Optimal Control
University of Florida
Mechanical and Aerospace Engineering

Project - Report

Optimal Control Indirect and Direct Method Implementation

Contents

1 Elementary Problem: Linear Tangent Steering

1.1 Problem Formulation:

The Linear Tangent Steering Optimal Control Problem is as follows;

Minimize the cost functional,

$$J = t_f, \tag{1}$$

Subject to the dynamic constraints,

$$\begin{aligned} \dot{x}_1 &= x_3, \\ \dot{x}_2 &= x_4, \\ \dot{x}_3 &= a.cos(u), \\ \dot{x}_4 &= a.sin(u), \end{aligned} \tag{2}$$

With the following boundary conditions,

$$\begin{aligned} t_0 &= 0 & t_f &= \text{Free}, \\ x_1(0) &= 0 & x_1(t_f) &= \text{Free}, \\ x_2(0) &= 0 & x_2(t_f) &= 5, \\ x_3(0) &= 0 & x_3(t_f) &= 45, \\ x_4(0) &= 0 & x_4(t_f) &= 0, \end{aligned} \tag{3}$$

Where $a = 100$.

1.2 Indirect Method: Hamiltonian Boundary Value Problem [HBVP]

1.2.1 Optimal Solution: Using Optimal Control Theory

We have,

$$J = t_f,$$

We know,

$$J = M + \int_{t_0}^{t_f} L \; dt,$$

Where,

$$\begin{aligned} M &= \text{Meyer Cost} \\ L &= \text{Lagrange Cost} \end{aligned}$$

Now we know that,

$$\begin{aligned} M &= t_f, \\ L &= 0, \end{aligned}$$

Creating the Hamiltonian,

$$H = L + \underline{\lambda}^T \underline{\text{f}}, \tag{4}$$

Where,

$$\begin{aligned} L &= 0, \\ \underline{\lambda}^T &= \left[\lambda_{x_1} \quad \lambda_{x_2} \quad \lambda_{x_3} \quad \lambda_{x_4}\right], \\ \underline{\text{f}} &= \begin{bmatrix} x_3 \\ x_4 \\ a.cos(u) \\ a.sin(u) \end{bmatrix}, \end{aligned} \tag{5}$$

Hence the Hamiltonian is as follows,

$$H = \lambda_{x_1}[x_3] + \lambda_{x_2}[x_4] + \lambda_{x_3}[a.cos(u)] + \lambda_{x_4}[a.sin(u)], \tag{6}$$

Now, using 1^{st} Order Optimality Conditions for $\underline{\mathrm{x}}$,

$$\dot{\underline{\mathrm{x}}} = \left[\frac{\partial H}{\partial \underline{\lambda}}\right]^T, \tag{7}$$

$$\tag{8}$$

We get,

$$\begin{aligned}
\dot{x}_1 &= \frac{\partial H}{\partial \lambda_{x_1}} &&= x_3,\\
\dot{x}_2 &= \frac{\partial H}{\partial \lambda_{x_2}} &&= x_4,\\
\dot{x}_3 &= \frac{\partial H}{\partial \lambda_{x_3}} &&= a.cos(u),\\
\dot{x}_4 &= \frac{\partial H}{\partial \lambda_{x_4}} &&= a.sin(u),
\end{aligned} \tag{9}$$

Now, using 1^{st} Order Optimality Conditions for $\underline{\lambda}$,

$$\dot{\underline{\lambda}} = -\left[\frac{\partial H}{\partial \underline{\mathrm{x}}}\right]^T, \tag{10}$$

We get,

$$\begin{aligned}
\dot{\lambda_{x_1}} &= -\frac{\partial H}{\partial x_1} &&= 0,\\
\dot{\lambda_{x_2}} &= -\frac{\partial H}{\partial x_2} &&= 0,\\
\dot{\lambda_{x_3}} &= -\frac{\partial H}{\partial x_3} &&= -\lambda_{x_1},\\
\dot{\lambda_{x_4}} &= -\frac{\partial H}{\partial x_4} &&= -\lambda_{x_2},
\end{aligned} \tag{11}$$

Now, using 1^{st} Order Optimality Conditions for H,

$$\left[\frac{\partial H}{\partial u}\right] = 0, \tag{12}$$

We get,

$$\left[\frac{\partial H}{\partial u}\right] = -a.\lambda_{x_3}.sin(u) + a.\lambda_{x_4}.cos(u),$$
$$\frac{\lambda_{x_4}}{\lambda_{x_3}} = tan(u),$$

Therefore, we have,

$$u = tan^{-1}\left[\frac{\lambda_{x_4}}{\lambda_{x_3}},\right] \tag{13}$$

Since δt_f and $\delta \underline{\mathrm{x}}_f$ are not fixed, we can use the following Transversality Conditions,

$$\left[\frac{\partial M}{\partial \underline{\mathrm{x}}(t_f)} - \nu^T \frac{\partial \underline{\mathrm{b}}}{\partial \underline{\mathrm{x}}(t_f)} - \underline{\lambda}^T(t_f)\right] = \underline{0}^T \tag{14}$$
$$\left[\frac{\partial M}{\partial t_f} - \nu^T \frac{\partial \underline{\mathrm{b}}}{\partial t_f} + H(t_f)\right] = 0, \tag{15}$$

Where,

$$\underline{\mathrm{b}} = \begin{bmatrix} x_1(t_0) - 0 \\ x_2(t_0) - 0 \\ x_3(t_0) - 0 \\ x_4(t_0) - 0 \\ x_2(t_f) - 5 \\ x_3(t_f) - 45 \\ x_4(t_f) - 0 \end{bmatrix},$$

and,

$$\underline{\nu^T} = \left[\nu_1 \quad \nu_2 \quad \nu_3 \quad \nu_4 \quad \nu_5 \quad \nu_6 \quad \nu_7\right],$$

Computing terms for the Transversality condition corresponding to $\delta\underline{\mathrm{x}}_f \neq 0$, we get,

$$\begin{aligned}
\frac{\partial M}{\partial \underline{\mathrm{x}}_{t_f}} &= \begin{bmatrix} \dfrac{\partial M}{\partial x_1(t_f)} & \dfrac{\partial M}{\partial x_2(t_f)} & \dfrac{\partial M}{\partial x_3(t_f)} & \dfrac{\partial M}{\partial x_4(t_f)} \end{bmatrix}, \\
\frac{\partial M}{\partial \underline{\mathrm{x}}_{t_f}} &= \begin{bmatrix} \dfrac{\partial t_f}{\partial x_1(t_f)} & \dfrac{\partial t_f}{\partial x_2(t_f)} & \dfrac{\partial t_f}{\partial x_3(t_f)} & \dfrac{\partial t_f}{\partial x_4(t_f)} \end{bmatrix}, \\
\frac{\partial M}{\partial \underline{\mathrm{x}}_{t_f}} &= \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix},
\end{aligned} \tag{16}$$

$$\begin{aligned}
\frac{\partial \underline{\mathrm{b}}}{\partial \underline{\mathrm{x}}_{t_f}} &= \begin{bmatrix} \dfrac{\partial \underline{\mathrm{b}}}{\partial x_1(t_f)} & \dfrac{\partial \underline{\mathrm{b}}}{\partial x_2(t_f)} & \dfrac{\partial \underline{\mathrm{b}}}{\partial x_3(t_f)} & \dfrac{\partial \underline{\mathrm{b}}}{\partial x_4(t_f)} \end{bmatrix}, \\
\frac{\partial \underline{\mathrm{b}}}{\partial \underline{\mathrm{x}}_{t_f}} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
\end{aligned} \tag{17}$$

$$\nu^T \frac{\partial \underline{\mathrm{b}}}{\partial \underline{\mathrm{x}}(t_f)} = \begin{bmatrix} 0 & \nu_5 & \nu_6 & \nu_7 \end{bmatrix}, \tag{18}$$

Now, we have,

$$\left[\frac{\partial M}{\partial \underline{\mathrm{x}}(t_f)} - \nu^T \frac{\partial \underline{\mathrm{b}}}{\partial \underline{\mathrm{x}}(t_f)} - \underline{\lambda}^T(t_f)\right] = \underline{0}^T$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & \nu_5 & \nu_6 & \nu_7 \end{bmatrix} - \begin{bmatrix} \lambda_{x_1}(t_f) & \lambda_{x_2}(t_f) & \lambda_{x_3}(t_f) & \lambda_{x_4}(t_f) \end{bmatrix} = \underline{0}^T$$

Therefore, we get,

$$\lambda_{x_1}(t_f) = 0, \tag{19}$$

and from equations (11) and (19) we have,

$$\lambda_{x_1}(t_0) = 0, \tag{20}$$

Computing terms for the Transversality condition corresponding to $\delta t_f \neq 0$, we get,

$$\frac{\partial \underline{\mathrm{M}}}{\partial t_f} = \begin{bmatrix} \dfrac{\partial t_f}{\partial x_1(t_f)} \end{bmatrix} = 1, \tag{21}$$

$$\frac{\partial b}{\partial t_f} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{22}$$

$$\nu^T \frac{\partial \underline{\mathrm{b}}}{\partial \underline{\mathrm{x}}(t_f)} = 0, \tag{23}$$

Now, we have,

$$\left[\frac{\partial M}{\partial \underline{\mathrm{x}}(t_f)} - \nu^T \frac{\partial \underline{\mathrm{b}}}{\partial \underline{\mathrm{x}}(t_f)} + H(t_f)\right] = 0$$
$$\begin{bmatrix} 1 \end{bmatrix} - \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} H(t_f) \end{bmatrix} = 0$$

Therefore, we get,

$$H(t_f) = -1, \tag{24}$$

Formulating the HBVP,
Known Initial Conditions are as follows,

$$\begin{aligned} x_1(0) &= 0, \\ x_2(0) &= 0, \\ x_3(0) &= 0, \\ x_4(0) &= 0, \\ \lambda_{x_1}(0) &= 0, \\ t_0 &= 0, \end{aligned} \tag{25}$$

$$\tag{26}$$

Unknown Initial Conditions are as follows,

$$\begin{aligned} \lambda_{x_2}(0) &= ??, \\ \lambda_{x_3}(0) &= ??, \\ \lambda_{x_4}(0) &= ??, \\ t_f &= 0, \end{aligned} \tag{27}$$

$$\tag{28}$$

Note:

- The Root Finder has to find the Optimal Values for these variables.
- We need to provide an initial guess for these variables.

