

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	3
1.1	Problem Formulation:	3
1.2	Indirect Method: Hamiltonian Boundary Value Problem [HBVP]	4
1.2.1	Optimal Solution: Using Optimal Control Theory	4
1.2.2	MATLAB Code	11
1.2.3	Results	18
1.3	Direct Method: Collocation	20
1.3.1	Formulation of the NLP	20
1.3.2	MATLAB Code	24
1.3.3	Results	37
1.4	Analysis	39
1.4.1	Quality of Numerical Approximations	39
1.4.2	Key Computational Issues	39
2	Advanced Problem: Robot Arm	40
2.1	Problem Formulation:	40
2.2	Direct Method: Collocation	41
2.2.1	Formulation of the NLP	41
2.3	MATLAB Code	46
2.4	Results	60
2.5	Analysis	62
2.5.1	Proximity of Numerical Solutions to Optimal Solutions	62
2.5.2	Computational Efficiency of the Numerical Method	62
2.5.3	Limitation of the Numerical Method	62
2.5.4	The Ideal Numerical Method	62

1 Elementary Problem: Linear Tangent Steering

1.1 Problem Formulation:

The Linear Tangent Steering Optimal Control Problem is as follows;

Minimize the cost functional,

$$J = t_f, \quad (1)$$

Subject to the dynamic constraints,

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

With the following boundary conditions,

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

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,

$M = \text{Meyer Cost}$

$L = \text{Lagrange Cost}$

Now we know that,

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

Creating the Hamiltonian,

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

Where,

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

Hence the Hamiltonian is as follows,

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

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

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

$$(8)$$

We get,

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

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

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

We get,

$$\begin{aligned} \dot{\lambda}_{x_1} &= - \frac{\partial H}{\partial x_1} &= 0, \\ \dot{\lambda}_{x_2} &= - \frac{\partial H}{\partial x_2} &= 0, \\ \dot{\lambda}_{x_3} &= - \frac{\partial H}{\partial x_3} &= -\lambda_{x_1}, \\ \dot{\lambda}_{x_4} &= - \frac{\partial H}{\partial x_4} &= -\lambda_{x_2}, \end{aligned} \quad (11)$$

Now, using 1st Order Optimality Conditions for H ,

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

We get,

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

Therefore, we have,

$$u = \tan^{-1} \left[\frac{\lambda_{x_4}}{\lambda_{x_3}} \right] \quad (13)$$

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

$$\left[\frac{\partial M}{\partial \underline{x}(t_f)} - \nu^T \frac{\partial \underline{b}}{\partial \underline{x}(t_f)} - \underline{\lambda}^T(t_f) \right] = \underline{0}^T \quad (14)$$

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

Where,

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

and,

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

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

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

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

$$\nu^T \frac{\partial \underline{b}}{\partial \underline{x}(t_f)} = [0 \ \nu_5 \ \nu_6 \ \nu_7],\tag{18}$$

Now, we have,

$$\begin{aligned}& \left[\frac{\partial M}{\partial \underline{x}(t_f)} - \nu^T \frac{\partial \underline{b}}{\partial \underline{x}(t_f)} - \underline{\lambda}^T(t_f) \right] = \underline{0}^T \\ & [0 \ 0 \ 0 \ 0] - [0 \ \nu_5 \ \nu_6 \ \nu_7] - [\lambda_{x_1}(t_f) \ \lambda_{x_2}(t_f) \ \lambda_{x_3}(t_f) \ \lambda_{x_4}(t_f)] = \underline{0}^T\end{aligned}$$

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,

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

$$\frac{\partial b}{\partial t_f} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad (22)$$

