality conditions.

MainScript Linear Tangent Steering Problem _ Indirect shooting Method

```
clc; clear;

%Defining the constants in the problem
a=100;

%Initial State
X0=[0;0;0;0];

%Guess for the costates
guessedlambdas=[0;1;1;1];

%Guess for the final time
tfguess= 4;

%Overall Guess vector
Guess=[guessedlambdas;tfguess];

% Iterative scheme using fsolve to find the correct guess
[CorrectGuess,fval,exitflag,output]=fsolve(@(Guess)error_bc_lineartgt(Guess,X0),Guess);

% Getting the states, costates and controls as functions of time  by numerical integration using the correct guess
tf_optimal=CorrectGuess(end);
lambda0=CorrectGuess(1:end-1);
tspan=[0 tf_optimal];
P0=[X0;lambda0];
options = odeset('AbsTol',1e-10,'RelTol',1e-10);
[time,P_Sol]=ode113(@ode_lineartgt,tspan,P0,options);
X1=P_Sol(:,1); X2=P_Sol(:,2); X3=P_Sol(:,3); X4=P_Sol(:,4);
Lambda1=P_Sol(:,5); Lambda2=P_Sol(:,6); Lambda3=P_Sol(:,7); Lambda4=P_Sol(:,8);
U=atan2(-Lambda4,-Lambda3);

%Computing Hamiltonian along the solution
Hamiltonian = Lambda1.*X3 + Lambda2.*X4 + a*Lambda3.*cos(U) + a*Lambda4.*sin(U);

%Plotting
plots_lineartgt(time,X1,X2,X3,X4,Lambda1,Lambda2,Lambda3,Lambda4,U,Hamiltonian)
```

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
```

Functions _ Linear Tangent Steering Problem _ Indirect shooting Method

```
%%%%%Function to find the error in boundary conditions
function errvec=error_bc_lineartgt(Guess,X0)
a=100;
tfguess=Guess(end);
lambda0=Guess(1:end-1);
P0=[X0;lambda0];
tspan=[0 tfguess];

%Numerical Integration over tspan
options = odeset('AbsTol',1e-10,'RelTol',1e-10);
[~,P_Sol]=ode113(@ode_lineartgt,tspan,P0,options);
P_tf=P_Sol(end,:)';

% The states and costates appearing in the boundary conditions at the final time
x2=P_tf(2); x3=P_tf(3); x4=P_tf(4);
lambda1=P_tf(5); lambda2=P_tf(6); lambda3=P_tf(7); lambda4=P_tf(8);

%CONTROL
u=atan2(-lambda4,-lambda3);
H=lambda1*x3 + lambda2*x4 + a*lambda3*cos(u) + a*lambda4*sin(u);

% Computing error in the final boundary conditions
%(given and the ones arising from Transversality conditions conditions)
%Final boundary conditions
lambda1f=0;
x2f=5;
x3f=45;
x4f=0;
Hf=-1;

%Error
errvec=[lambda1-lambda1f; x2-x2f; x3-x3f; x4-x4f; H-Hf];
end

%%%%%ODE function
function Pdot=ode_lineartgt(~,P)
a=100;
x3=P(3); x4=P(4);
lambda1=P(5); lambda2=P(6); lambda3=P(7); lambda4=P(8);

%Control as given by the optimality conditions:
u=atan2(-lambda4,-lambda3);

Pdot(1,1)= x3;
Pdot(2,1)= x4;
Pdot(3,1)= a*cos(u);
Pdot(4,1)= a*sin(u);
Pdot(5,1)= 0;
Pdot(6,1)= 0;
Pdot(7,1)= -lambda1;
Pdot(8,1)= -lambda2;
end

%%%%%Function for plotting
function plots_lineartgt(time,X1,X2,X3,X4,Lambda1,Lambda2,Lambda3,Lambda4,U,Hamiltonian)
% Plots
close all;

figure;
subplot(2,2,1); plot(time,X1,'r',time,X2,'b',time,X3,'k',time,X4,'g','LineWidth',1.5);
xlabel('Time'); ylabel('States'); title('Optimal State Trajectory'); grid on;
xlim([0 time(end)]); ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4;
leg1=legend('$x_1$','$x_2$','$x_3$','$x_4$'); set(leg1,'Interpreter','latex'); hold on;

subplot(2,2,2); plot(time,Lambda1,'r',time,Lambda2,'b',time,Lambda3,'k',time,Lambda4,'g','LineWidth',1.5);
xlabel('Time'); ylabel('Costates'); title('Optimal Costate Trajectory'); grid on;
xlim([0 time(end)]); ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4;
leg2=legend('$\lambda_1$','$\lambda_2$','$\lambda_3$','$\lambda_4$'); set(leg2,'Interpreter','latex'); hold on;

subplot(2,2,3); plot(time,U,'b','LineWidth',1.5);
xlabel('Time'); ylabel('Control'); title('Control as a function of time'); grid on;
xlim([0 time(end)]); ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4;
leg3=legend('$u$'); set(leg3,'Interpreter','latex'); hold on;

subplot(2,2,4); plot(X1,X2,'b','LineWidth',1.5);
title('Path taken by the particle in the XY-plane'); grid on;
xlim([0 max(X1)]); ax.GridLineStyle = ':'; ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; hold on;

figure; plot(time,Hamiltonian,'b','LineWidth',1.5);
xlabel('Time'); ylabel('Hamiltonian'); title('Hamiltonian along the solution'); grid on;
xlim([0 time(end)]); ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4;
leg3=legend('$H$'); set(leg3,'Interpreter','latex'); hold on;
```

Main Script Linear Tangent Steering Problem - Collocation Method

```
%-------------------------------------------------------------------%
%            Linear Tangent Steering Problem                        %
%-------------------------------------------------------------------%
% Solve the following optimal control problem:                      %
% Minimize t_f                                                      %
% subject to the differential equation constraints                  %
% dx1/dt= x3                                                        %
% dx2/dt= x4                                                        %
% dx3/dt= a*cos(u)
% dx4/dt= a*sin(u)
%-------------------------------------------------------------------%
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                %
%-------------------------------------------------------------------%
global a psStuff nstates ncontrols
global iGfun jGvar
%-------------------------------------------------------------------%
% END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                %
%-------------------------------------------------------------------%


%-------------------------------------------------------------------%
%             Define the constants for the problem                  %
%-------------------------------------------------------------------%
a=100;

%-------------------------------------------------------------------%
%  Define the sizes of quantities in the optimal control problem  %
%-------------------------------------------------------------------%
nstates = 4;
ncontrols = 1;

%-------------------------------------------------------------------%
%  Define bounds on the variables in the optimal control problem  %
%-------------------------------------------------------------------%
x10=0;  x20=0; x30=0; x40=0;
x2f=5; x3f=45; x4f=0;
x1min = -20; x1max = 50;
x2min = -20; x2max = 20;
x3min = -20; x3max = 50;
x4min = -20; x4max = 20;
umin = -pi;  umax = pi;
t0min = 0; t0max = 0;
tfmin = 0; tfmax = 10;

%-------------------------------------------------------------------%
% In this section, we define the three type of discretizations    %
% that can be employed.  These three approaches are as follows:   %
%    (1) p-method  = global pseudospectral method                 %
%                  = single interval and the degree of the        %
%                    polynomial in the interval can be varied     %
%    (2) h-method  = fixed-degree polynomial in each interval     %
%                    and the number of intervals can be varied    %
%    (3) hp-method = can vary BOTH the degree of the polynomial   %
%                    in each interval and the number of intervals %
%                                                                 %
% For simplicity in this tutorial, we will allow for either a     %
```

```
% p-method or an h-method.  Regardless of which method is being   %
% employed, the user needs to specify the following parameters:   %
%    (a) N = Polynomial Degree                                    %
%    (b) meshPoints = Set of Monotonically Increasing Mesh Points %
%                     on the Interval $\tau\in[-1,+1]$.           %
%                                                                  %
% When using a p-method, the parameters N and meshPoints must be  %
% specified as follows:                                           %
%    (i)  meshPoints = [-1 1]                                     %
%    (ii) N = Choice of Polynomial Degree (e.g., N=10, N=20)      %
% When using an h-method, the parameters N and meshPoints must be %
% specified as follows:                                           %
%    (i)  meshPoints = $[\tau_1,\tau_2,\tau_3,\ldots,\tau_N]$     %
%                      where $\tau_1 = -1$, $\tau_N = 1$ and      %
%                      (\tau_2,\ldots,\tau_{N-1}) are             %
%                      monotonically increasing on the open       %
%                      interval $(-1,+1)$.                        %
%-----------------------------------------------------------------%
%      Compute Points, Weights, and Differentiation Matrix        %
%-----------------------------------------------------------------%
%-----------------------------------------------------------------%
% Choose Polynomial Degree and Number of Mesh Intervals           %
% numIntervals = 1 ===> p-method                                  %
% numIntervals > 1 ===> h-method                                  %
%-----------------------------------------------------------------%
N =8;
numIntervals = 20;
%-----------------------------------------------------------------%
% DO NOT ALTER THE LINE OF CODE SHOWN BELOW!                      %
%-----------------------------------------------------------------%
meshPoints = linspace(-1,1,numIntervals+1).';
polyDegrees = N*ones(numIntervals,1);
[tau,w,D] = lgrPS(meshPoints,polyDegrees);
psStuff.tau = tau; psStuff.w = w; psStuff.D = D; NLGR = length(w);
%-----------------------------------------------------------------%
% DO NOT ALTER THE LINES OF CODE SHOWN ABOVE!                     %
%-----------------------------------------------------------------%