Dynamics to be integrated,

$$\begin{aligned} \dot{x}_1(t) &= x_3(t), \\ \dot{x}_2(t) &= x_4(t), \\ \dot{x}_3(t) &= a.cos(u(t)), \\ \dot{x}_4(t) &= a.sin(u(t)), \\ \dot{\lambda}_{x_1}(t) &= 0, \\ \dot{\lambda}_{x_2}(t) &= 0, \\ \dot{\lambda}_{x_3}(t) &= -\lambda_{x_1}(t), \\ \dot{\lambda}_{x_4}(t) &= -\lambda_{x_2}(t), \end{aligned} \tag{29}$$

Where,

$$u = tan^{-1}\left[\frac{\lambda_{x_4}}{\lambda_{x_3}},\right] \tag{30}$$

Error to be minimized to zero is as follows,

$$\underline{\mathrm{e}} = \begin{bmatrix} x_2(t_f) - 5 \\ x_3(t_f) - 45 \\ x_4(t_f) - 0 \\ \lambda_{x_1}(t_f) - 0 \\ H(t_f) + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{31}$$

1.2.2 MATLAB Code

```
%% Main File : Linear Tangent Steering : Indirect Shooting

clear all;
clc;
close all;

%% Initial Conditions

% Known Initial Conditions for the ODE Solver
x1_0=0;
x2_0=0;
x3_0=0;
x4_0=0;
Lam_x1_0=0;
t0_0=0;

X0_Known_ODESolver=[x1_0;x2_0;x3_0;x4_0;Lam_x1_0;t0_0];

% Unknown Initial Conditions for the Root Finder
Lam_x2_0=randn(1);
Lam_x3_0=randn(1);
Lam_x4_0=randn(1);
tf_0=1;

X0_RootFinder=[Lam_x2_0;Lam_x3_0;Lam_x4_0;tf_0];

%% Root Finder

% Options for the Root Finder
Options=optimset('Display','iter','TolFun',1e-5);

% Calling the Root Finder
tic; % Timing the Process

[X_Sol_RootFinder,fval,exitflag,output] = ...
    fsolve(@LinearTangent_ODES0lver,X0_RootFinder,Options);

TimeTaken = toc;

%% Solving the ODE with Optimal Initial Conditions

% Inital Conditions

```

```
% Known Initial Conditions for the ODE Solver
x1_0=0;
x2_0=0;
x3_0=0;
x4_0=0;
Lam_x1_0=0;
t0_0=0;

% Initial Conditions from the Root Finder
Lam_x2_0=X_Sol_RootFinder(1);
Lam_x3_0=X_Sol_RootFinder(2);
Lam_x4_0=X_Sol_RootFinder(3);
tf_0=X_Sol_RootFinder(4);

% Complete Initial Conditions for the ODE Solver
X0_ODESolver = [x1_0;x2_0;x3_0;x4_0;Lam_x1_0;Lam_x2_0;Lam_x3_0;Lam_x4_0];

% Creating Time Span for Integration
tf=tf_0;
T_Span_ODESolver=[0,tf];

% Options for the ODE Solver
Options=optimset('Display','Iter','TolFun',1e-3,'TolX',1e-3);

% Calling the ODE Solver
[t_ODESolver,X_sol_ODESolver]=ode113(@LinearTangent_ODEEquations ,...
    T_Span_ODESolver,X0_ODESolver,Options);

% Getting the Outputs from the ODE Solver
x1=X_sol_ODESolver(:,1);
x2=X_sol_ODESolver(:,2);
x3=X_sol_ODESolver(:,3);
x4=X_sol_ODESolver(:,4);
Lam_x1=X_sol_ODESolver(:,5);
Lam_x2=X_sol_ODESolver(:,6);
Lam_x3=X_sol_ODESolver(:,7);
Lam_x4=X_sol_ODESolver(:,8);

% Computing Control

Control=atan2(Lam_x4,Lam_x3);

%% Plotting Results

% Plotting States
```

```
figure(1)
hold on
grid on
plot(t_ODESolver,x1,'-g','LineWidth',1.5);
plot(t_ODESolver,x2,'-.m','LineWidth',1.5);
plot(t_ODESolver,x3,'-r','LineWidth',1.5);
plot(t_ODESolver,x4,'--b','LineWidth',1.5);
title('States vs. Time','Interpreter','latex');
xlabel('Time ','Interpreter','latex');
ylabel('States','Interpreter','latex');
legend1=legend('$x_{1}$','$x_{2}$','$x_{3}$','$x_{4}$');
set(legend1,'Interpreter','latex');
hold off;

% Plotting Control
figure(2)
hold on
grid on
plot(t_ODESolver,Control,'-k','LineWidth',1.5);
title('Control - $u(t)$ vs. Time','Interpreter','latex');
xlabel('Time ','Interpreter','latex');
ylabel('$u(t)$','Interpreter','latex');
hold off;

% Plotting Costates
figure(3)
hold on
grid on
plot(t_ODESolver,Lam_x1,'-g','LineWidth',1.5);
plot(t_ODESolver,Lam_x2,'-.m','LineWidth',1.5);
plot(t_ODESolver,Lam_x3,'-r','LineWidth',1.5);
plot(t_ODESolver,Lam_x4,'--b','LineWidth',1.5);
title('Co-States vs. Time','Interpreter','latex');
xlabel('Time ','Interpreter','latex');
ylabel('Co-States','Interpreter','latex');
legend2=legend('$\lambda_{x_{1}}$','$\lambda_{x_{2}}$','$\lambda_{x_{3}}$','$\lambda_{x_{4}}$');
set(legend2,'Interpreter','latex');
hold off;

fprintf('Time taken to solve the HBVP = %.4f',TimeTaken)
```

```
function [ Equation_Derivative ] = LinearTangent_ODEEquations(
    T_Span_ODESolver,X0_ODESolver )

%% ODE Equations : Problem 1 - Part 2

%% Getting Required Values from the incoming Vectors

% From T_Span_ODESolver
t=T_Span_ODESolver;

% From X0_ODESolver
x1=X0_ODESolver(1);
x2=X0_ODESolver(2);
x3=X0_ODESolver(3);
x4=X0_ODESolver(4);
Lam_x1=X0_ODESolver(5);
Lam_x2=X0_ODESolver(6);
Lam_x3=X0_ODESolver(7);
Lam_x4=X0_ODESolver(8);

% Initializing P_Dot
Equation_Derivative=zeros(8,1);

%% Setting up the ODE Equations

% Constant
a=100;

% Computing U - Control
u=atan2(Lam_x4,Lam_x3);

% Equations
x1_Derivative=x3;
x2_Derivative=x4;
x3_Derivative=a*cos(u);
x4_Derivative=a*sin(u);
Lam_x1_Derivative=0;
Lam_x2_Derivative=0;
Lam_x3_Derivative=-Lam_x1;
Lam_x4_Derivative=-Lam_x2;

% Creating Equation Vector
Equation_Derivative(1)=x1_Derivative;
Equation_Derivative(2)=x2_Derivative;
Equation_Derivative(3)=x3_Derivative;
```

```
Equation_Derivative(4)=x4_Derivative;
Equation_Derivative(5)=Lam_x1_Derivative;
Equation_Derivative(6)=Lam_x2_Derivative;
Equation_Derivative(7)=Lam_x3_Derivative;
Equation_Derivative(8)=Lam_x4_Derivative;

end
```

```matlab
function [ Error ] = LinearTangent_ODES0lver( X0_RootFinder )

%% ODE Solver and Error Calculator : Problem 1 - Part 2

%% Inital Conditions

% Known Initial Conditions for the ODE Solver
x1_0=0;
x2_0=0;
x3_0=0;
x4_0=0;
Lam_x1_0=0;
t0_0=0;

X0_Known_ODESolver=[x1_0;x2_0;x3_0;x4_0;Lam_x1_0];

% Initial Conditions from the Root Finder
Lam_x2_0=X0_RootFinder(1);
Lam_x3_0=X0_RootFinder(2);
Lam_x4_0=X0_RootFinder(3);
tf_0=X0_RootFinder(4);

X0_RootFinder_Guess=X0_RootFinder(1:end-1);

% Complete Initial Conditions for the ODE Solver
X0_ODESolver = [X0_Known_ODESolver;X0_RootFinder_Guess];

%% ODE Solver

% Creating Time Span for Integration
tf=tf_0;

T_Span_ODESolver=[0,tf];

% Options for the ODE Solver
Options=optimset('Display','Iter','TolFun',1e-6,'TolX',1e-6);

% Calling the ODE Solver
[t_ODESolver,X_sol_ODESolver]=ode113(@LinearTangent_ODEEquations ,...
    T_Span_ODESolver,X0_ODESolver,Options);

% Getting the Outputs from the ODE Solver
x1=X_sol_ODESolver(:,1);
x2=X_sol_ODESolver(:,2);
x3=X_sol_ODESolver(:,3);
```

```
x4=X_sol_ODESolver(:,4);
Lam_x1=X_sol_ODESolver(:,5);
Lam_x2=X_sol_ODESolver(:,6);
Lam_x3=X_sol_ODESolver(:,7);
Lam_x4=X_sol_ODESolver(:,8);

%% Computing the Hamiltonian for getting H_tf

% Getting Boundary Values
x1_tf=x1(end);
x2_tf=x2(end);
x3_tf=x3(end);
x4_tf=x4(end);
Lam_x1_tf=Lam_x1(end);
Lam_x2_tf=Lam_x2(end);
Lam_x3_tf=Lam_x3(end);
Lam_x4_tf=Lam_x4(end);

% Computing U - Control at tf
u_tf=atan2(Lam_x4_tf,Lam_x3_tf);

a=10; % Constant

% Computing H(tf)
H_tf=(Lam_x1_tf*x3_tf) + (Lam_x2_tf*x4_tf) + (Lam_x3_tf*(a*cos(u_tf)))
    +...
    (Lam_x4_tf*(a*sin(u_tf)));

%% Computing the Error

% Initializing Error
Error=zeros(5,1);

% Creating Error Vector
Error(1)=x2_tf-5;
Error(2)=x3_tf-45;
Error(3)=x4_tf-0;
Error(4)=Lam_x1_tf-0;
Error(5)=H_tf+1;

end
```