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

Now, we have,

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

Therefore, we get,

$$H(t_f) = -1, \quad (24)$$

Formulating the HBVP,
Known Initial Conditions are as follows,

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

$$\begin{aligned}\lambda_{x_1}(0) &= 0, \\t_0 &= 0, \end{aligned} \tag{26}$$

Unknown Initial Conditions are as follows,

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

$$t_f = 0, \tag{28}$$

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,

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

1.2.2 MATLAB Code

```

1 %% Main File : Linear Tangent Steering : Indirect Shooting
2
3 clear all;
4 clc;
5 close all;
6
7 %% Initial Conditions
8
9 % Known Initial Conditions for the ODE Solver
10 x1_0=0;
11 x2_0=0;
12 x3_0=0;
13 x4_0=0;
14 Lam_x1_0=0;
15 t0_0=0;
16
17 X0_Known_ODESolver=[x1_0;x2_0;x3_0;x4_0;Lam_x1_0;t0_0];
18
19 % Unknown Initial Conditions for the Root Finder
20 Lam_x2_0=randn(1);
21 Lam_x3_0=randn(1);
22 Lam_x4_0=randn(1);
23 tf_0=1;
24
25 X0_RootFinder=[Lam_x2_0;Lam_x3_0;Lam_x4_0;tf_0];
26
27 %% Root Finder
28
29 % Options for the Root Finder
30 Options=optimset('Display','iter','TolFun',1e-5);
31
32 % Calling the Root Finder
33 tic; % Timing the Process
34
35 [X_Sol_RootFinder,fval,exitflag,output] = ...
36     fsolve(@LinearTangent_ODESolver,X0_RootFinder,Options);
37
38 TimeTaken = toc;
39
40 %% Solving the ODE with Optimal Initial Conditions
41
42 % Inital Conditions
43

```

```

44 % Known Initial Conditions for the ODE Solver
45 x1_0=0;
46 x2_0=0;
47 x3_0=0;
48 x4_0=0;
49 Lam_x1_0=0;
50 t0_0=0;
51
52 % Initial Conditions from the Root Finder
53 Lam_x2_0=X_sol_RootFinder(1);
54 Lam_x3_0=X_sol_RootFinder(2);
55 Lam_x4_0=X_sol_RootFinder(3);
56 tf_0=X_sol_RootFinder(4);
57
58 % Complete Initial Conditions for the ODE Solver
59 X0_ODESolver = [x1_0;x2_0;x3_0;x4_0;Lam_x1_0;Lam_x2_0;Lam_x3_0;Lam_x4_0];
60
61 % Creating Time Span for Integration
62 tf=tf_0;
63 T_Span_ODESolver=[0,tf];
64
65 % Options for the ODE Solver
66 Options=optimset('Display','Iter','TolFun',1e-3,'TolX',1e-3);
67
68 % Calling the ODE Solver
69 [t_ODESolver,X_sol_ODESolver]=ode113(@LinearTangent_ODEEquations,...
70     T_Span_ODESolver,X0_ODESolver,Options);
71
72 % Getting the Outputs from the ODE Solver
73 x1=X_sol_ODESolver(:,1);
74 x2=X_sol_ODESolver(:,2);
75 x3=X_sol_ODESolver(:,3);
76 x4=X_sol_ODESolver(:,4);
77 Lam_x1=X_sol_ODESolver(:,5);
78 Lam_x2=X_sol_ODESolver(:,6);
79 Lam_x3=X_sol_ODESolver(:,7);
80 Lam_x4=X_sol_ODESolver(:,8);
81
82 % Computing Control
83
84 Control=atan2(Lam_x4,Lam_x3);
85
86 %% Plotting Results
87
88 % Plotting States

```

```

89 figure(1)
90 hold on
91 grid on
92 plot(t_ODESolver,x1,'-g','LineWidth',1.5);
93 plot(t_ODESolver,x2,'-m','LineWidth',1.5);
94 plot(t_ODESolver,x3,'-r','LineWidth',1.5);
95 plot(t_ODESolver,x4,'-b','LineWidth',1.5);
96 title('States vs. Time','Interpreter','latex');
97 xlabel('Time','Interpreter','latex');
98 ylabel('States','Interpreter','latex');
99 legend1=legend('$x_{1}$','$x_{2}$','$x_{3}$','$x_{4}$');
100 set(legend1,'Interpreter','latex');
101 hold off;
102
103 % Plotting Control
104 figure(2)
105 hold on
106 grid on
107 plot(t_ODESolver,Control,'-k','LineWidth',1.5);
108 title('Control - $u(t)$ vs. Time','Interpreter','latex');
109 xlabel('Time','Interpreter','latex');
110 ylabel('$u(t)$','Interpreter','latex');
111 hold off;
112
113 % Plotting Costates
114 figure(3)
115 hold on
116 grid on
117 plot(t_ODESolver,Lam_x1,'-g','LineWidth',1.5);
118 plot(t_ODESolver,Lam_x2,'-m','LineWidth',1.5);
119 plot(t_ODESolver,Lam_x3,'-r','LineWidth',1.5);
120 plot(t_ODESolver,Lam_x4,'-b','LineWidth',1.5);
121 title('Co-States vs. Time','Interpreter','latex');
122 xlabel('Time','Interpreter','latex');
123 ylabel('Co-States','Interpreter','latex');
124 legend2=legend('$\lambda_{x_{1}}$','$\lambda_{x_{2}}$','$\lambda_{x_{3}}$','$\lambda_{x_{4}}$');
125 set(legend2,'Interpreter','latex');
126 hold off;
127
128 fprintf('Time taken to solve the HBVP = %.4f',TimeTaken)