%-----------------------------------------------------------------%
% Set the bounds on the variables in the NLP.                     %
%-----------------------------------------------------------------%
zx1min = x1min*ones(length(tau),1);
zx1max = x1max*ones(length(tau),1);
zx1min(1) = x10;       zx1max(1) = x10;

zx2min = x2min*ones(length(tau),1);
zx2max = x2max*ones(length(tau),1);
zx2min(1) = x20;       zx2max(1) = x20;
zx2min(NLGR+1) = x2f;  zx2max(NLGR+1) = x2f;

zx3min = x3min*ones(length(tau),1);
zx3max = x3max*ones(length(tau),1);
zx3min(1) = x30;        zx3max(1) = x30;
zx3min(NLGR+1) = x3f;   zx3max(NLGR+1) = x3f;

zx4min = x4min*ones(length(tau),1);
zx4max = x4max*ones(length(tau),1);
zx4min(1) = x40;       zx4max(1) = x40;
zx4min(NLGR+1) = x4f;  zx4max(NLGR+1) = x4f;

zumin = umin*ones(length(tau)-1,1);
```

```
zumax = umax*ones(length(tau)-1,1);

zmin = [zx1min; zx2min; zx3min; zx4min; zumin; t0min; tfmin];
zmax = [zx1max; zx2max; zx3max; zx4max; zumax; t0max; tfmax];

%--------------------------------------------------------------------%
% Set the bounds on the constraints in the NLP.                      %
%--------------------------------------------------------------------%
defectMin = zeros(nstates*(length(tau)-1),1);
defectMax = zeros(nstates*(length(tau)-1),1);
pathMin = []; pathMax = [];
eventMin = []; eventMax = [];
Fmin = [ defectMin; pathMin; eventMin];
Fmax = [ defectMax; pathMax; eventMax];

%--------------------------------------------------------------------%
% Supply an initial guess for the NLP.                               %
%--------------------------------------------------------------------%
x1guess = linspace(x10,x10,NLGR+1).';
x2guess = linspace(x20,x2f,NLGR+1).';
x3guess = linspace(x30,x3f,NLGR+1).';
x4guess = linspace(x40,x4f,NLGR+1).';
uguess = linspace(0,0,NLGR).';
t0guess = 0;
tfguess = 15;
z0 = [x1guess; x2guess; x3guess; x4guess; uguess; t0guess; tfguess];

%--------------------------------------------------------------------%
% Generate derivatives and sparsity pattern using Adigator           %
%--------------------------------------------------------------------%
% - Constraint Funtction Derivatives
xsize  = size(z0);
x      = adigatorCreateDerivInput(xsize,'z0');
output = adigatorGenJacFile('tgtsteerFun',{x});
S_jac  = output.JacobianStructure;
[iGfun,jGvar] = find(S_jac);

% - Objective Funtcion Derivatives
xsize  = size(z0);
x      = adigatorCreateDerivInput(xsize,'z0');
output = adigatorGenJacFile('tgtsteerObj',{x});
grd_structure = output.JacobianStructure;

%--------------------------------------------------------------------%
% Set IPOPT callback functions
%--------------------------------------------------------------------%
funcs.objective   = @(Z)tgtsteerObj(Z);
funcs.gradient    = @(Z)tgtsteerGrd(Z);
funcs.constraints = @(Z)tgtsteerCon(Z);
funcs.jacobian    = @(Z)tgtsteerJac(Z);
funcs.jacobianstructure = @()tgtsteerJacPat(S_jac);
options.ipopt.hessian_approximation = 'limited-memory';

%--------------------------------------------------------------------%
% Set IPOPT Options %
%--------------------------------------------------------------------%
options.ipopt.tol = 1e-5;
options.ipopt.linear_solver = 'ma57';
options.ipopt.max_iter = 2000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
```

```
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['robotArm','IPOPTinfo.txt']; % print output file
options.ipopt.print_level = 5; % set print level default

options.lb = zmin; % Lower bound on the variables.
options.ub = zmax; % Upper bound on the variables.
options.cl = Fmin; % Lower bounds on the constraint functions.
options.cu = Fmax; % Upper bounds on the constraint functions.

%------------------------------------------------------------------%
% Call IPOPT
%------------------------------------------------------------------%
[z, info] = ipopt(z0,funcs,options);

%------------------------------------------------------------------%
% extract lagrange multipliers from ipopt output, info
%------------------------------------------------------------------%
Fmul = info.lambda;

%------------------------------------------------------------------%
% Extract the state and control from the decision vector z.       %
% Remember that the state is approximated at the LGR points       %
% plus the final point, while the control is only approximated    %
% at only the LGR points.                                         %
%------------------------------------------------------------------%
x1 = z(1:NLGR+1);
x2 = z(NLGR+2:2*(NLGR+1));
x3 = z(2*(NLGR+1)+1:3*(NLGR+1));
x4 = z(3*(NLGR+1)+1:4*(NLGR+1));
u = z(4*(NLGR+1)+1:4*(NLGR+1)+NLGR);
t0 = z(end-1);
tf = z(end);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%------------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                   %
% the defect constraints.                                         %
%------------------------------------------------------------------%
multipliersDefects = Fmul(1:nstates*NLGR);
multipliersDefects = reshape(multipliersDefects,NLGR,nstates);
%------------------------------------------------------------------%
% Compute the costates at the LGR points via transformation       %
%------------------------------------------------------------------%
costateLGR = inv(diag(w))*multipliersDefects;
%------------------------------------------------------------------%
% Compute the costate at the tau=+1 via transformation            %
%------------------------------------------------------------------%
costateF = D(:,end).'*multipliersDefects;
%------------------------------------------------------------------%
% Now assemble the costates into a single matrix                  %
%------------------------------------------------------------------%
costate = [costateLGR; costateF];
lamx1 = -costate(:,1); lamx2 = -costate(:,2); lamx3 = -costate(:,3); lamx4 = -costate(:,4);

%Evaluating the Hamiltonian to estimate the closeness of the obtained solution to the true optimal solution%
x1LGR=x1(1:(end-1)); x2LGR=x2(1:(end-1)); x3LGR=x3(1:(end-1)); x4LGR=x4(1:(end-1));
lamx1LGR=lamx1(1:(end-1)); lamx2LGR=lamx2(1:(end-1)); lamx3LGR=lamx3(1:(end-1)); lamx4LGR=lamx4(1:(end-1));

x1dot=x3LGR;
x2dot=x4LGR;
```

```
x3dot=a*cos(u);
x4dot=a*sin(u);
Hamiltonian=lamx1LGR.*x1dot + lamx2LGR.*x2dot + lamx3LGR.*x3dot + lamx4LGR.*x4dot;
%--------------------------------------------------------------------%
% plot results
%--------------------------------------------------------------------%
close all;
lw=1; % Setting Line Width for the Plots

figure;
subplot(2,2,1); plot(t,x1,'-bs','LineWidth',lw);
xlabel('Time'); yl1=ylabel('$x_1$');
set(yl1,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16;  ax.LineWidth = 1.4; xlim([0 max(t)]);
leg1=legend('$x_1$'); set(leg1,'Interpreter','latex'); hold on;

subplot(2,2,2); plot(t,x2,'-bs','LineWidth',lw);
xlabel('Time');yl2=ylabel('$x_2$');
set(yl2,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg2=legend('$x_2$'); set(leg2,'Interpreter','latex'); hold on;

subplot(2,2,3); plot(t,x3,'-bs','LineWidth',lw);
xlabel('Time'); yl3=ylabel('$x_3$');
set(yl3,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg3=legend('$x_3$'); set(leg3,'Interpreter','latex'); hold on;

subplot(2,2,4); plot(t,x4,'-bs','LineWidth',lw);
xlabel('Time'); yl4=ylabel('$x_4$');
set(yl4,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg4=legend('$x_4$'); set(leg4,'Interpreter','latex'); hold on;


figure; plot(tLGR,u,'-bs','LineWidth',lw);
xlabel('Time'); yl7=ylabel('$u$');
set(yl7,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg5=legend('$u_1$'); set(leg5,'Interpreter','latex'); hold on;

figure;
subplot(2,2,1); plot(t,lamx1,'-bs','LineWidth',lw);
xlabel('Time'); yl1=ylabel('$\lambda_{x_1}$');
set(yl1,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16;  ax.LineWidth = 1.4; xlim([0 max(t)]);
leg1=legend('$\lambda_{x_1}$'); set(leg1,'Interpreter','latex'); hold on;

subplot(2,2,2); plot(t,lamx2,'-bs','LineWidth',lw);
xlabel('Time');yl2=ylabel('$\lambda_{x_2}$');
set(yl2,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg2=legend('$\lambda_{x_2}$'); set(leg2,'Interpreter','latex'); hold on;
```