1.2.3 Results

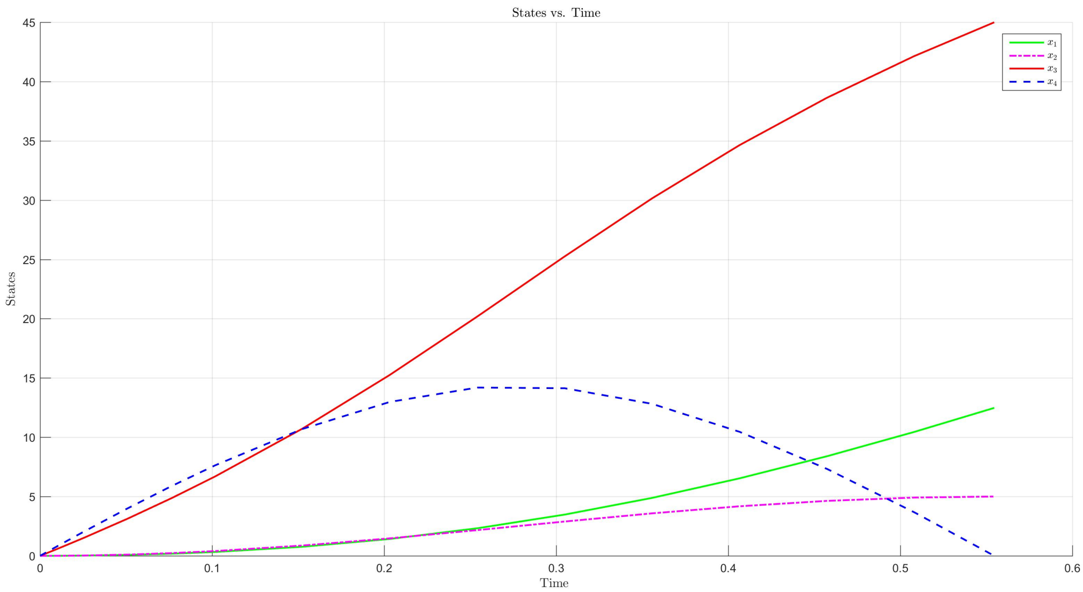

Figure 1: Linear Tangent Steering - HBVP - States

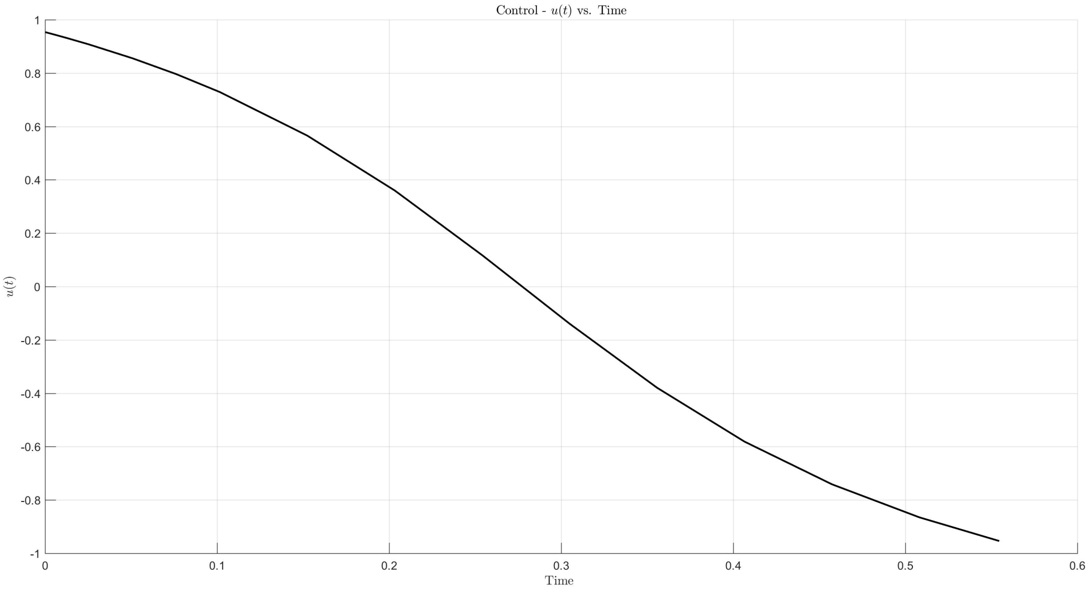

Figure 2: Linear Tangent Steering - HBVP - Control

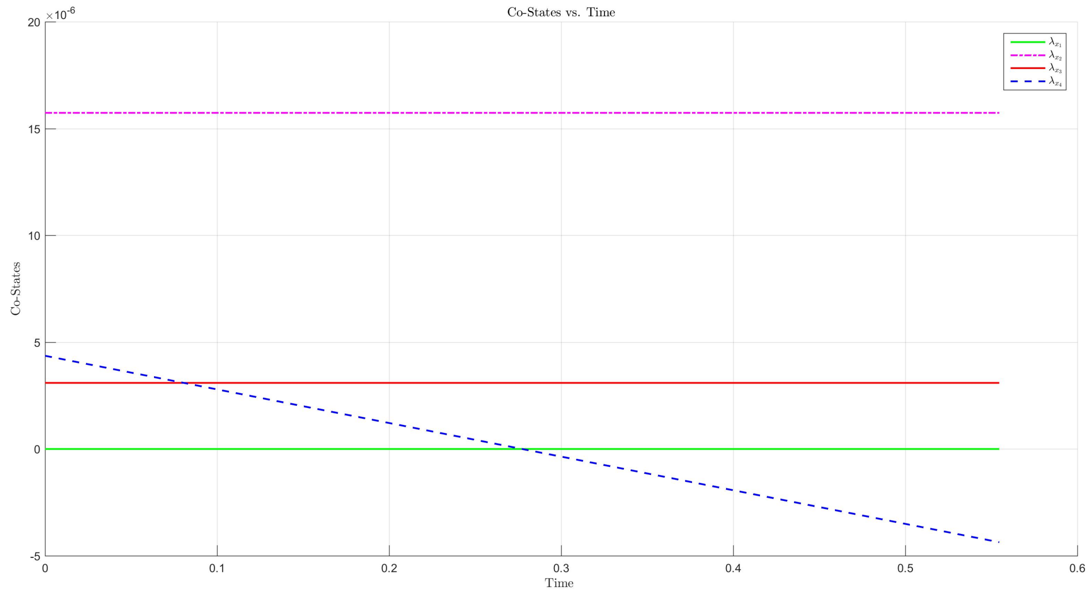

Figure 3: Linear Tangent Steering - HBVP - CoStates

1.3 Direct Method: Collocation

1.3.1 Formulation of the NLP

The general Nonlinear Program (NLP) is given as follows,
Minimize the cost functional,

$$J = F(\underline{\mathrm{Z}}), \tag{32}$$

Subject to the dynamic and path constraints constraints,

$$\mathrm{g}_{min} \leq \mathrm{g}(\underline{\mathrm{Z}}) \leq \mathrm{g}_{max}, \tag{33}$$

With the following state constraints,

$$\underline{\mathrm{Z}}_{min} \leq \underline{\mathrm{Z}} \leq \underline{\mathrm{Z}}_{max}, \tag{34}$$

For the Linear Tangent Steering Problem we have,

$$\underline{\mathrm{Z}} = \begin{bmatrix} \underline{\mathrm{X}}_1 \\ \underline{\mathrm{X}}_2 \\ \underline{\mathrm{X}}_3 \\ \underline{\mathrm{X}}_4 \\ \underline{\mathrm{U}} \\ t_0 \\ t_f \end{bmatrix} \tag{35}$$

Where,

$$\underline{\mathrm{X}}_1 = \begin{bmatrix} x_1(1) \\ \vdots \\ x_1(N+1) \end{bmatrix} \quad \underline{\mathrm{X}}_2 = \begin{bmatrix} x_2(1) \\ \vdots \\ x_2(N+1) \end{bmatrix} \quad \underline{\mathrm{X}}_3 = \begin{bmatrix} x_3(1) \\ \vdots \\ x_3(N+1) \end{bmatrix} \quad \underline{\mathrm{X}}_4 = \begin{bmatrix} x_4(1) \\ \vdots \\ x_4(N+1) \end{bmatrix} \quad \underline{\mathrm{U}} = \begin{bmatrix} u(1) \\ \vdots \\ u(N) \end{bmatrix} \tag{36}$$

Where N is the number of Legendre-Gauss-Radau (LGR) Points.
Now we have $g(\underline{\mathrm{Z}})$ as follows,

$$g(\underline{\mathrm{Z}}) = \begin{bmatrix} \triangle\underline{\mathrm{X}}_1 \\ \triangle\underline{\mathrm{X}}_2 \\ \triangle\underline{\mathrm{X}}_3 \\ \triangle\underline{\mathrm{X}}_4 \end{bmatrix} \tag{37}$$

The $\triangle\underline{\mathrm{X}}_1$, $\triangle\underline{\mathrm{X}}_2$, $\triangle\underline{\mathrm{X}}_3$ and $\triangle\underline{\mathrm{X}}_4$ are defined as follows,

$$
\begin{aligned}
\triangle\underline{\mathrm{X}}_1 &= D\underline{\mathrm{X}}_{1_{1:N+1}} - \frac{t_f}{2}\left[x_{3_{1:N+1}}\right] && = \underline{0}, \\
\triangle\underline{\mathrm{X}}_2 &= D\underline{\mathrm{X}}_{2_{1:N+1}} - \frac{t_f}{2}\left[x_{4_{1:N+1}}\right] && = \underline{0}, \\
\triangle\underline{\mathrm{X}}_3 &= D\underline{\mathrm{X}}_{3_{1:N+1}} - \frac{t_f}{2}\left[a.cos(u_{1:N})\right] && = \underline{0}, \\
\triangle\underline{\mathrm{X}}_4 &= D\underline{\mathrm{X}}_{4_{1:N+1}} - \frac{t_f}{2}\left[a.sin(u_{1:N})\right] && = \underline{0},
\end{aligned}
\tag{38}
$$

Where D is a differentiation matrix.