```

```

1  function [ Equation_Derivative ] = LinearTangent_ODEEquations(
    T_Span_ODESolver , X0_ODESolver )
2
3  %% ODE Equations : Problem 1 – Part 2
4
5  %% Getting Required Values from the incoming Vectors
6
7  % From T_Span_ODESolver
8  t=T_Span_ODESolver;
9
10 % From X0_ODESolver
11 x1=X0_ODESolver(1);
12 x2=X0_ODESolver(2);
13 x3=X0_ODESolver(3);
14 x4=X0_ODESolver(4);
15 Lam_x1=X0_ODESolver(5);
16 Lam_x2=X0_ODESolver(6);
17 Lam_x3=X0_ODESolver(7);
18 Lam_x4=X0_ODESolver(8);
19
20 % Initializing P.Dot
21 Equation_Derivative=zeros(8,1);
22
23 %% Setting up the ODE Equations
24
25 % Constant
26 a=100;
27
28 % Computing U – Control
29 u=atan2(Lam_x4,Lam_x3);
30
31 % Equations
32 x1_Derivative=x3;
33 x2_Derivative=x4;
34 x3_Derivative=a*cos(u);
35 x4_Derivative=a*sin(u);
36 Lam_x1_Derivative=0;
37 Lam_x2_Derivative=0;
38 Lam_x3_Derivative=-Lam_x1;
39 Lam_x4_Derivative=-Lam_x2;
40
41 % Creating Equation Vector
42 Equation_Derivative(1)=x1_Derivative;
43 Equation_Derivative(2)=x2_Derivative;
44 Equation_Derivative(3)=x3_Derivative;

```

```
45 Equation_Derivative(4)=x4_Derivative;  
46 Equation_Derivative(5)=Lam_x1_Derivative;  
47 Equation_Derivative(6)=Lam_x2_Derivative;  
48 Equation_Derivative(7)=Lam_x3_Derivative;  
49 Equation_Derivative(8)=Lam_x4_Derivative;  
50  
51 end
```

```

1 function [ Error ] = LinearTangent_ODESolver( X0_RootFinder )
2
3 %% ODE Solver and Error Calculator : Problem 1 – Part 2
4
5 %% Initial Conditions
6
7 % Known Initial Conditions for the ODE Solver
8 x1_0=0;
9 x2_0=0;
10 x3_0=0;
11 x4_0=0;
12 Lam_x1_0=0;
13 t0_0=0;
14
15 X0_Known_ODESolver=[x1_0;x2_0;x3_0;x4_0;Lam_x1_0];
16
17 % Initial Conditions from the Root Finder
18 Lam_x2_0=X0_RootFinder(1);
19 Lam_x3_0=X0_RootFinder(2);
20 Lam_x4_0=X0_RootFinder(3);
21 tf_0=X0_RootFinder(4);
22
23 X0_RootFinder_Guess=X0_RootFinder(1:end-1);
24
25 % Complete Initial Conditions for the ODE Solver
26 X0_ODESolver = [X0_Known_ODESolver;X0_RootFinder_Guess];
27
28 %% ODE Solver
29
30 % Creating Time Span for Integration
31 tf=tf_0;
32
33 T_Span_ODESolver=[0,tf];
34
35 % Options for the ODE Solver
36 Options=optimset('Display','Iter','TolFun',1e-6,'TolX',1e-6);
37
38 % Calling the ODE Solver
39 [t_ODESolver,X_sol_ODESolver]=ode113(@LinearTangent_ODEEquations,...
40     T_Span_ODESolver,X0_ODESolver,Options);
41
42 % Getting the Outputs from the ODE Solver
43 x1=X_sol_ODESolver(:,1);
44 x2=X_sol_ODESolver(:,2);
45 x3=X_sol_ODESolver(:,3);