```
subplot(2,2,3); plot(t,lamx3,'-bs','LineWidth',lw);
xlabel('Time'); yl3=ylabel('$\lambda_{x_3}$');
set(yl3,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg3=legend('$\lambda_{x_3}$'); set(leg3,'Interpreter','latex'); hold on;

subplot(2,2,4); plot(t,lamx4,'-bs','LineWidth',lw);
xlabel('Time'); yl4=ylabel('$\lambda_{x_4}$');
set(yl4,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg4=legend('$\lambda_{x_4}$'); set(leg4,'Interpreter','latex'); hold on;


figure; plot(tLGR,u,'-bs','LineWidth',lw);
xlabel('Time'); yl7=ylabel('$u$');
set(yl7,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':';
ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 max(t)]);
leg5=legend('$u_1$'); set(leg5,'Interpreter','latex'); hold on;


figure; plot(tLGR,Hamiltonian,'-bs','LineWidth',lw);
xlabel('Time'); yl9=ylabel('Hamiltonian'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3; xlim([0 max(t)]);
ax.FontSize = 16; ax.LineWidth = 1.4; hold on;
```

```
function C = tgtsteerFun(z)

%---------------------------------------------------------------------%
% Functions for the Robot Arm problem                                 %
% This function is designed to be used with the NLP solver IPOPT      %
%---------------------------------------------------------------------%
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE BELOW!            %
global psStuff nstates ncontrols a
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE!            %
%---------------------------------------------------------------------%

%---------------------------------------------------------------------%
% Radau pseudospectral method quantities required:                    %
%   - Differentiation matrix (psStuff.D)                              %
%   - Legendre-Gauss-Radau weights (psStuff.w)                        %
%   - Legendre-Gauss-Radau points (psStuff.tau)                       %
%---------------------------------------------------------------------%
D = psStuff.D; tau = psStuff.tau; w = psStuff.w;

%---------------------------------------------------------------------%
% Decompose the NLP decision vector into pieces containing            %
%    - the state                                                      %
%    - the control                                                    %
%    - the initial time                                               %
%    - the final time                                                 %
%---------------------------------------------------------------------%
N = length(tau)-1;
stateIndices = 1:nstates*(N+1);
controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
t0Index = controlIndices(end)+1;
tfIndex = t0Index+1;
stateVector = z(stateIndices);
controlVector = z(controlIndices);
t0 = z(t0Index);
tf = z(tfIndex);

%---------------------------------------------------------------------%
% Reshape the state and control parts of the NLP decision vector      %
% to matrices of sizes (N+1) by nstates and (N+1) by ncontrols,       %
% respectively.  The state is approximated at the N LGR points        %
% plus the final point.  Thus, each column of the state vector is     %
% length N+1.  The LEFT-HAND SIDE of the defect constraints, D*X,     %
% uses the state at all of the points (N LGR points plus final        %
% point).  The RIGHT-HAND SIDE of the defect constraints,             %
% (tf-t0)F/2, uses the state and control at only the LGR points.      %
% Thus, it is necessary to extract the state approximations at        %
% only the N LGR points.  Finally, in the Radau pseudospectral        %
% method, the control is approximated at only the N LGR points.       %
%---------------------------------------------------------------------%
statePlusEnd   = reshape(stateVector,N+1,nstates);
control = reshape(controlVector,N,ncontrols);
stateLGR = statePlusEnd(1:end-1,:);

%---------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR.     %
%---------------------------------------------------------------------%
x1 = stateLGR(:,1);
x2 = stateLGR(:,2);
x3 = stateLGR(:,3);
```

```
x4 = stateLGR(:,4);
u = control;

%-------------------------------------------------------------------%
% Compute the right-hand side of the differential equations at    %
% the N LGR points.  Each component of the right-hand side is     %
% stored as a column vector of length N, that is each column has  %
% the form                                                        %
%                   [ f_i(x_1,u_1,t_1) ]                          %
%                   [ f_i(x_2,u_2,t_2) ]                          %
%                           .                                     %
%                           .                                     %
%                           .                                     %
%                   [ f_i(x_N,u_N,t_N) ]                          %
% where "i" is the right-hand side of the ith component of the    %
% vector field f.  It is noted that in MATLABB the calculation of %
% the right-hand side is vectorized.                              %
%-------------------------------------------------------------------%
%Defining some terms in the dynamics:
x1dot=x3;
x2dot=x4;
x3dot=a*cos(u);
x4dot=a*sin(u);
diffeqRHS = [x1dot, x2dot, x3dot, x4dot];

%-------------------------------------------------------------------%
% Compute the left-hand side of the defect constraints, recalling %
% that the left-hand side is computed using the state at the LGR  %
% points PLUS the final point.                                    %
%-------------------------------------------------------------------%
diffeqLHS = D*statePlusEnd;

%-------------------------------------------------------------------%
% Construct the defect constraints at the N LGR points.           %
% Remember that the right-hand side needs to be scaled by the     %
% factor (tf-t0)/2 because the rate of change of the state is     %
% being taken with respect to $\tau\in[-1,+1]$.  Thus, we have    %
% $dt/t\dau=(tf-t0)/2$.                                           %
%-------------------------------------------------------------------%
defects = diffeqLHS-(tf-t0)*diffeqRHS/2;

%-------------------------------------------------------------------%
% Reshape the defect contraints into a column vector.             %
%-------------------------------------------------------------------%
defects = reshape(defects,N*nstates,1);

%-------------------------------------------------------------------%
% Construct the constraint vector.        %
%-------------------------------------------------------------------%
C = defects;
```

```
function constraints = tgtsteerCon(Z)
% computes the constraints
output      = tgtsteerFun(Z);
constraints = output;
end

function grd = tgtsteerGrd(Z)
% computes the gradient
output = tgtsteerObj_Jac(Z);
grd    = output;
end

function jac = tgtsteerJac(Z)
% computes the jacobian
[jac,~] = tgtsteerFun_Jac(Z);
end

function jacpat = tgtsteerJacPat(S_jac)
% computes the jacobian structure
jacpat = S_jac;
end