The g_{min} and g_{max} are constant vectors of zeros since all the equations in (38) are equality constraints with zeros on the LHS,

The $\underline{\mathrm{Z}}_{min}$ and $\underline{\mathrm{Z}}_{max}$ are constant vectors too but include the known boundary conditions and approximately correct lower and upper bounds for the decision vector contained in (35) as follows;

$$
\underline{\mathrm{Z}}_{min} = \begin{bmatrix} x_{1_{min}}(t_0) \\ x_{1_{min}}(t_1) \\ \vdots \\ x_{1_{min}}(t_f) \\ x_{2_{min}}(t_0) \\ x_{2_{min}}(t_1) \\ \vdots \\ x_{2_{min}}(t_f) \\ x_{3_{min}}(t_0) \\ x_{3_{min}}(t_1) \\ \vdots \\ x_{3_{min}}(t_f) \\ x_{4_{min}}(t_0) \\ x_{4_{min}}(t_1) \\ \vdots \\ x_{4_{min}}(t_f) \\ u_{min}(t_0) \\ u_{min}(t_1) \\ \vdots \\ u_{min}(t_f) \\ t_{0_{min}} \\ t_{f_{min}} \end{bmatrix} = \begin{bmatrix} x_1(t_0) \\ x_{1_{min}}(t_1) \\ \vdots \\ x_{1_{min}}(t_f) \\ x_2(t_0) \\ x_{2_{min}}(t_1) \\ \vdots \\ x_2(t_f) \\ x_3(t_0) \\ x_{3_{min}}(t_1) \\ \vdots \\ x_3(t_f) \\ x_4(t_0) \\ x_{4_{min}}(t_1) \\ \vdots \\ x_4(t_f) \\ u_{min}(t_0) \\ u_{min}(t_1) \\ \vdots \\ u_{min}(t_f) \\ t_0 \\ t_{f_{min}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 5 \\ 0 \\ 0 \\ \vdots \\ 45 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -\pi/2 \\ -\pi/2 \\ \vdots \\ -\pi/2 \\ 0 \\ 0 \end{bmatrix}
\tag{39}
$$

$$
\underline{Z}_{max} = \begin{bmatrix} x_{1_{max}}(t_0) \\ x_{1_{max}}(t_1) \\ \vdots \\ x_{1_{max}}(t_f) \\ x_{2_{max}}(t_0) \\ x_{2_{max}}(t_1) \\ \vdots \\ x_{2_{max}}(t_f) \\ x_{3_{max}}(t_0) \\ x_{3_{max}}(t_1) \\ \vdots \\ x_{3_{max}}(t_f) \\ x_{4_{max}}(t_0) \\ x_{4_{max}}(t_1) \\ \vdots \\ x_{4_{max}}(t_f) \\ u_{max}(t_0) \\ u_{max}(t_1) \\ \vdots \\ u_{max}(t_f) \\ t_{0_{max}} \\ t_{f_{max}} \end{bmatrix} = \begin{bmatrix} x_{1_{max}}(t_0) \\ x_{1_{max}}(t_1) \\ \vdots \\ x_{1_{max}}(t_f) \\ x_2(t_0) \\ x_{2_{max}}(t_1) \\ \vdots \\ x_2(t_f) \\ x_3(t_0) \\ x_{3_{max}}(t_1) \\ \vdots \\ x_3(t_f) \\ x_4(t_0) \\ x_{4_{max}}(t_1) \\ \vdots \\ x_4(t_f) \\ u_{max}(t_0) \\ u_{max}(t_1) \\ \vdots \\ u_{max}(t_f) \\ t_0 \\ t_{f_{max}} \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ \vdots \\ 100 \\ 0 \\ 100 \\ \vdots \\ 5 \\ 0 \\ 100 \\ \vdots \\ 45 \\ 0 \\ 100 \\ \vdots \\ 0 \\ \pi/2 \\ \pi/2 \\ \vdots \\ \pi/2 \\ 0 \\ 100 \end{bmatrix} \tag{40}
$$

Note: In equations (39) and (40) the vertical dots in the numerical vector represent values that are equal to the value before the dots begin.

Now the problem defined by the equations (32), (33) and (34) is solved using a NLP solver which in this case is IPOPT. Moreover, this NLP solver has two modes as follows,

- Full Newton: We have to provide Objective Function Gradient, Constraint Jacobian and Hessian of the NLP Lagrangian to the solver.
- Quasi-Newton: We have to provide Objective Function Gradient and Constraint Jacobian to the solver.

In our case, we solve our NLP in IPOPT using the Quasi-Newton mode and hence, provide the IPOPT solver with Objective Function Gradient and Constraint Jacobian computed by ADiGator (Algorithmic Differentiator) as follows,

The Objective Function Gradient is given as,

$$
\frac{\partial F}{\partial \underline{Z}} = \begin{bmatrix} \frac{\partial F}{\partial \underline{X}_1} & \frac{\partial F}{\partial \underline{X}_2} & \frac{\partial F}{\partial \underline{X}_3} & \frac{\partial F}{\partial \underline{X}_4} & \frac{\partial F}{\partial \underline{U}} & \frac{\partial F}{\partial t_0} & \frac{\partial F}{\partial t_f} \end{bmatrix} \tag{41}
$$

The Constraint Jacobian is given as,

$$\frac{\partial \underline{\mathrm{g}}}{\partial \underline{\mathrm{Z}}} = \begin{bmatrix} \dfrac{\partial \underline{\mathrm{g}}}{\partial \underline{\mathrm{X}}_1} & \dfrac{\partial \underline{\mathrm{g}}}{\partial \underline{\mathrm{X}}_2} & \dfrac{\partial \underline{\mathrm{g}}}{\partial \underline{\mathrm{X}}_3} & \dfrac{\partial \underline{\mathrm{g}}}{\partial \underline{\mathrm{X}}_4} & \dfrac{\partial \underline{\mathrm{g}}}{\partial \underline{\mathrm{U}}} & \dfrac{\partial \underline{\mathrm{g}}}{\partial t_0} & \dfrac{\partial \underline{\mathrm{g}}}{\partial t_f} \end{bmatrix} \tag{42}$$

Note: Each element in the LHS of equation (41) is a row vector, hence the LHs is a large row vector; and each element in the LHS of equation (42) is a column vector, hence the LHS is a large matrix.

1.3.2 MATLAB Code

```
%-------------------------------------------------------------------%
%                  Linear Tangent Steering Problem                  %

clear all;
close all;
clc;

%-------------------------------------------------------------------%
% 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 psStuff nstates ncontrols a
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    %
%-------------------------------------------------------------------%
x1_0          = 0;

x2_0          = 0;
```

```
x2_f        = 5;

x3_0        = 0;
x3_f        = 45;

x4_0        = 0;
x4_f        = 0;

x1min       = 0;            x1max       = 100;
x2min       = 0;            x2max       = 100;
x3min       = 0;            x3max       = 100;
x4min       = 0;            x4max       = 100;

umin        = -pi/2;        umax        = pi/2;

t0min      = 0;             t0max      = 0;
tfmin      = 0;             tfmax      = 100;

%-----------------------------------------------------------------------%
% 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 = 30;
%---------------------------------------------------------------------%
% 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) = x1_0; zx1max(1) = x1_0;

zx2min = x2min*ones(length(tau),1);
zx2max = x2max*ones(length(tau),1);
zx2min(1) = x2_0; zx2max(1) = x2_0;
zx2min(NLGR+1) = x2_f; zx2max(NLGR+1) = x2_f;

zx3min = x3min*ones(length(tau),1);
zx3max = x3max*ones(length(tau),1);
zx3min(1) = x3_0; zx3max(1) = x3_0;
zx3min(NLGR+1) = x3_f; zx3max(NLGR+1) = x3_f;

zx4min = x4min*ones(length(tau),1);
zx4max = x4max*ones(length(tau),1);
zx4min(1) = x4_0; zx4max(1) = x4_0;
zx4min(NLGR+1) = x4_f; zx4max(NLGR+1) = x4_f;

```

```
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 = [];
objMin = 0; objMax = inf;
Fmin = [objMin; defectMin; pathMin; eventMin];
Fmax = [objMax; defectMax; pathMax; eventMax];

%-------------------------------------------------------------------%
% Supply an initial guess for the NLP.                              %
%-------------------------------------------------------------------%
x1guess = x1_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x2guess = x2_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x3guess = x3_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x4guess = x4_0*ones(NLGR+1,1)+randn(NLGR+1,1);
uguess = ((umin-umax)/2)*ones(NLGR,1)+randn(NLGR,1);
t0guess = 0;
tfguess = 1+randn(1,1);

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('LinearTangentFun',{x});
S_jac  = output.JacobianStructure;
[iGfun,jGvar] = find(S_jac);

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

```

%-------------------------------------------------------------------%
% Set IPOPT callback functions
%-------------------------------------------------------------------%
funcs.objective   = @(Z)LinearTangentObj(Z);
funcs.gradient    = @(Z)LinearTangentGrd(Z);
funcs.constraints = @(Z)LinearTangentCon(Z);
funcs.jacobian    = @(Z)LinearTangentJac(Z);
funcs.jacobianstructure = @()LinearTangentJacPat(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 = 20000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['LinearTangent','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
%-------------------------------------------------------------------%
tic; % Timing the Process

[z, info] = ipopt(z0,funcs,options);

TimeTaken = toc;

%-------------------------------------------------------------------%
% 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; % Time for Plotting States
tLGR = t(1:end-1); % Time for Plotting Control

%-----------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                  %
% the defect constraints.                                         %
%-----------------------------------------------------------------%
multipliersDefects = Fmul(2:nstates*NLGR+1);
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];
lam_x1 = costate(:,1); lam_x2 = costate(:,2);
lam_x3 = costate(:,3); lam_x4 = costate(:,4);

%-----------------------------------------------------------------%
% plot results
%-----------------------------------------------------------------%
% Plotting States
figure(1)
hold on
grid on
plot(t,x1,'-g','LineWidth',1.5);
plot(t,x2,'-.m','LineWidth',1.5);
plot(t,x3,'-r','LineWidth',1.5);
```

```
plot(t,x4,'--b','LineWidth',1.5);
title('States vs. Time','Interpreter','latex');
xlabel('Time (sec)','Interpreter','latex');
ylabel('States','Interpreter','latex');
legend1=legend('$x_{1}$','$x_{2}$','$x_{3}$','$x_{4}$');
set(legend1,'Interpreter','latex');
hold off;