```



```

46 x4=X_sol_ODESolver(:,4);
47 Lam_x1=X_sol_ODESolver(:,5);
48 Lam_x2=X_sol_ODESolver(:,6);
49 Lam_x3=X_sol_ODESolver(:,7);
50 Lam_x4=X_sol_ODESolver(:,8);
51
52 %% Computing the Hamiltonian for getting H_tf
53
54 % Getting Boundary Values
55 x1_tf=x1(end);
56 x2_tf=x2(end);
57 x3_tf=x3(end);
58 x4_tf=x4(end);
59 Lam_x1_tf=Lam_x1(end);
60 Lam_x2_tf=Lam_x2(end);
61 Lam_x3_tf=Lam_x3(end);
62 Lam_x4_tf=Lam_x4(end);
63
64 % Computing U – Control at tf
65 u_tf=atan2(Lam_x4_tf, Lam_x3_tf);
66
67 a=10; % Constant
68
69 % Computing H(tf)
70 H_tf=(Lam_x1_tf*x3_tf) + (Lam_x2_tf*x4_tf) + (Lam_x3_tf*(a*cos(u_tf)))
    + ...
71     (Lam_x4_tf*(a*sin(u_tf)));
72
73 %% Computing the Error
74
75 % Initializing Error
76 Error=zeros(5,1);
77
78 % Creating Error Vector
79 Error(1)=x2_tf-5;
80 Error(2)=x3_tf-45;
81 Error(3)=x4_tf-0;
82 Error(4)=Lam_x1_tf-0;
83 Error(5)=H_tf+1;
84
85 end