function obj = tgtsteerObj(z)
% Computes the objective function of the problem
tf = z(end);
% Cost function
J   = tf;
obj = J;
end
```

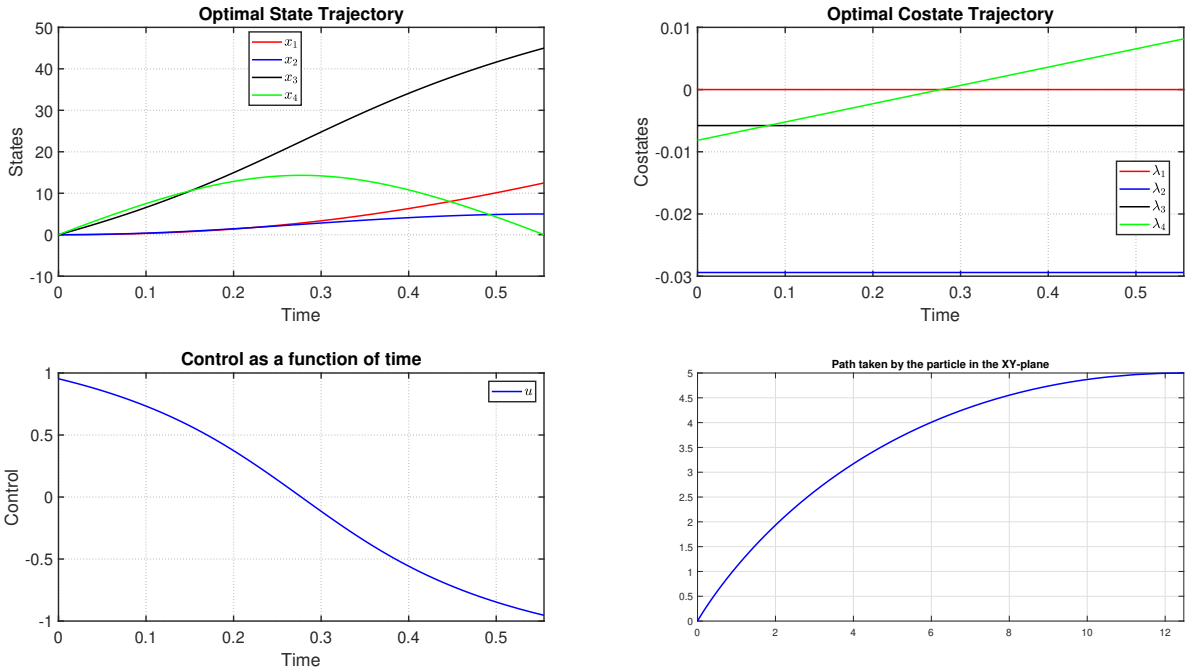
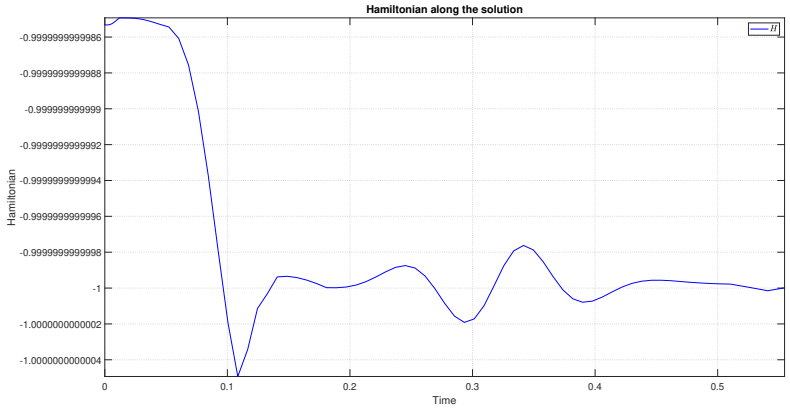


Figure 1: Indirect Shooting: Linear Tangent Steering



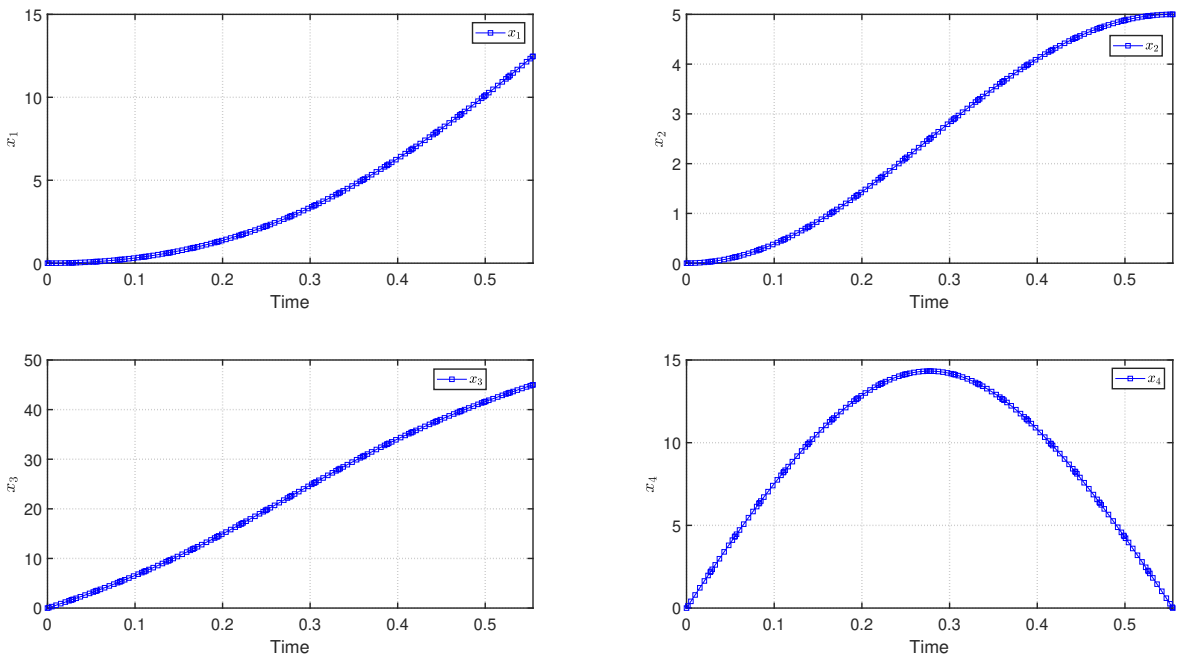
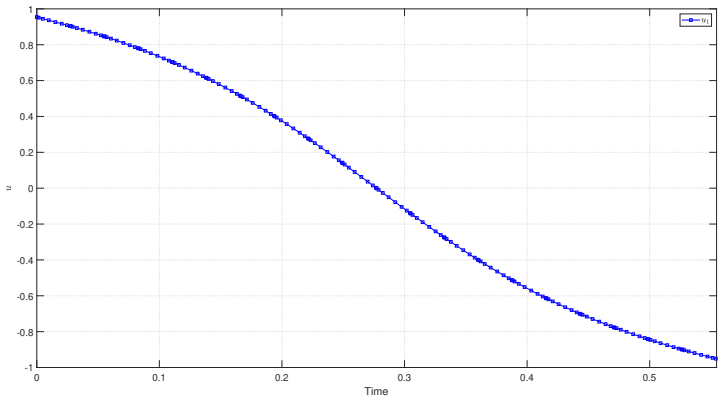


Figure 2: Collocation : Linear Tangent Steering - States and Control



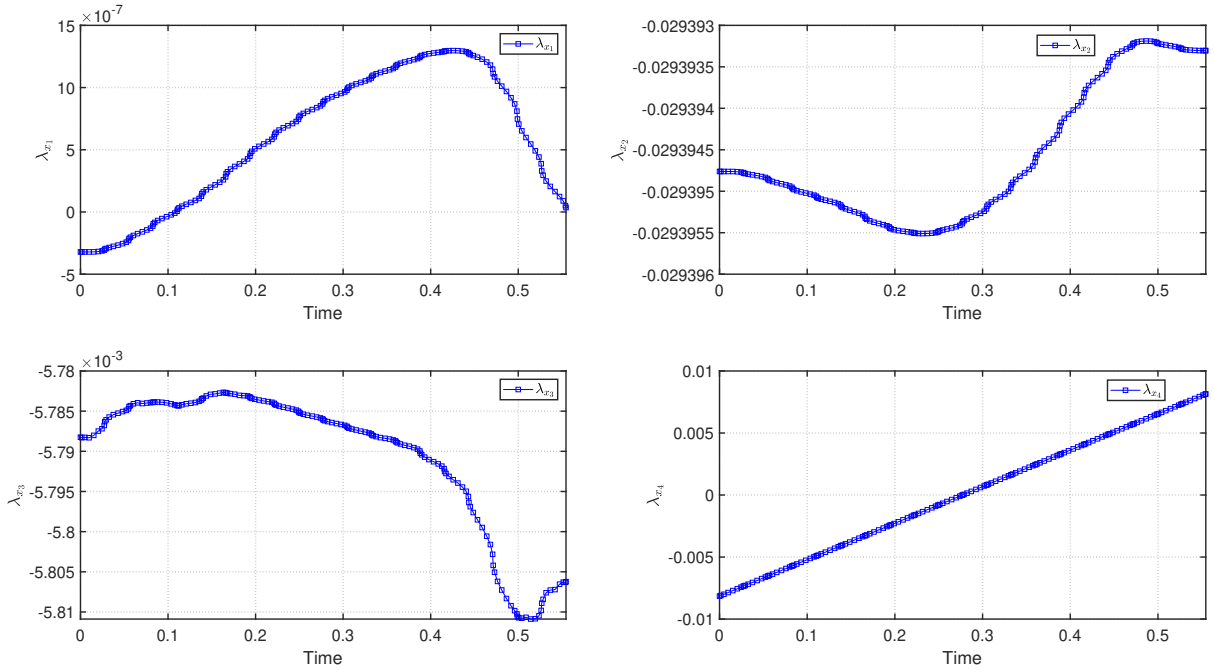
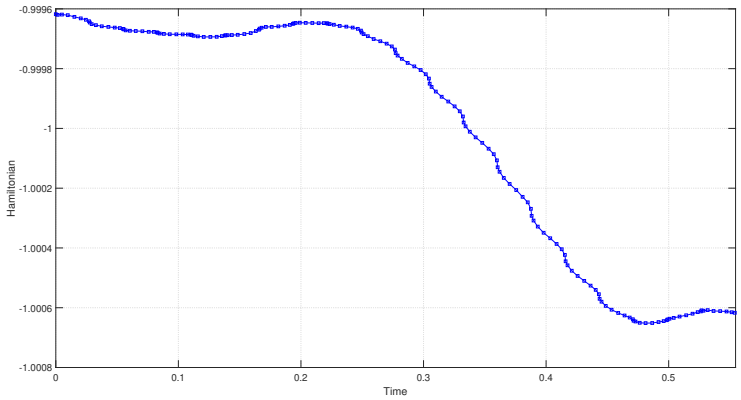


Figure 3: Collocation : Linear Tangent Steering - Costates and Hamiltonian



Comparison between the Indirect solution and the Direct solution (Linear Tangent Steering Problem)

1) Indirect Shooting:

Minimum final time , $t_{f_{optimal}} = 0.5546$

Transversality conditions:

$$\lambda_{x_1}(t_f) = -2.1 \times 10^{-28} \approx 0$$

$H(t) \approx -1 \quad \forall t$ to an accuracy of the order of 10^{-12}

2) Collocation (Direct Method):

For 20 sub-intervals and an 8th degree polynomial approximation within each sub-interval:

Minimum final time , $t_{f_{optimal}} = 0.5546$

Transversality conditions:

$$\lambda_{x_1}(t_f) = 3.5 \times 10^{-8} \approx 0$$

$H(t) \approx -1 \quad \forall t$ to an accuracy of the order of 10^{-3}

So, the solution obtained using the indirect approach looks closer to the "true" optimal solution in terms of satisfaction of the transversality conditions and the time invariance of the Hamiltonian. The collocation technique however achieves convergence even with a far-off initial guess whereas the indirect shooting requires the initial guess on the costates to be close to the true optimal solution's values for convergence to occur.

0.2 Advanced Optimal Control Problem

Robot Arm Problem

Minimize the cost functional : $J = t_f$ subject to the dynamic constraints:

$$\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= u_1/L \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= u_2/I_\theta \\
\dot{x_5} &= x_6 \\
\dot{x_6} &= u_3/I_\phi
\end{aligned}$$

the control inequality constraints

$$|u_i| \leq 1 \quad (i = 1, 2, 3)$$

and the boundary conditions:

$$\begin{aligned}
t_0 &= 0 & t_f &= \text{Free} \\
x_1(0) &= 4.5 & x_1(t_f) &= 4.5 \\
x_2(0) &= 0 & x_2(t_f) &= 0 \\
x_3(0) &= 0 & x_3(t_f) &= 2\pi/3 \\
x_4(0) &= 0 & x_4(t_f) &= 0 \\
x_5(0) &= \pi/4 & x_5(t_f) &= \pi/4 \\
x_6(0) &= 0 & x_6(t_f) &= 0
\end{aligned}$$

where

$$\begin{aligned}
I_\phi &= \frac{1}{3}\left[(L - x_1)^3 + x_1^3\right] \\
I_\theta &= I_\phi \sin^2(x_5)
\end{aligned}$$

and L=5.

Robot Arm Problem - Collocation Method