% Plotting Control
figure(2)
hold on
grid on
plot(tLGR,u,'-k','LineWidth',1.5);
title('Control - $u(t)$ vs. Time','Interpreter','latex');
xlabel('Time (sec)','Interpreter','latex');
ylabel('$u(t)$','Interpreter','latex');
hold off;

% Plotting Costates
figure(3)
hold on
grid on
plot(t,lam_x1,'-g','LineWidth',1.5);
plot(t,lam_x2,'-.m','LineWidth',1.5);
plot(t,lam_x3,'-r','LineWidth',1.5);
plot(t,lam_x4,'--b','LineWidth',1.5);
title('Co-States vs. Time','Interpreter','latex');
xlabel('Time (sec)','Interpreter','latex');
ylabel('Co-States','Interpreter','latex');
legend2=legend('$\lambda_{x_{1}}$','$\lambda_{x_{2}}$',...
    '$\lambda_{x_{3}}$','$\lambda_{x_{4}}$');
set(legend2,'Interpreter','latex');
hold off;

fprintf('Time taken to solve the Collocation Problem = %.4f',TimeTaken)
```

```matlab
function obj = LinearTangentObj(z)
% Computes the objective function of the problem

global psStuff nstates ncontrols a

%-------------------------------------------------------------------------%
% 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;

% Cost function
J   = tf;
obj = J;

end
```

```
function C = LinearTangentFun(z)

%-------------------------------------------------------------------------%
% Objective and constraint functions for the orbit-raising                %
% problem.  This function is designed to be used with the NLP             %
% solver SNOPT.                                                           %
%-------------------------------------------------------------------------%
%       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.                              %
%-----------------------------------------------------------------%
diffeqRHS = [x3, x4, a*cos(u), a*sin(u)];

%-----------------------------------------------------------------%
% 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 objective function plus constraint vector.         %
%------------------------------------------------------------------%
J   = tf;

C = [J; defects];
```

```
function constraints = LinearTangentCon(Z)
% computes the constraints

output      = LinearTangentFun(Z);
constraints = output;

end
```

```
function grd = LinearTangentGrd(Z)
% computes the gradient

output = LinearTangentObj_Jac(Z);
grd    = output;

end
```

```
function jac = LinearTangentJac(Z)
% computes the jacobian

[jac,~] = LinearTangentFun_Jac(Z);

end
```

```
function jacpat = LinearTangentJacPat(S_jac)
% computes the jacobian structure

jacpat = S_jac;