```

1.2.3 Results

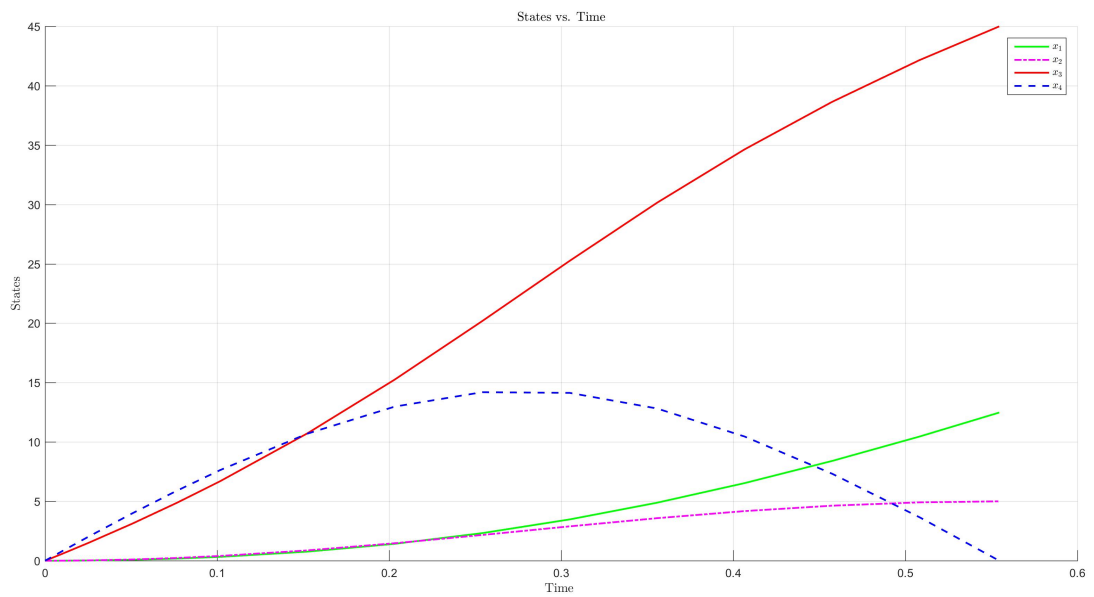


Figure 1: Linear Tangent Steering - HBVP - States

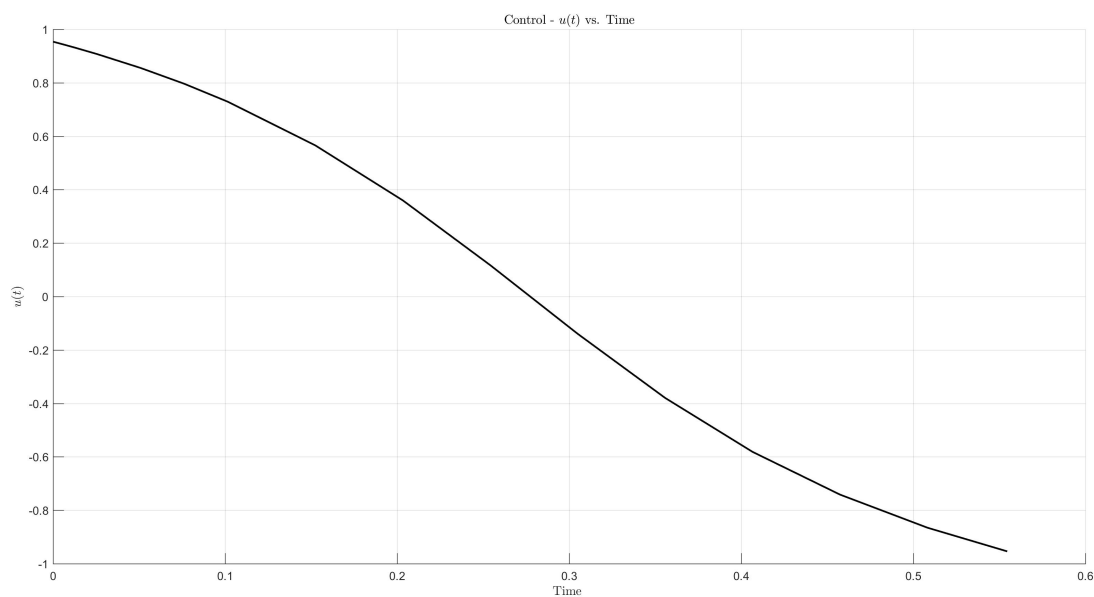


Figure 2: Linear Tangent Steering - HBVP - Control

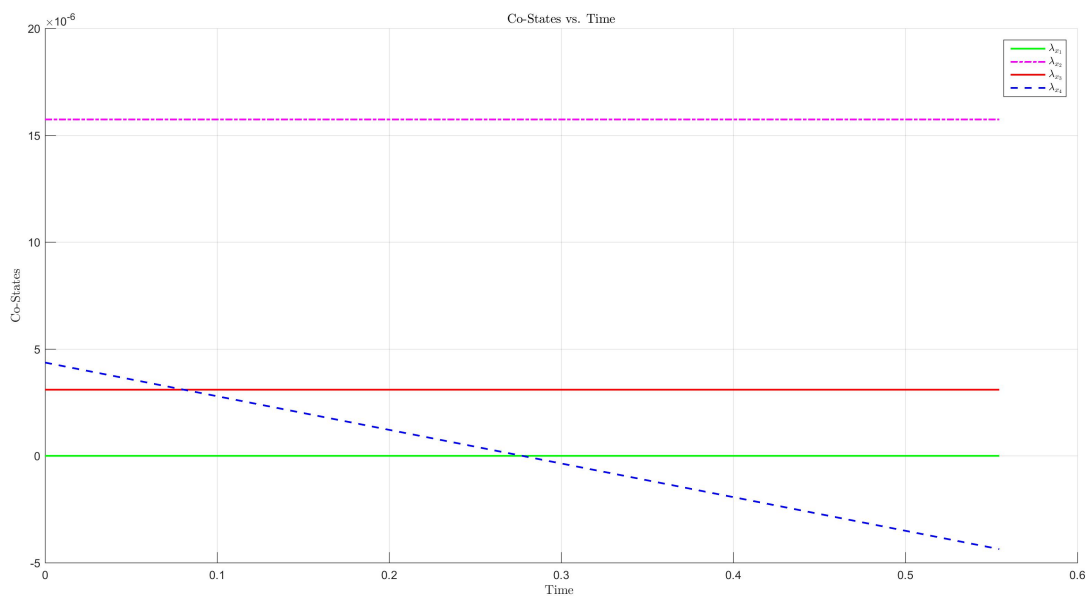


Figure 3: Linear Tangent Steering - HBVP - CoStates

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,

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

Subject to the dynamic and path constraints constraints,

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

With the following state constraints,

$$\underline{Z}_{min} \leq \underline{Z} \leq \underline{Z}_{max}, \quad (34)$$

For the Linear Tangent Steering Problem we have,

$$\underline{Z} = \begin{bmatrix} \underline{X}_1 \\ \underline{X}_2 \\ \underline{X}_3 \\ \underline{X}_4 \\ \underline{U} \\ t_0 \\ t_f \end{bmatrix} \quad (35)$$

Where,

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

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

$$g(\underline{Z}) = \begin{bmatrix} \Delta \underline{X}_1 \\ \Delta \underline{X}_2 \\ \Delta \underline{X}_3 \\ \Delta \underline{X}_4 \end{bmatrix} \quad (37)$$

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

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

Where D is a differentiation matrix.