```
%---------------------------------------------------------------------%
%                    Robot Arm Problem                                %
%---------------------------------------------------------------------%
% Solve the following optimal control problem:                        %
% Minimize t_f                                                        %
% subject to the differential equation constraints                    %
% dx1/dt= x2                                                          %
% dx2/dt= u1/L                                                        %
% dx3/dt= x4
% dx4/dt= u2/Itheta
% dx5/dt= x6
% dx6/dt= u3/Iphi
%---------------------------------------------------------------------%
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                  %
%---------------------------------------------------------------------%
global L psStuff nstates ncontrols
global iGfun jGvar
%---------------------------------------------------------------------%
% END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                  %
%---------------------------------------------------------------------%


%---------------------------------------------------------------------%
%             Define the constants for the problem                    %
%---------------------------------------------------------------------%
L = 5;

%---------------------------------------------------------------------%
%  Define the sizes of quantities in the optimal control problem      %
%---------------------------------------------------------------------%
nstates = 6;
ncontrols = 3;

%---------------------------------------------------------------------%
%  Define bounds on the variables in the optimal control problem      %
%---------------------------------------------------------------------%
x10=4.5;  x20=0; x30=0;      x40=0; x50=pi/4; x60=0;
x1f=4.5;  x2f=0; x3f=2*pi/3; x4f=0; x5f=pi/4; x6f=0;
x1min = 0;   x1max = 2*pi;
x2min = -pi; x2max = pi;
x3min = -pi; x3max = pi;
x4min = -pi; x4max = pi;
x5min = -pi; x5max = pi;
x6min = -pi; x6max = pi;
u1min = -1;  u1max = 1;
u2min = -1;  u2max = 1;
u3min = -1;  u3max = 1;
t0min = 0; t0max = 0;
tfmin = 0; tfmax = 50;

%---------------------------------------------------------------------%
% In this section, we define the three type of discretizations        %
% that can be employed.  These three approaches are as follows:       %
%    (1) p-method  = global pseudospectral method                     %
%                  = single interval and the degree of the            %
%                    polynomial in the interval can be varied         %
%    (2) h-method  = fixed-degree polynomial in each interval         %
%                    and the number of intervals can be varied        %
%    (3) hp-method = can vary BOTH the degree of the polynomial       %
%                    in each interval and the number of intervals     %
%                                                                     %
% For simplicity in this tutorial, we will allow for either a         %
% p-method or an h-method.  Regardless of which method is being       %
% employed, the user needs to specify the following parameters:       %
%    (a) N = Polynomial Degree                                        %
%    (b) meshPoints = Set of Monotonically Increasing Mesh Points     %
%                     on the Interval $\tau\in[-1,+1]$.               %
%                                                                     %
% When using a p-method, the parameters N and meshPoints must be      %
% specified as follows:                                               %
%    (i)  meshPoints = [-1 1]                                         %
%    (ii) N = Choice of Polynomial Degree (e.g., N=10, N=20)          %
```

```
% When using an h-method, the parameters N and meshPoints must be %
% specified as follows:                                           %
%    (i)  meshPoints = $[\tau_1,\tau_2,\tau_3,\ldots,\tau_N]$     %
%                      where $\tau_1 = -1$, $\tau_N = 1$ and      %
%                      (\tau_2,\ldots,\tau_{N-1}) are             %
%                      monotonically increasing on the open       %
%                      interval $(-1,+1)$.                        %
%-----------------------------------------------------------------%
%      Compute Points, Weights, and Differentiation Matrix        %
%-----------------------------------------------------------------%
%-----------------------------------------------------------------%
% Choose Polynomial Degree and Number of Mesh Intervals           %
% numIntervals = 1 ===> p-method                                  %
% numIntervals > 1 ===> h-method                                  %
%-----------------------------------------------------------------%
N = 4;
numIntervals = 40;
%-----------------------------------------------------------------%
% DO NOT ALTER THE LINE OF CODE SHOWN BELOW!                      %
%-----------------------------------------------------------------%
meshPoints = linspace(-1,1,numIntervals+1).';
polyDegrees = N*ones(numIntervals,1);
[tau,w,D] = lgrPS(meshPoints,polyDegrees);
psStuff.tau = tau; psStuff.w = w; psStuff.D = D; NLGR = length(w);
%-----------------------------------------------------------------%
% DO NOT ALTER THE LINES OF CODE SHOWN ABOVE!                     %
%-----------------------------------------------------------------%

%-----------------------------------------------------------------%
% Set the bounds on the variables in the NLP.                     %
%-----------------------------------------------------------------%
zx1min = x1min*ones(length(tau),1);
zx1max = x1max*ones(length(tau),1);
zx1min(1) = x10;       zx1max(1) = x10;
zx1min(NLGR+1) = x1f;  zx1max(NLGR+1) = x1f;

zx2min = x2min*ones(length(tau),1);
zx2max = x2max*ones(length(tau),1);
zx2min(1) = x20;       zx2max(1) = x20;
zx2min(NLGR+1) = x2f;  zx2max(NLGR+1) = x2f;

zx3min = x3min*ones(length(tau),1);
zx3max = x3max*ones(length(tau),1);
zx3min(1) = x30;        zx3max(1) = x30;
zx3min(NLGR+1) = x3f;   zx3max(NLGR+1) = x3f;

zx4min = x4min*ones(length(tau),1);
zx4max = x4max*ones(length(tau),1);
zx4min(1) = x40;       zx4max(1) = x40;
zx4min(NLGR+1) = x4f;  zx4max(NLGR+1) = x4f;

zx5min = x5min*ones(length(tau),1);
zx5max = x5max*ones(length(tau),1);
zx5min(1) = x50;       zx5max(1) = x50;
zx5min(NLGR+1) = x5f;  zx5max(NLGR+1) = x5f;

zx6min = x6min*ones(length(tau),1);
zx6max = x6max*ones(length(tau),1);
zx6min(1) = x60;       zx6max(1) = x60;
zx6min(NLGR+1) = x6f;  zx6max(NLGR+1) = x6f;

zu1min = u1min*ones(length(tau)-1,1);
zu1max = u1max*ones(length(tau)-1,1);

zu2min = u2min*ones(length(tau)-1,1);
zu2max = u2max*ones(length(tau)-1,1);

zu3min = u3min*ones(length(tau)-1,1);
zu3max = u3max*ones(length(tau)-1,1);

zmin = [zx1min; zx2min; zx3min; zx4min; zx5min; zx6min; zu1min; zu2min; zu3min; t0min; tfmin];
zmax = [zx1max; zx2max; zx3max; zx4max; zx5max; zx6max; zu1max; zu2max; zu3max; t0max; tfmax];