end
```

1.3.3 Results

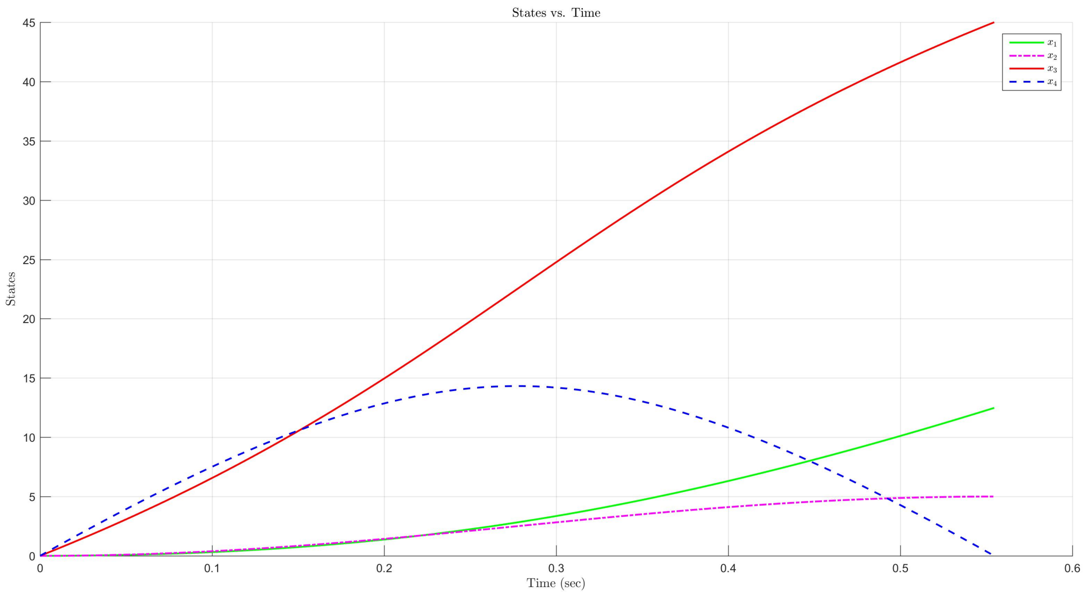

Figure 4: Linear Tangent Steering - Collocation - States

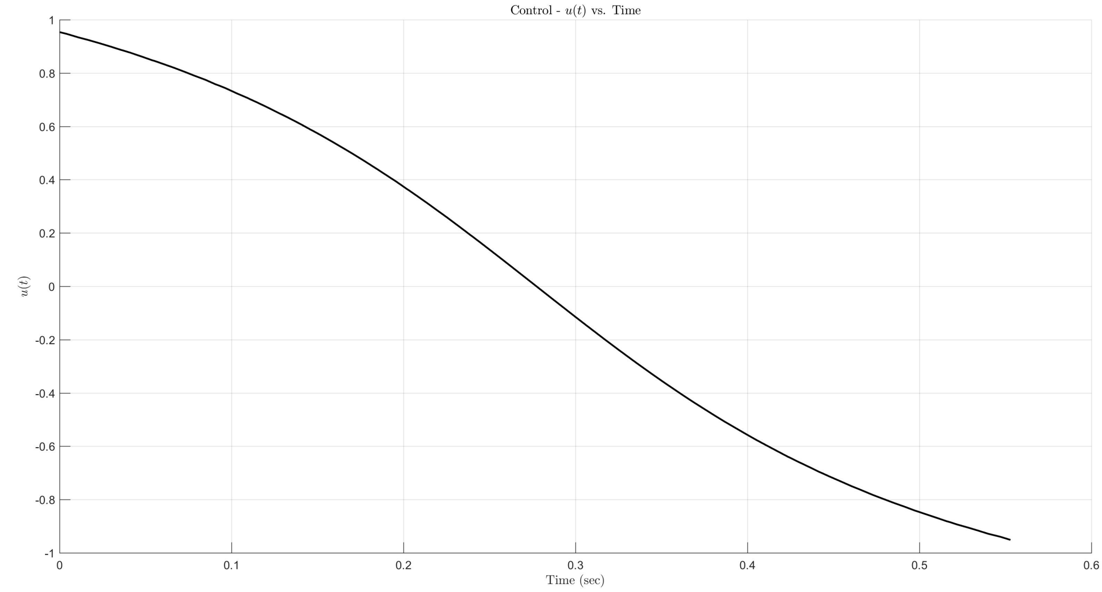

Figure 5: Linear Tangent Steering - Collocation - Control

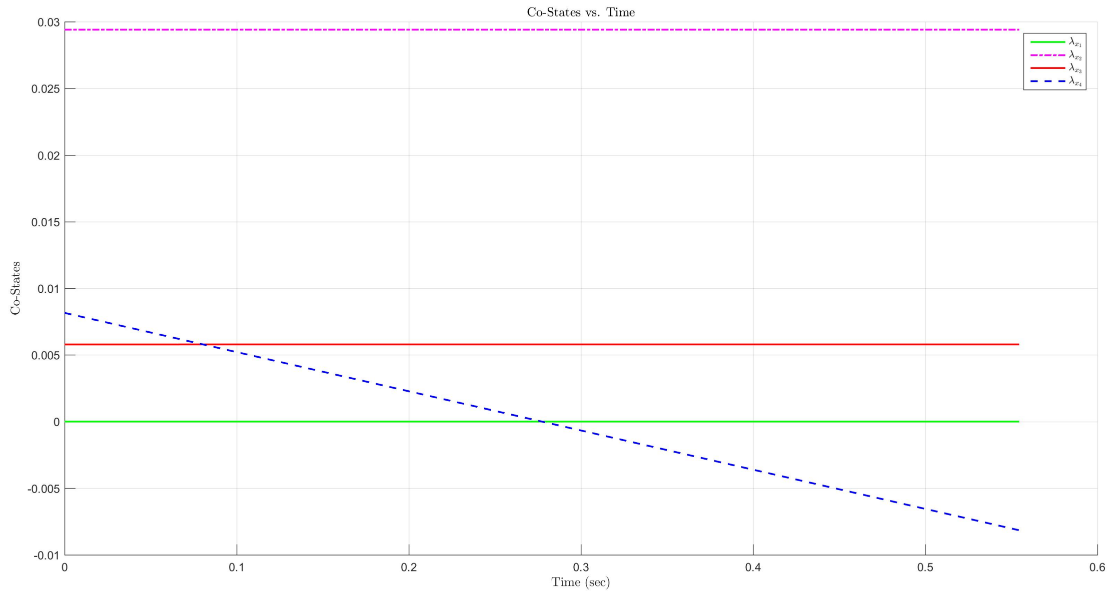

Figure 6: Linear Tangent Steering - Collocation - CoStates

1.4 Analysis

1.4.1 Quality of Numerical Approximations

1. For the Linear Tangent Steering problem the solutions from both the HBVP and Collocation methods are equivalent.

2. Even when initial conditions for both the HBVP and Collocation methods are changed the optimal solution remains the same; hence, showing the robustness of both the methods.

1.4.2 Key Computational Issues

1. The time taken to solve by the HBVP (1.2847 sec) is approximately three times the time taken by the Collocation method (0.4771 sec). Hence, Collocation method is computationally faster.

2. The Collocation method ends with the solver exiting with status of "Optimal solution found"; however the root finder in the HBVP method exits with the status "last step was ineffective", but when the TolFun tolerance is relaxed it is able to solve the equation .

3. Overall setting up the Collocation problem is easier as compared to HBVP; as one does not have to analytically solve for the first order optimality conditions. However, tools like ADiGator which compute the Gradient, Jacobian and Hessian of the Objective, Constraints and NLP Lagrangian respectively actually make the process of setting up the Collocation problem.

2 Advanced Problem: Robot Arm

2.1 Problem Formulation:

The Robot Arm Optimal Control Problem is as follows;

Minimize the cost functional,

$$J = t_f, \tag{43}$$

Subject to the dynamic constraints,

$$\begin{aligned}
\dot{x}_1 &= x_2, \\
\dot{x}_2 &= u_1/L, \\
\dot{x}_3 &= x_4, \\
\dot{x}_4 &= u_2/I_\theta, \\
\dot{x}_5 &= x_{6,} \\
\dot{x}_6 &= u_3/I_\phi,
\end{aligned} \tag{44}$$

With the control inequalities,,

$$|u_i| \leq 1 \quad , \quad (i = 1, 2, 3), \tag{45}$$

With the following boundary conditions,

$$\begin{aligned}
t_0 &= 0 & t_f &= \text{Free}, \\
x_1(0) &= 4.5 & x_1(t_f) &= 4.5, \\
x_2(0) &= 0 & x_2(t_f) &= 0, \\
x_3(0) &= 0 & x_3(t_f) &= 2\pi/3, \\
x_3(0) &= 0 & x_3(t_f) &= 0, \\
x_4(0) &= \pi/4 & x_4(t_f) &= \pi/4, \\
x_5(0) &= 0 & x_5(t_f) &= 0,
\end{aligned} \tag{46}$$

Where $L = 5$ and I_θ and I_ϕ are as follows,

$$I_\theta = \frac{1}{3}\left[(L - x_1)^3 + x_1^3\right], \tag{47}$$

$$I_\phi = I_\theta sin^2(x_5), \tag{48}$$

2.2 Direct Method: Collocation

2.2.1 Formulation of the NLP

The general Nonlinear Program (NLP) is given as follows,
Minimize the cost functional,

$$J = F(\underline{\mathrm{Z}}), \tag{49}$$

Subject to the dynamic and path constraints constraints,

$$\mathrm{g}_{min} \leq \mathrm{g}(\underline{\mathrm{Z}}) \leq \mathrm{g}_{max}, \tag{50}$$

With the following state constraints,

$$\underline{\mathrm{Z}}_{min} \leq \underline{\mathrm{Z}} \leq \underline{\mathrm{Z}}_{max}, \tag{51}$$

For the Linear Tangent Steering Problem we have,

$$\underline{\mathrm{Z}} = \begin{bmatrix} \underline{\mathrm{X}}_1 \\ \underline{\mathrm{X}}_2 \\ \underline{\mathrm{X}}_3 \\ \underline{\mathrm{X}}_4 \\ \underline{\mathrm{X}}_5 \\ \underline{\mathrm{X}}_6 \\ \underline{\mathrm{U}}_1 \\ \underline{\mathrm{U}}_2 \\ \underline{\mathrm{U}}_3 \\ t_0 \\ t_f \end{bmatrix} \tag{52}$$

Where,

$$\underline{\mathrm{X}}_1 = \begin{bmatrix} x_1(1) \\ \vdots \\ x_1(N+1) \end{bmatrix} \quad \underline{\mathrm{X}}_2 = \begin{bmatrix} x_2(1) \\ \vdots \\ x_2(N+1) \end{bmatrix} \quad \underline{\mathrm{X}}_3 = \begin{bmatrix} x_3(1) \\ \vdots \\ x_3(N+1) \end{bmatrix} \quad \underline{\mathrm{X}}_4 = \begin{bmatrix} x_4(1) \\ \vdots \\ x_4(N+1) \end{bmatrix} \quad \underline{\mathrm{X}}_5 = \begin{bmatrix} x_5(1) \\ \vdots \\ x_5(N+1) \end{bmatrix} \tag{53}$$

$$\underline{\mathrm{X}}_6 = \begin{bmatrix} x_6(1) \\ \vdots \\ x_6(N+1) \end{bmatrix} \qquad \underline{\mathrm{U}}_1 = \begin{bmatrix} u_1(1) \\ \vdots \\ u_1(N) \end{bmatrix} \qquad \underline{\mathrm{U}}_2 = \begin{bmatrix} u_2(1) \\ \vdots \\ u_2(N) \end{bmatrix} \qquad \underline{\mathrm{U}}_3 = \begin{bmatrix} u_4(1) \\ \vdots \\ u_4(N) \end{bmatrix} \tag{54}$$

Where N is the number of Legendre-Gauss-Radau (LGR) Points.

Now we have $g(\underline{Z})$ as follows,

$$g(\underline{Z}) = \begin{bmatrix} \triangle\underline{X}_1 \\ \triangle\underline{X}_2 \\ \triangle\underline{X}_3 \\ \triangle\underline{X}_4 \\ \triangle\underline{X}_5 \\ \triangle\underline{X}_6 \end{bmatrix} \tag{55}$$

The $\triangle\underline{X}_1$, $\triangle\underline{X}_2$, $\triangle\underline{X}_3$, $\triangle\underline{X}_4$, $\triangle\underline{X}_5$ and $\triangle\underline{X}_6$ are defined as follows,

$$\begin{aligned}
\triangle\underline{X}_1 &= D\underline{X}_{1_{1:N+1}} - \frac{t_f}{2}\left[x_{2_{1:N+1}}\right] &&= \underline{0}, \\
\triangle\underline{X}_2 &= D\underline{X}_{2_{1:N+1}} - \frac{t_f}{2}\left[u_{1_{1:N}}/L\right] &&= \underline{0}, \\
\triangle\underline{X}_3 &= D\underline{X}_{3_{1:N+1}} - \frac{t_f}{2}\left[x_{4_{1:N+1}}\right] &&= \underline{0}, \\
\triangle\underline{X}_4 &= D\underline{X}_{4_{1:N+1}} - \frac{t_f}{2}\left[u_{2_{1:N}}/I_\theta\right] &&= \underline{0}, \\
\triangle\underline{X}_5 &= D\underline{X}_{5_{1:N+1}} - \frac{t_f}{2}\left[x_{6_{1:N+1}}\right] &&= \underline{0}, \\
\triangle\underline{X}_6 &= D\underline{X}_{6_{1:N+1}} - \frac{t_f}{2}\left[u_{3_{1:N}}/I_\phi\right] &&= \underline{0},
\end{aligned} \tag{56}$$

Where D is a differentiation matrix and L, I_θ and I_ϕ are as defined in the previous section.

The g_{min} and g_{max} are constant vectors of zeros since all the equations in (56) are equality constraints with zeros on the LHS,

The $\underline{Z}_{min}$ and $\underline{Z}_{max}$ are constant vectors too but include the known boundary conditions and approximately correct lower and upper bounds for the decision vector contained in (52) as follows;

$$\underline{Z}_{min} = \begin{bmatrix} x_{1_{min}}(t_0) \\ x_{1_{min}}(t_1) \\ \vdots \\ x_{1_{min}}(t_f) \\ x_{2_{min}}(t_0) \\ x_{2_{min}}(t_1) \\ \vdots \\ x_{2_{min}}(t_f) \\ x_{3_{min}}(t_0) \\ x_{3_{min}}(t_1) \\ \vdots \\ x_{3_{min}}(t_f) \\ x_{4_{min}}(t_0) \\ x_{4_{min}}(t_1) \\ \vdots \\ x_{4_{min}}(t_f) \\ x_{5_{min}}(t_0) \\ x_{5_{min}}(t_1) \\ \vdots \\ x_{5_{min}}(t_f) \\ x_{6_{min}}(t_0) \\ x_{6_{min}}(t_1) \\ \vdots \\ x_{6_{min}}(t_f) \\ u_{1_{min}}(t_0) \\ u_{1_{min}}(t_1) \\ \vdots \\ u_{1_{min}}(t_f) \\ u_{2_{min}}(t_0) \\ u_{2_{min}}(t_1) \\ \vdots \\ u_{2_{min}}(t_f) \\ u_{3_{min}}(t_0) \\ u_{3_{min}}(t_1) \\ \vdots \\ u_{3_{min}}(t_f) \\ t_{0_{min}} \\ t_{f_{min}} \end{bmatrix} = \begin{bmatrix} x_1(t_0) \\ x_{1_{min}}(t_1) \\ \vdots \\ x_1(t_f) \\ x_2(t_0) \\ x_{2_{min}}(t_1) \\ \vdots \\ x_2(t_f) \\ x_3(t_0) \\ x_{3_{min}}(t_1) \\ \vdots \\ x_3(t_f) \\ x_4(t_0) \\ x_{4_{min}}(t_1) \\ \vdots \\ x_4(t_f) \\ x_5(t_0) \\ x_{5_{min}}(t_1) \\ \vdots \\ x_5(t_f) \\ x_6(t_0) \\ x_{6_{min}}(t_1) \\ \vdots \\ x_6(t_f) \\ u_{1_{min}}(t_0) \\ u_{1_{min}}(t_1) \\ \vdots \\ u_{1_{min}}(t_f) \\ u_{2_{min}}(t_0) \\ u_{2_{min}}(t_1) \\ \vdots \\ u_{2_{min}}(t_f) \\ u_{3_{min}}(t_0) \\ u_{3_{min}}(t_1) \\ \vdots \\ u_{3_{min}}(t_f) \\ t_0 \\ t_{f_{min}} \end{bmatrix} = \begin{bmatrix} 4.5 \\ 0 \\ \vdots \\ 4.5 \\ 0 \\ -100 \\ \vdots \\ 0 \\ 0 \\ -\pi \\ \vdots \\ 2\pi/3 \\ 0 \\ -100 \\ \vdots \\ 0 \\ \pi/4 \\ 0 \\ \vdots \\ \pi/4 \\ 0 \\ -100 \\ \vdots \\ 0 \\ -1 \\ -1 \\ \vdots \\ -1 \\ -1 \\ -1 \\ \vdots \\ -1 \\ -1 \\ -1 \\ \vdots \\ -1 \\ 0 \\ 0 \end{bmatrix} \quad (57)$$

$$\underline{Z}_{max} = \begin{bmatrix} x_{1_{max}}(t_0) \\ x_{1_{max}}(t_1) \\ \vdots \\ x_{1_{max}}(t_f) \\ x_{2_{max}}(t_0) \\ x_{2_{max}}(t_1) \\ \vdots \\ x_{2_{max}}(t_f) \\ x_{3_{max}}(t_0) \\ x_{3_{max}}(t_1) \\ \vdots \\ x_{3_{max}}(t_f) \\ x_{4_{max}}(t_0) \\ x_{4_{max}}(t_1) \\ \vdots \\ x_{4_{max}}(t_f) \\ x_{5_{max}}(t_0) \\ x_{5_{max}}(t_1) \\ \vdots \\ x_{5_{max}}(t_f) \\ x_{6_{max}}(t_0) \\ x_{6_{max}}(t_1) \\ \vdots \\ x_{6_{max}}(t_f) \\ u_{1_{max}}(t_0) \\ u_{1_{max}}(t_1) \\ \vdots \\ u_{1_{max}}(t_f) \\ u_{2_{max}}(t_0) \\ u_{2_{max}}(t_1) \\ \vdots \\ u_{2_{max}}(t_f) \\ u_{3_{max}}(t_0) \\ u_{3_{max}}(t_1) \\ \vdots \\ u_{3_{max}}(t_f) \\ t_{0_{max}} \\ t_{f_{max}} \end{bmatrix} = \begin{bmatrix} x_1(t_0) \\ x_{1_{max}}(t_1) \\ \vdots \\ x_1(t_f) \\ x_2(t_0) \\ x_{2_{max}}(t_1) \\ \vdots \\ x_2(t_f) \\ x_3(t_0) \\ x_{3_{max}}(t_1) \\ \vdots \\ x_3(t_f) \\ x_4(t_0) \\ x_{4_{max}}(t_1) \\ \vdots \\ x_4(t_f) \\ x_5(t_0) \\ x_{5_{max}}(t_1) \\ \vdots \\ x_5(t_f) \\ x_6(t_0) \\ x_{6_{max}}(t_1) \\ \vdots \\ x_6(t_f) \\ u_{1_{max}}(t_0) \\ u_{1_{max}}(t_1) \\ \vdots \\ u_{1_{max}}(t_f) \\ u_{2_{max}}(t_0) \\ u_{2_{max}}(t_1) \\ \vdots \\ u_{2_{max}}(t_f) \\ u_{3_{max}}(t_0) \\ u_{3_{max}}(t_1) \\ \vdots \\ u_{3_{max}}(t_f) \\ t_0 \\ t_{f_{max}} \end{bmatrix} = \begin{bmatrix} 4.5 \\ 0 \\ \vdots \\ 4.5 \\ 0 \\ 100 \\ \vdots \\ 0 \\ 0 \\ \pi \\ \vdots \\ 2\pi/3 \\ 0 \\ 100 \\ \vdots \\ 0 \\ \pi/4 \\ 100 \\ \vdots \\ \pi/4 \\ 0 \\ 100 \\ \vdots \\ 0 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 0 \\ 100 \end{bmatrix} \tag{58}$$

Note: In equations (57) and (58) the vertical dots in the numerical vector represent values that are equal to the value before the dots begin.

Now the problem defined by the equations (49), (50) and (51) is solved using a NLP solver which in this case is IPOPT. Moreover, this NLP solver has two modes as follows,

- Full Newton: We have to provide Objective Function Gradient, Constraint Jacobian and Hessian of the NLP Lagrangian to the solver.
- Quasi-Newton: We have to provide Objective Function Gradient and Constraint Jacobian to the solver.

In our case, we solve our NLP in IPOPT using the Quasi-Newton mode and hence, provide the IPOPT solver with Objective Function Gradient and Constraint Jacobian computed by ADiGator (Algorithmic Differentiator) as follows,

The Objective Function Gradient is given as,

$$\frac{\partial F}{\partial \underline{\mathrm{Z}}} = \begin{bmatrix} \frac{\partial F}{\partial \underline{\mathrm{X}}_1} & \frac{\partial F}{\partial \underline{\mathrm{X}}_2} & \frac{\partial F}{\partial \underline{\mathrm{X}}_3} & \frac{\partial F}{\partial \underline{\mathrm{X}}_4} & \frac{\partial F}{\partial \underline{\mathrm{X}}_5} & \frac{\partial F}{\partial \underline{\mathrm{X}}_6} & \frac{\partial F}{\partial \underline{\mathrm{U}}_1} & \frac{\partial F}{\partial \underline{\mathrm{U}}_2} & \frac{\partial F}{\partial \underline{\mathrm{U}}_3} & \frac{\partial F}{\partial t_0} & \frac{\partial F}{\partial t_f} \end{bmatrix} \tag{59}$$

The Constraint Jacobian is given as,

$$\frac{\partial \mathrm{g}}{\partial \underline{\mathrm{Z}}} = \begin{bmatrix} \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{X}}_1} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{X}}_2} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{X}}_3} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{X}}_4} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{X}}_5} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{X}}_6} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{U}}_1} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{U}}_2} & \frac{\partial \mathrm{g}}{\partial \underline{\mathrm{U}}_3} & \frac{\partial \mathrm{g}}{\partial t_0} & \frac{\partial \mathrm{g}}{\partial t_f} \end{bmatrix} \tag{60}$$

Note: Each element in the LHS of equation (59) is a row vector, hence the LHs is a large row vector; and each element in the LHS of equation (60) is a column vector, hence the LHS is a large matrix.

2.3 MATLAB Code