The \underline{g}_{min} and \underline{g}_{max} are constant vectors of zeros since all the equations in (38) 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 (35) as follows;

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

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

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,

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

The Constraint Jacobian is given as,

$$\frac{\partial \underline{g}}{\partial \underline{Z}} = \begin{bmatrix} \frac{\partial g}{\partial \underline{X}_1} & \frac{\partial g}{\partial \underline{X}_2} & \frac{\partial g}{\partial \underline{X}_3} & \frac{\partial g}{\partial \underline{X}_4} & \frac{\partial g}{\partial \underline{U}} & \frac{\partial g}{\partial t_0} & \frac{\partial g}{\partial t_f} \end{bmatrix} \quad (42)$$

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

```

1  %-----%
2  %                               Linear Tangent Steering Problem          %
3
4  clear all;
5  close all;
6  clc;
7
8  %-----%
9  % Solve the following optimal control problem:                          %
10 % Minimize t_f                                                            %
11 % subject to the differential equation constraints                        %
12 % dx1/dt      = x3                                                        %
13 % dx2/dt      = x4                                                        %
14 % dx3/dt      = a*cos(u)                                                  %
15 % dx4/dt      = a*sin(u)                                                  %
16 %-----%
17 % BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                    %
18 %-----%
19 global psStuff nstates ncontrols a
20 global iGfun jGvar
21 %-----%
22 % END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                    %
23 %-----%
24
25
26 %-----%
27 %                               Define the constants for the problem      %
28 %-----%
29
30 a = 100;
31
32 %-----%
33 % Define the sizes of quantities in the optimal control problem          %
34 %-----%
35 nstates = 4;
36 ncontrols = 1;
37
38 %-----%
39 % Define bounds on the variables in the optimal control problem          %
40 %-----%
41 x1_0      = 0;
42
43 x2_0      = 0;

```



```

44 x2_f          = 5;
45
46 x3_0          = 0;
47 x3_f          = 45;
48
49 x4_0          = 0;
50 x4_f          = 0;
51
52 x1min         = 0;          x1max         = 100;
53 x2min         = 0;          x2max         = 100;
54 x3min         = 0;          x3max         = 100;
55 x4min         = 0;          x4max         = 100;
56
57 umin          = -pi/2;     umax          = pi/2;
58
59 t0min         = 0;          t0max         = 0;
60 tfmin         = 0;          tfmax         = 100;
61
62 %-----%
63 % In this section, we define the three type of discretizations %
64 % that can be employed. These three approaches are as follows: %
65 % (1) p-method = global pseudospectral method %
66 %              = single interval and the degree of the %
67 %              polynomial in the interval can be varied %
68 % (2) h-method = fixed-degree polynomial in each interval %
69 %              and the number of intervals can be varied %
70 % (3) hp-method = can vary BOTH the degree of the polynomial %
71 %              in each interval and the number of intervals %
72 % %
73 % For simplicity in this tutorial, we will allow for either a %
74 % p-method or an h-method. Regardless of which method is being %
75 % employed, the user needs to specify the following parameters: %
76 % (a) N = Polynomial Degree %
77 % (b) meshPoints = Set of Monotonically Increasing Mesh Points %
78 %              on the Interval  $\tau \in [-1, +1]$ . %
79 % %
80 % When using a p-method, the parameters N and meshPoints must be %
81 % specified as follows: %
82 % (i) meshPoints = [-1 1] %
83 % (ii) N = Choice of Polynomial Degree (e.g., N=10, N=20) %
84 % When using an h-method, the parameters N and meshPoints must be %
85 % specified as follows: %
86 % (i) meshPoints =  $[\tau_1, \tau_2, \tau_3, \dots, \tau_N]$  %
87 %              where  $\tau_1 = -1$ ,  $\tau_N = 1$  and %
88 %               $(\tau_2, \dots, \tau_{N-1})$  are %