%-----------------------------------------------------------------%
% Set the bounds on the constraints in the NLP.                   %
%-----------------------------------------------------------------%
defectMin = zeros(nstates*(length(tau)-1),1);
```

```
defectMax = zeros(nstates*(length(tau)-1),1);
pathMin = []; pathMax = [];
eventMin = []; eventMax = [];
Fmin = [ defectMin; pathMin; eventMin];
Fmax = [ defectMax; pathMax; eventMax];

%--------------------------------------------------------------------%
% Supply an initial guess for the NLP.                               %
%--------------------------------------------------------------------%
x1guess = linspace(x10,x1f,NLGR+1).';
x2guess = linspace(x20,x2f,NLGR+1).';
x3guess = linspace(x30,x3f,NLGR+1).';
x4guess = linspace(x40,x4f,NLGR+1).';
x5guess = linspace(x50,x5f,NLGR+1).';
x6guess = linspace(x60,x6f,NLGR+1).';
u1guess = linspace(0,0,NLGR).';
u2guess = linspace(0,0,NLGR).';
u3guess = linspace(0,0,NLGR).';
t0guess = 0;
tfguess = 15;
z0 = [x1guess; x2guess; x3guess; x4guess; x5guess; x6guess; u1guess; u2guess; u3guess; t0guess; tfguess];

%--------------------------------------------------------------------%
% Generate derivatives and sparsity pattern using Adigator           %
%--------------------------------------------------------------------%
% - Constraint Funtction Derivatives
xsize  = size(z0);
x      = adigatorCreateDerivInput(xsize,'z0');
output = adigatorGenJacFile('robotArmFun',{x});
S_jac  = output.JacobianStructure;
[iGfun,jGvar] = find(S_jac);

% - Objective Funtcion Derivatives
xsize  = size(z0);
x      = adigatorCreateDerivInput(xsize,'z0');
output = adigatorGenJacFile('robotArmObj',{x});
grd_structure = output.JacobianStructure;

%--------------------------------------------------------------------%
% Set IPOPT callback functions
%--------------------------------------------------------------------%
funcs.objective   = @(Z)robotArmObj(Z);
funcs.gradient    = @(Z)robotArmGrd(Z);
funcs.constraints = @(Z)robotArmCon(Z);
funcs.jacobian    = @(Z)robotArmJac(Z);
funcs.jacobianstructure = @()robotArmJacPat(S_jac);
options.ipopt.hessian_approximation = 'limited-memory';

%--------------------------------------------------------------------%
% Set IPOPT Options %
%--------------------------------------------------------------------%
options.ipopt.tol = 1e-5;
options.ipopt.linear_solver = 'ma57';
options.ipopt.max_iter = 2000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['robotArm','IPOPTinfo.txt']; % print output file
options.ipopt.print_level = 5; % set print level default

options.lb = zmin; % Lower bound on the variables.
options.ub = zmax; % Upper bound on the variables.
options.cl = Fmin; % Lower bounds on the constraint functions.
options.cu = Fmax; % Upper bounds on the constraint functions.

%--------------------------------------------------------------------%
% Call IPOPT
%--------------------------------------------------------------------%
[z, info] = ipopt(z0,funcs,options);

%--------------------------------------------------------------------%
% extract lagrange multipliers from ipopt output, info
%--------------------------------------------------------------------%
Fmul = info.lambda;

%--------------------------------------------------------------------%
% Extract the state and control from the decision vector z.          %
```

```
% Remember that the state is approximated at the LGR points       %
% plus the final point, while the control is only approximated    %
% at only the LGR points.                                         %
%-----------------------------------------------------------------%
x1 = z(1:NLGR+1);
x2 = z(NLGR+2:2*(NLGR+1));
x3 = z(2*(NLGR+1)+1:3*(NLGR+1));
x4 = z(3*(NLGR+1)+1:4*(NLGR+1));
x5 = z(4*(NLGR+1)+1:5*(NLGR+1));
x6 = z(5*(NLGR+1)+1:6*(NLGR+1));
u1 = z(6*(NLGR+1)+1:6*(NLGR+1)+NLGR);
u2 = z(6*(NLGR+1)+NLGR+1:6*(NLGR+1)+2*NLGR);
u3 = z(6*(NLGR+1)+2*NLGR+1:6*(NLGR+1)+3*NLGR);
t0 = z(end-1);
tf = z(end);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%-----------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                  %
% the defect constraints.                                         %
%-----------------------------------------------------------------%
multipliersDefects = Fmul(1:nstates*NLGR);
multipliersDefects = reshape(multipliersDefects,NLGR,nstates);
%-----------------------------------------------------------------%
% Compute the costates at the LGR points via transformation       %
%-----------------------------------------------------------------%
costateLGR = inv(diag(w))*multipliersDefects;
%-----------------------------------------------------------------%
% Compute the costate at the tau=+1 via transformation            %
%-----------------------------------------------------------------%
costateF = D(:,end).'*multipliersDefects;
%-----------------------------------------------------------------%
% Now assemble the costates into a single matrix                  %
%-----------------------------------------------------------------%
costate = [costateLGR; costateF];
lamx1 = -costate(:,1); lamx2 = -costate(:,2); lamx3 = -costate(:,3); lamx4 = -costate(:,4); lamx5 = -costate(:,5); lamx6 = -costate(:,6);

%Evaluating the Hamiltonian to estimate the closeness of the obtained solution to the true optimal solution%
x1LGR=x1(1:(end-1)); x2LGR=x2(1:(end-1)); x3LGR=x3(1:(end-1));
x4LGR=x4(1:(end-1)); x5LGR=x5(1:(end-1)); x6LGR=x6(1:(end-1));

lamx1LGR=lamx1(1:(end-1)); lamx2LGR=lamx2(1:(end-1)); lamx3LGR=lamx3(1:(end-1));
lamx4LGR=lamx4(1:(end-1)); lamx5LGR=lamx5(1:(end-1)); lamx6LGR=lamx6(1:(end-1));

IphiLGR=1/3.*((L-x1LGR).^3 + x1LGR.^3);
IthetaLGR=IphiLGR.*sin(x5LGR).^2;
x1dot=x2LGR;
x2dot=u1./L;
x3dot=x4LGR;
x4dot=u2./IthetaLGR;
x5dot=x6LGR;
x6dot=u3./IphiLGR;
Hamiltonian=lamx1LGR.*x1dot + lamx2LGR.*x2dot + lamx3LGR.*x3dot + lamx4LGR.*x4dot + lamx5LGR.*x5dot + lamx6LGR.*x6dot;
%-----------------------------------------------------------------%
% plot results
%-----------------------------------------------------------------%
close all;
lw=1; %Specifying LineWidth for Plots

figure;
subplot(2,3,1); plot(t,x1,'-bs','LineWidth',lw);
xlabel('Time'); yl1=ylabel('$x_1$');
set(yl1,'Interpreter','latex'); grid on;
ax = gca;ax.GridLineStyle = ':';ax.GridAlpha = 0.3;
ax.FontSize = 16;ax.LineWidth = 1.4;leg1=legend('$x_1$');
set(leg1,'Interpreter','latex');xlim([0 t(end)]);hold on;

subplot(2,3,2); plot(t,x2,'-bs','LineWidth',lw);
xlabel('Time'); yl2=ylabel('$x_2$');
set(yl2,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg2=legend('$x_2$');
set(leg2,'Interpreter','latex'); xlim([0 t(end)]); hold on;

subplot(2,3,3); plot(t,x3,'-bs','LineWidth',lw);
xlabel('Time'); yl3=ylabel('$x_3$');
```