```
%---------------------------------------------------------------------------%
%                         Robot Arm Problem                                 %

clear all;
close all;
clc;

%---------------------------------------------------------------------------%
% 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/I_Theta                                              %
% dx5/dt          = x6                                                      %
% dx6/dt          = au3/I_Phi                                               %
%---------------------------------------------------------------------------%
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                        %
%---------------------------------------------------------------------------%
global psStuff nstates ncontrols L
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            %
%---------------------------------------------------------------------------%
x1_0          = 4.5;
```

```
x1_f        = 4.5;

x2_0        = 0;
x2_f        = 0;

x3_0        = 0;
x3_f        = 2*pi/3;

x4_0        = 0;
x4_f        = 0;

x5_0        = pi/4;
x5_f        = pi/4;

x6_0        = 0;
x6_f        = 0;

x1min       = 0;            x1max       = L;
x2min       = -100;         x2max       = 100;
x3min       = -pi;          x3max       = pi;
x4min       = -100;         x4max       = 100;
x5min       = 0;            x5max       = pi;
x6min       = -100;         x6max       = 100;

u1min       = -1;           u1max       = 1;
u2min       = -1;           u2max       = 1;
u3min       = -1;           u3max       = 1;

t0min      = 0;          t0max      = 0;
tfmin      = 0;          tfmax      = 100;

%-------------------------------------------------------------------%
% 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 = 10;
%------------------------------------------------------------------%
% 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) = x1_0; zx1max(1) = x1_0;
zx1min(NLGR+1) = x1_f; zx1max(NLGR+1) = x1_f;

zx2min = x2min*ones(length(tau),1);
```

```
zx2max = x2max*ones(length(tau),1);
zx2min(1) = x2_0; zx2max(1) = x2_0;
zx2min(NLGR+1) = x2_f; zx2max(NLGR+1) = x2_f;

zx3min = x3min*ones(length(tau),1);
zx3max = x3max*ones(length(tau),1);
zx3min(1) = x3_0; zx3max(1) = x3_0;
zx3min(NLGR+1) = x3_f; zx3max(NLGR+1) = x3_f;

zx4min = x4min*ones(length(tau),1);
zx4max = x4max*ones(length(tau),1);
zx4min(1) = x4_0; zx4max(1) = x4_0;
zx4min(NLGR+1) = x4_f; zx4max(NLGR+1) = x4_f;

zx5min = x5min*ones(length(tau),1);
zx5max = x5max*ones(length(tau),1);
zx5min(1) = x5_0; zx5max(1) = x5_0;
zx5min(NLGR+1) = x5_f; zx5max(NLGR+1) = x5_f;

zx6min = x6min*ones(length(tau),1);
zx6max = x6max*ones(length(tau),1);
zx6min(1) = x6_0; zx6max(1) = x6_0;
zx6min(NLGR+1) = x6_f; zx6max(NLGR+1) = x6_f;

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 = [];
objMin = 0; objMax = inf;
Fmin = [objMin; defectMin; pathMin; eventMin];
Fmax = [objMax; defectMax; pathMax; eventMax];

%-------------------------------------------------------------------%
% Supply an initial guess for the NLP.                              %
%-------------------------------------------------------------------%
x1guess = x1_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x2guess = x2_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x3guess = x3_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x4guess = x4_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x5guess = x5_0*ones(NLGR+1,1)+randn(NLGR+1,1);
x6guess = x6_0*ones(NLGR+1,1);
u1guess = ((u1min+u1max)/2)*ones(NLGR,1)+randn(NLGR,1);
u2guess = ((u2min+u2max)/2)*ones(NLGR,1)+randn(NLGR,1);
u3guess = ((u3min+u3max)/2)*ones(NLGR,1)+randn(NLGR,1);
t0guess = 0;
tfguess = 1+randn(1,1);
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 = 'mumps';
options.ipopt.max_iter = 20000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['LinearTangent','IPOPTinfo.txt'];
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
%-------------------------------------------------------------------------%
tic; % Timing the Process

[z, info] = ipopt(z0,funcs,options);

TimeTaken = toc;

%-------------------------------------------------------------------------%
% 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; % Time for Plotting States
tLGR = t(1:end-1); % Time for Plotting Control

%-----------------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                        %
% the defect constraints.                                               %
%-----------------------------------------------------------------------%
multipliersDefects = Fmul(2:nstates*NLGR+1);
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];
lam_x1 = costate(:,1); lam_x2 = costate(:,2);
lam_x3 = costate(:,3); lam_x4 = costate(:,4);
lam_x5 = costate(:,3); lam_x6 = costate(:,4);

%-----------------------------------------------------------------------%
% plot results
%-----------------------------------------------------------------------%
% Plotting States
figure(1)
hold on
grid on
plot(t,x1,'-g','LineWidth',1.5);
plot(t,x2,'-.m','LineWidth',1.5);
plot(t,x3,'-r','LineWidth',1.5);
plot(t,x4,'--b','LineWidth',1.5);
plot(t,x5,'-c','LineWidth',1.5);
plot(t,x6,'--m','LineWidth',1.5);
```