```

```

89 %                                     monotonically increasing on the open %
90 %                                     interval  $(-1,+1)$ . %
91 %----- %
92 %      Compute Points, Weights, and Differentiation Matrix %
93 %----- %
94 %----- %
95 % Choose Polynomial Degree and Number of Mesh Intervals %
96 % numIntervals = 1  $\implies$  p-method %
97 % numIntervals > 1  $\implies$  h-method %
98 %----- %
99 N = 4;
100 numIntervals = 30;
101 %----- %
102 % DO NOT ALTER THE LINE OF CODE SHOWN BELOW! %
103 %----- %
104 meshPoints = linspace(-1,1,numIntervals+1).';
105 polyDegrees = N*ones(numIntervals,1);
106 [tau,w,D] = lgrPS(meshPoints,polyDegrees);
107 psStuff.tau = tau; psStuff.w = w; psStuff.D = D; NLGR = length(w);
108 %----- %
109 % DO NOT ALTER THE LINES OF CODE SHOWN ABOVE! %
110 %----- %
111
112 %----- %
113 % Set the bounds on the variables in the NLP. %
114 %----- %
115 zx1min = x1min*ones(length(tau),1);
116 zx1max = x1max*ones(length(tau),1);
117 zx1min(1) = x1_0; zx1max(1) = x1_0;
118
119 zx2min = x2min*ones(length(tau),1);
120 zx2max = x2max*ones(length(tau),1);
121 zx2min(1) = x2_0; zx2max(1) = x2_0;
122 zx2min(NLGR+1) = x2_f; zx2max(NLGR+1) = x2_f;
123
124 zx3min = x3min*ones(length(tau),1);
125 zx3max = x3max*ones(length(tau),1);
126 zx3min(1) = x3_0; zx3max(1) = x3_0;
127 zx3min(NLGR+1) = x3_f; zx3max(NLGR+1) = x3_f;
128
129 zx4min = x4min*ones(length(tau),1);
130 zx4max = x4max*ones(length(tau),1);
131 zx4min(1) = x4_0; zx4max(1) = x4_0;
132 zx4min(NLGR+1) = x4_f; zx4max(NLGR+1) = x4_f;
133

```

```

134 zumin = umin*ones(length(tau)-1,1);
135 zumax = umax*ones(length(tau)-1,1);
136
137 zmin = [zx1min; zx2min; zx3min; zx4min; zumin; t0min; tfmin];
138 zmax = [zx1max; zx2max; zx3max; zx4max; zumax; t0max; tfmax];
139
140 %-----%
141 % Set the bounds on the constraints in the NLP. %
142 %-----%
143 defectMin = zeros(nstates*(length(tau)-1),1);
144 defectMax = zeros(nstates*(length(tau)-1),1);
145 pathMin = []; pathMax = [];
146 eventMin = []; eventMax = [];
147 objMin = 0; objMax = inf;
148 Fmin = [objMin; defectMin; pathMin; eventMin];
149 Fmax = [objMax; defectMax; pathMax; eventMax];
150
151 %-----%
152 % Supply an initial guess for the NLP. %
153 %-----%
154 x1guess = x1_0*ones(NLGR+1,1)+randn(NLGR+1,1);
155 x2guess = x2_0*ones(NLGR+1,1)+randn(NLGR+1,1);
156 x3guess = x3_0*ones(NLGR+1,1)+randn(NLGR+1,1);
157 x4guess = x4_0*ones(NLGR+1,1)+randn(NLGR+1,1);
158 uguess = ((umin-umax)/2)*ones(NLGR,1)+randn(NLGR,1);
159 t0guess = 0;
160 tfguess = 1+randn(1,1);
161
162 z0 = [x1guess; x2guess; x3guess; x4guess; uguess; t0guess; tfguess];
163
164 %-----%
165 % Generate derivatives and sparsity pattern using Adigator %
166 %-----%
167 % - Constraint Funtction Derivatives
168 xsize = size(z0);
169 x = adigatorCreateDerivInput(xsize, 'z0');
170 output = adigatorGenJacFile('LinearTangentFun', {x});
171 S_jac = output.JacobianStructure;
172 [iGfun, jGvar] = find(S_jac);
173
174 % - Objective Funtcion Derivatives
175 xsize = size(z0);
176 x = adigatorCreateDerivInput(xsize, 'z0');
177 output = adigatorGenJacFile('LinearTangentObj', {x});
178 grd_structure = output.JacobianStructure;