```
set(yl3,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg3=legend('$x_3$');
set(leg3,'Interpreter','latex'); xlim([0 t(end)]); hold on;

subplot(2,3,4); plot(t,x4,'-bs','LineWidth',lw);
xlabel('Time'); yl4=ylabel('$x_4$');
set(yl4,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg4=legend('$x_4$');
set(leg4,'Interpreter','latex'); xlim([0 t(end)]); hold on;

subplot(2,3,5); plot(t,x5,'-bs','LineWidth',lw);
xlabel('Time'); yl5=ylabel('$x_5$');
set(yl5,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg5=legend('$x_5$');
set(leg5,'Interpreter','latex'); xlim([0 t(end)]); hold on;

subplot(2,3,6); plot(t,x6,'-bs','LineWidth',lw);
xlabel('Time'); yl6=ylabel('$x_6$');
set(yl6,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg6=legend('$x_6$');
set(leg6,'Interpreter','latex'); xlim([0 t(end)]); hold on;

figure;
subplot(2,2,1); plot(tLGR,u1,'-bs','LineWidth',lw);
xlabel('Time'); yl7=ylabel('$u_1$');
set(yl7,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg7=legend('$u_1$');
set(leg7,'Interpreter','latex'); xlim([0 t(end)]); hold on;

subplot(2,2,2); plot(tLGR,u2,'-bs','LineWidth',lw);
xlabel('Time'); yl8=ylabel('$u_2$');
set(yl8,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg8=legend('$u_2$');
set(leg8,'Interpreter','latex'); xlim([0 t(end)]); hold on;

subplot(2,2,3); plot(tLGR,u3,'-bs','LineWidth',lw);
xlabel('Time'); yl9=ylabel('$u_3$');
set(yl9,'Interpreter','latex'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; leg9=legend('$u_3$');
set(leg9,'Interpreter','latex'); xlim([0 t(end)]); hold on;

figure;
plot(t,lamx1,'-b+',t,lamx2,'-ro',t,lamx3,'-bs',t,lamx4,'-go',t,lamx5,'-mo',t,lamx6,'-y+','LineWidth',lw);
xlabel('Time'); ylabel('Costates'); title('Optimal Costate Trajectory'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3; ax.FontSize = 16; ax.LineWidth = 1.4;
leg10=legend('$\lambda_{x_1}$','$\lambda_{x_2}$','$\lambda_{x_3}$','$\lambda_{x_4}$','$\lambda_{x_5}$','$\lambda_{x_6}$');
set(leg10,'Interpreter','latex'); xlim([0 t(end)]); hold on;

figure;
plot(tLGR,Hamiltonian,'-bs','LineWidth',lw);
xlabel('Time'); yl9=ylabel('Hamiltonian'); grid on;
ax = gca; ax.GridLineStyle = ':'; ax.GridAlpha = 0.3;
ax.FontSize = 16; ax.LineWidth = 1.4; xlim([0 t(end)]);
hold on;
```

```
function C = robotArmFun(z)

%---------------------------------------------------------------------%
% Constraint functions for the Robot Arm problem                      %
% problem.  This function is designed to be used with the NLP         %
% solver IPOPT.                                                       %
%---------------------------------------------------------------------%
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE BELOW!            %
global psStuff nstates ncontrols L
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE!            %
%---------------------------------------------------------------------%

%---------------------------------------------------------------------%
% Radau pseudospectral method quantities required:                    %
%   - Differentiation matrix (psStuff.D)                              %
%   - Legendre-Gauss-Radau weights (psStuff.w)                        %
%   - Legendre-Gauss-Radau points (psStuff.tau)                       %
%---------------------------------------------------------------------%
D = psStuff.D; tau = psStuff.tau; w = psStuff.w;

%---------------------------------------------------------------------%
% Decompose the NLP decision vector into pieces containing            %
%    - the state                                                      %
%    - the control                                                    %
%    - the initial time                                               %
%    - the final time                                                 %
%---------------------------------------------------------------------%
N = length(tau)-1;
stateIndices = 1:nstates*(N+1);
controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
t0Index = controlIndices(end)+1;
tfIndex = t0Index+1;
stateVector = z(stateIndices);
controlVector = z(controlIndices);
t0 = z(t0Index);
tf = z(tfIndex);

%---------------------------------------------------------------------%
% Reshape the state and control parts of the NLP decision vector      %
% to matrices of sizes (N+1) by nstates and (N+1) by ncontrols,       %
% respectively.  The state is approximated at the N LGR points        %
% plus the final point.  Thus, each column of the state vector is     %
% length N+1.  The LEFT-HAND SIDE of the defect constraints, D*X,     %
% uses the state at all of the points (N LGR points plus final        %
% point).  The RIGHT-HAND SIDE of the defect constraints,             %
% (tf-t0)F/2, uses the state and control at only the LGR points.      %
% Thus, it is necessary to extract the state approximations at        %
% only the N LGR points.  Finally, in the Radau pseudospectral        %
% method, the control is approximated at only the N LGR points.      %
%---------------------------------------------------------------------%
statePlusEnd   = reshape(stateVector,N+1,nstates);
control = reshape(controlVector,N,ncontrols);
stateLGR = statePlusEnd(1:end-1,:);

%---------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR.     %
%---------------------------------------------------------------------%
x1 = stateLGR(:,1);
x2 = stateLGR(:,2);
```

```
x3 = stateLGR(:,3);
x4 = stateLGR(:,4);
x5 = stateLGR(:,5);
x6 = stateLGR(:,6);
u1 = control (:,1);
u2 = control (:,2);
u3 = control (:,3);


%---------------------------------------------------------------------%
% Compute the right-hand side of the differential equations at    %
% the N LGR points.  Each component of the right-hand side is     %
% stored as a column vector of length N, that is each column has  %
% the form                                                        %
%                   [ f_i(x_1,u_1,t_1) ]                          %
%                   [ f_i(x_2,u_2,t_2) ]                          %
%                           .                                     %
%                           .                                     %
%                           .                                     %
%                   [ f_i(x_N,u_N,t_N) ]                          %
% where "i" is the right-hand side of the ith component of the    %
% vector field f.  It is noted that in MATLABB the calculation of %
% the right-hand side is vectorized.                              %
%---------------------------------------------------------------------%
%Defining some terms in the dynamics:
Iphi=1/3.*((L-x1).^3 + x1.^3);
Itheta=Iphi.*sin(x5).^2;

x1dot=x2;
x2dot=u1./L;
x3dot=x4;
x4dot=u2./Itheta;
x5dot=x6;
x6dot=u3./Iphi;
diffeqRHS = [x1dot, x2dot, x3dot, x4dot, x5dot, x6dot];

%---------------------------------------------------------------------%
% Compute the left-hand side of the defect constraints, recalling %
% that the left-hand side is computed using the state at the LGR  %
% points PLUS the final point.                                    %
%---------------------------------------------------------------------%
diffeqLHS = D*statePlusEnd;

%---------------------------------------------------------------------%
% Construct the defect constraints at the N LGR points.          %
% Remember that the right-hand side needs to be scaled by the     %
% factor (tf-t0)/2 because the rate of change of the state is     %
% being taken with respect to $\tau\in[-1,+1]$.  Thus, we have    %
% $dt/t\dau=(tf-t0)/2$.                                           %
%---------------------------------------------------------------------%
defects = diffeqLHS-(tf-t0)*diffeqRHS/2;

%---------------------------------------------------------------------%
% Reshape the defect contraints into a column vector.             %
%---------------------------------------------------------------------%
defects = reshape(defects,N*nstates,1);

%---------------------------------------------------------------------%
% Construct the constraint vector.        %
%---------------------------------------------------------------------%
C = defects;
```