```
title('States vs. Time','Interpreter','latex');
xlabel('Time (sec)','Interpreter','latex');
ylabel('States','Interpreter','latex');
legend1=legend('$x_{1}$','$x_{2}$','$x_{3}$','$x_{4}$','$x_{5}$',...
    '$x_{6}$');
set(legend1,'Interpreter','latex');
hold off;

% Plotting Control
figure(2)
hold on
grid on
plot(tLGR,u1,'-r','LineWidth',1.5);
plot(tLGR,u2,'--b','LineWidth',1.5);
plot(tLGR,u3,'-.g','LineWidth',1.5);
title('Control - $u(t)$ vs. Time','Interpreter','latex');
xlabel('Time (sec)','Interpreter','latex');
ylabel('$u(t)$','Interpreter','latex');
legend1=legend('$u_{1}$','$u_{2}$','$u_{3}$');
set(legend1,'Interpreter','latex');
hold off;

% Plotting Costates
figure(3)
hold on
grid on
plot(t,lam_x1,'-g','LineWidth',1.5);
plot(t,lam_x2,'-.m','LineWidth',1.5);
plot(t,lam_x3,'-r','LineWidth',1.5);
plot(t,lam_x4,'--b','LineWidth',1.5);
plot(t,lam_x5,'-c','LineWidth',1.5);
plot(t,lam_x6,'--m','LineWidth',1.5);
title('Co-States vs. Time','Interpreter','latex');
xlabel('Time (sec)','Interpreter','latex');
ylabel('Co-States','Interpreter','latex');
legend2=legend('$\lambda_{x_{1}}$','$\lambda_{x_{2}}$'...
    ,'$\lambda_{x_{3}}$','$\lambda_{x_{4}}$','$\lambda_{x_{5}}$'...
    ,'$\lambda_{x_{6}}$');
set(legend2,'Interpreter','latex');
hold off;

fprintf('Time taken to solve the Collocation Problem = %.4f',TimeTaken)
```

```
function obj = RobotArmObj(z)
% Computes the objective function of the problem

global psStuff nstates ncontrols L

%-------------------------------------------------------------------------%
% 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);

% Cost function
J   = tf;
obj = J;

end
```

```
function C = RobotArmFun(z)

%-------------------------------------------------------------------------%
% Objective and constraint functions for the orbit-raising                %
% problem.  This function is designed to be used with the NLP             %
% solver SNOPT.                                                           %
%-------------------------------------------------------------------------%
%       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.                              %
%-----------------------------------------------------------------%

% Basic Computation
I_Phi=(1/3)*((L-x1).^3+x1.^3);
I_Theta=I_Phi.*sin(x5).^2;

diffeqRHS = [x2, u1/L, x4, u2./I_Theta, x6, u3./I_Phi];

%-----------------------------------------------------------------%
```

```
% 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 objective function plus constraint vector.        %
%-----------------------------------------------------------------%
J   = tf;

C = [J; 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
```

2.4 Results

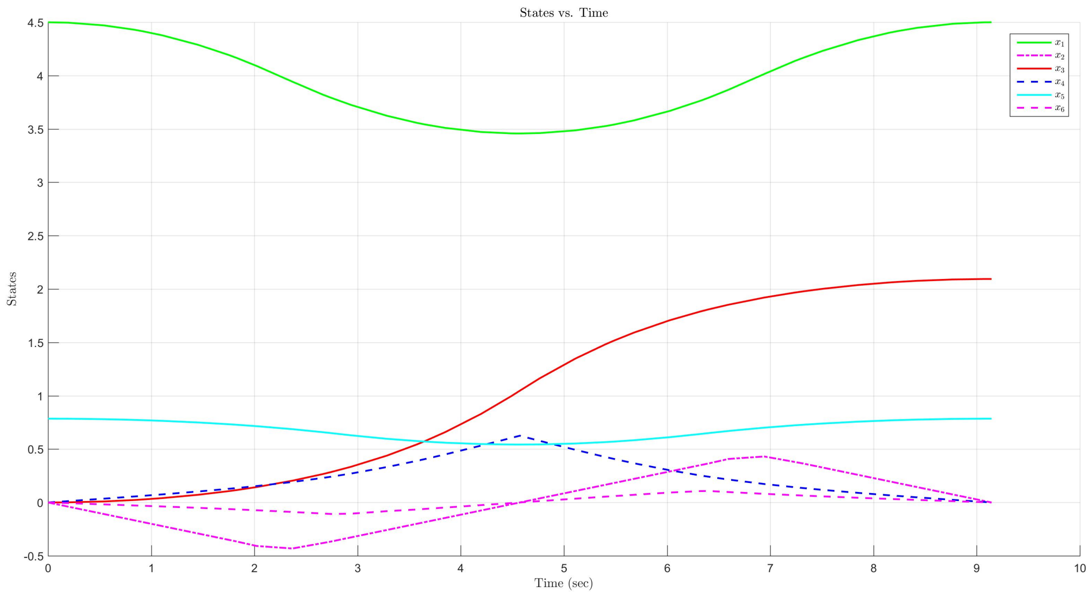

Figure 7: Robot Arm - Collocation - States

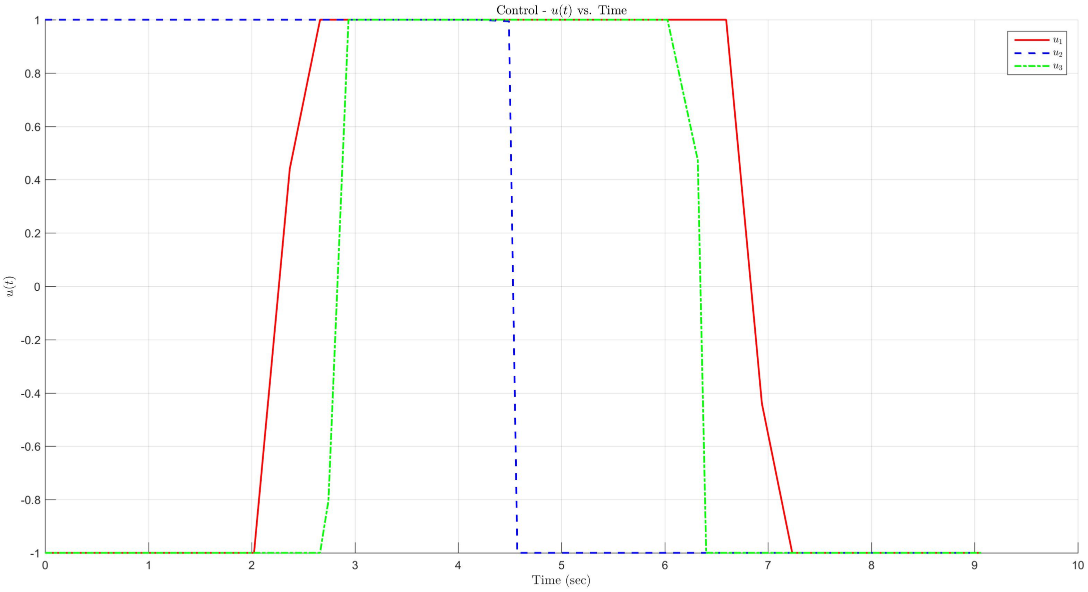

Figure 8: Robot Arm - Collocation - Controls

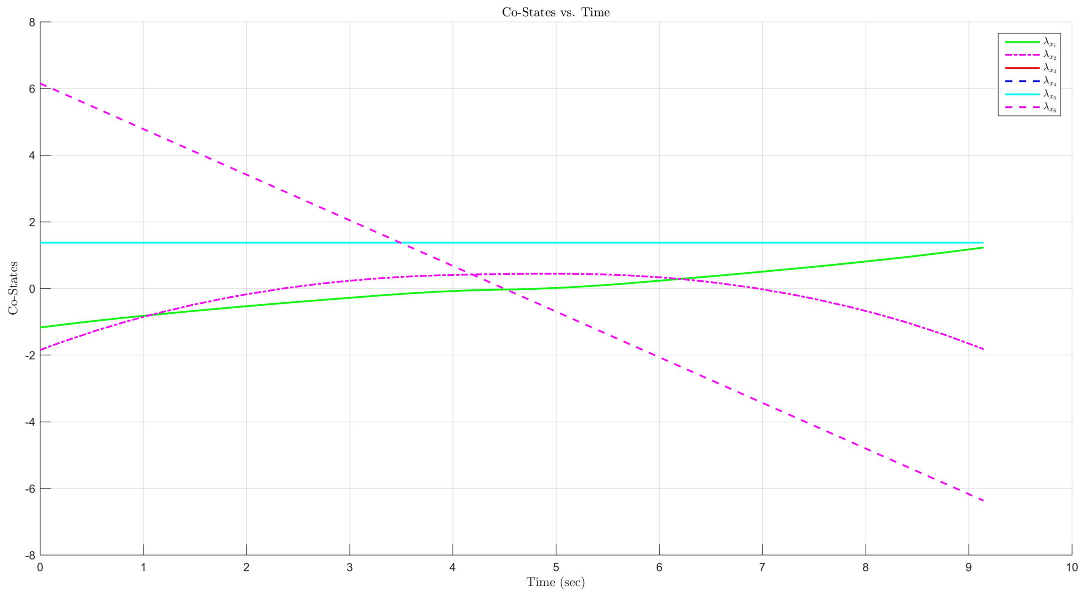

Figure 9: Robot Arm - Collocation - CoStates

2.5 Analysis

2.5.1 Proximity of Numerical Solutions to Optimal Solutions

The numerical solution found is near to the optimal solution; as, even when the initial conditions are changed the NLP solver gives the same numerical solution. Hence, it shows that this method is robust.

2.5.2 Computational Efficiency of the Numerical Method

The time taken by this method increases as we increase the degree of the polynomial and/or the number of mesh intervals. Also, in particular to IPOPT the ma57 solver (0.2856 sec) is faster as compared to the mumps solver (0.3761 sec).

2.5.3 Limitation of the Numerical Method

The limitation of this numerical method is that the true global minimum is not guaranteed, and we usually find a local optimal solution to our problem. Moreover, this method is not usually robust to changes in the initial guess of the decision variables. Also, for higher accuracy and speed higher order numerical derivatives are required which are complicated to calculate even with algorithmic differentiators.

2.5.4 The Ideal Numerical Method

An ideal numerical method would be one which is robust to the initial guesses, utilize lower order numerical derivatives and still guarantee global optimum solution.