```
function constraints = robotArmCon(Z)
% computes the constraints
output      = robotArmFun(Z);
constraints = output;
end



function grd = robotArmGrd(Z)
% computes the gradient
output = robotArmObj_Jac(Z);
grd    = output;
end

function jac = robotArmJac(Z)
% computes the jacobian
[jac,~] = robotArmFun_Jac(Z);
end

function jacpat = robotArmJacPat(S_jac)
% computes the jacobian structure
jacpat = S_jac;
end

function obj = robotArmObj(z)
% Computes the objective function of the problem
tf = z(end);
% Cost function
J   = tf;
obj = J;
end
```

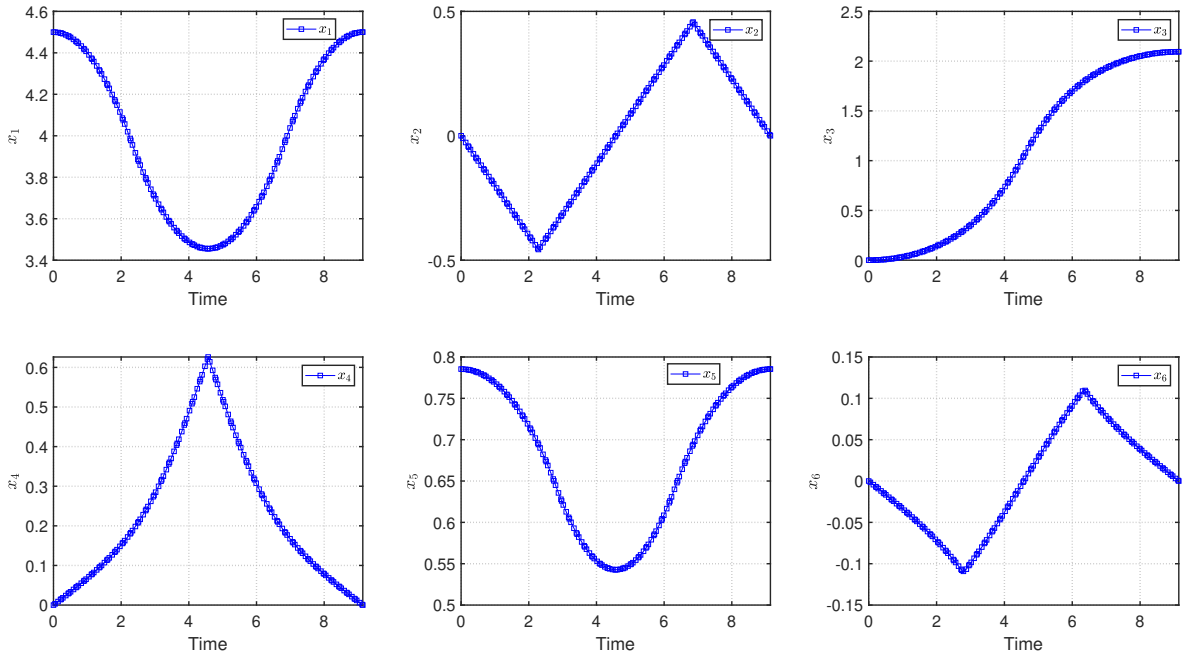
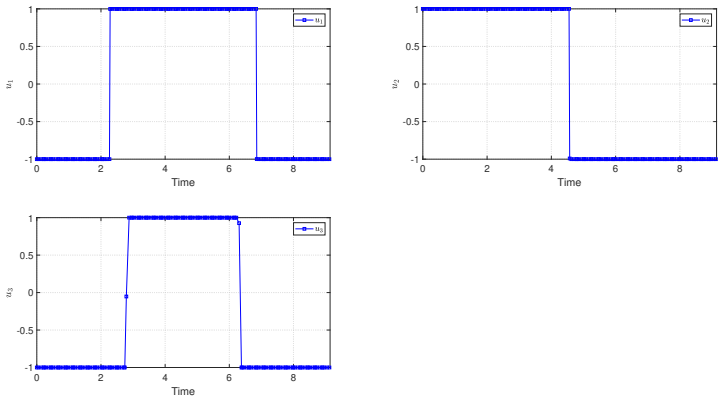


Figure 1: Collocation : Robot Arm Problem - States and Controls



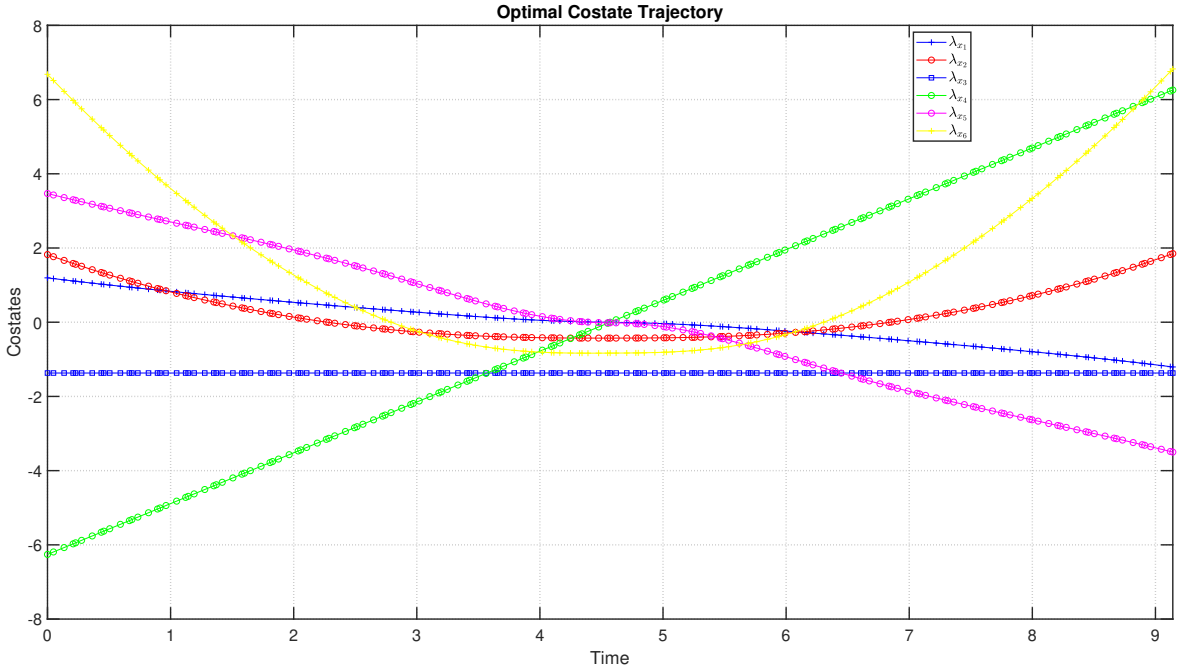
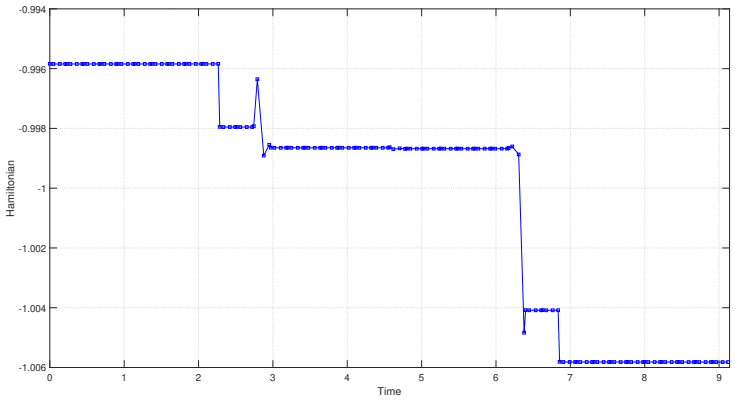


Figure 2: Collocation : Robot Arm Problem - Costates and Hamiltonian



Robot Arm Problem:

Closeness of the numerical solution to the true optimal solution.

The first-order optimality conditions characterize the true optimal solution. So, we can see how close the numerical solution is to the true solution by looking it how well these optimality conditions are satisfied. To this end, costate estimation is done after a solution is obtained using the collocation method.

t_f being free, the transversality condition, $\frac{\partial M}{\partial t_f} - \boldsymbol{\nu}^{\boldsymbol{T}} \frac{\partial \boldsymbol{b}}{\partial t_f} + H(t_f) = 0$, gives us $H(t_f) = -1$.
Since the Hamiltonian is not an explicit function of time $H(t) \equiv -1$.
Since all final states are fixed, all the final costates are free.
For 40 sub-intervals and a 4th degree polynomial approximation within each sub-interval: $H(t) \approx -1 \quad \forall t$ to an accuracy of the order of 10^{-2} (error in the third decimal point).

Robustness and computational efficiency of the collocation method

Unlike indirect shooting, collocation can achieve convergence from a relatively far-off initial guess. It is also computationally efficient since the matrix computations involve sparse matrices.
In the parameter optimization method, how good the numerical solution is depends how well the parametrization we choose reflects the true solution. For instance, if we approximate a bang-bang control with, say, a quadratic polynomial, it would be a lousy approximation and the numerical solution might not look anything like the true optimal solution.
Collocation method, on the other hand uses several sub-intervals, with each sub-interval having a polynomial approximation (with LGR support points). The number of sub-intervals and the degree of the polynomial can be varied to get a numerical solution which is very close to the true solution.

Conclusion

For the robot arm problem, it turns out that the optimal control is bang-bang. So, parameter optimization would've been a poor choice. Although possible, it would be terribly hard to get a solution using indirect shooting, given that it's a 6th order system and the method relies on a very good initial guess for convergence. So, collocation seems to be the best method for solving this problem.