```

```

179
180 %-----%
181 % Set IPOPT callback functions
182 %-----%
183 funcs.objective = @(Z) LinearTangentObj(Z);
184 funcs.gradient = @(Z) LinearTangentGrd(Z);
185 funcs.constraints = @(Z) LinearTangentCon(Z);
186 funcs.jacobian = @(Z) LinearTangentJac(Z);
187 funcs.jacobianstructure = @() LinearTangentJacPat(S_jac);
188 options.ipopt.hessian_approximation = 'limited-memory';
189
190 %-----%
191 % Set IPOPT Options %
192 %-----%
193 options.ipopt.tol = 1e-5;
194 options.ipopt.linear_solver = 'ma57';
195 options.ipopt.max_iter = 20000;
196 options.ipopt.mu_strategy = 'adaptive';
197 options.ipopt.ma57_automatic_scaling = 'yes';
198 options.ipopt.print_user_options = 'yes';
199 options.ipopt.output_file = ['LinearTangent', 'IPOPTinfo.txt']; % print
    output file
200 options.ipopt.print_level = 5; % set print level default
201
202 options.lb = zmin; % Lower bound on the variables.
203 options.ub = zmax; % Upper bound on the variables.
204 options.cl = Fmin; % Lower bounds on the constraint functions.
205 options.cu = Fmax; % Upper bounds on the constraint functions.
206
207 %-----%
208 % Call IPOPT
209 %-----%
210 tic; % Timing the Process
211
212 [z, info] = ipopt(z0, funcs, options);
213
214 TimeTaken = toc;
215
216 %-----%
217 % extract lagrange multipliers from ipopt output, info
218 %-----%
219 Fmul = info.lambda;
220
221 %-----%
222 % Extract the state and control from the decision vector z. %

```

```

223 % Remember that the state is approximated at the LGR points %
224 % plus the final point, while the control is only approximated %
225 % at only the LGR points. %
226 % %
227 x1 = z(1:NLGR+1);
228 x2 = z(NLGR+2:2*(NLGR+1));
229 x3 = z(2*(NLGR+1)+1:3*(NLGR+1));
230 x4 = z(3*(NLGR+1)+1:4*(NLGR+1));
231 u = z(4*(NLGR+1)+1:4*(NLGR+1)+NLGR);
232 t0 = z(end-1);
233 tf = z(end);
234 t = (tf-t0)*(tau+1)/2+t0; % Time for Plotting States
235 tLGR = t(1:end-1); % Time for Plotting Control
236
237 % %
238 % Extract the Lagrange multipliers corresponding %
239 % the defect constraints. %
240 % %
241 multipliersDefects = Fmul(2:nstates*NLGR+1);
242 multipliersDefects = reshape(multipliersDefects,NLGR,nstates);
243 % %
244 % Compute the costates at the LGR points via transformation %
245 % %
246 costateLGR = inv(diag(w))*multipliersDefects;
247 % %
248 % Compute the costate at the tau=+1 via transformation %
249 % %
250 costateF = D(:,end).'*multipliersDefects;
251 % %
252 % Now assemble the costates into a single matrix %
253 % %
254 costate = [costateLGR; costateF];
255 lam_x1 = costate(:,1); lam_x2 = costate(:,2);
256 lam_x3 = costate(:,3); lam_x4 = costate(:,4);
257
258 % %
259 % plot results %
260 % %
261 % Plotting States
262 figure(1)
263 hold on
264 grid on
265 plot(t,x1,'-g','LineWidth',1.5);
266 plot(t,x2,'-m','LineWidth',1.5);
267 plot(t,x3,'-r','LineWidth',1.5);

```

```

268 plot(t,x4,'—b','LineWidth',1.5);
269 title('States vs. Time','Interpreter','latex');
270 xlabel('Time (sec)','Interpreter','latex');
271 ylabel('States','Interpreter','latex');
272 legend1=legend('$x_{1}$','$x_{2}$','$x_{3}$','$x_{4}$');
273 set(legend1,'Interpreter','latex');
274 hold off;
275
276 % Plotting Control
277 figure(2)
278 hold on
279 grid on
280 plot(tLGR,u,'-k','LineWidth',1.5);
281 title('Control - $u(t)$ vs. Time','Interpreter','latex');
282 xlabel('Time (sec)','Interpreter','latex');
283 ylabel('$u(t)$','Interpreter','latex');
284 hold off;
285
286 % Plotting Costates
287 figure(3)
288 hold on
289 grid on
290 plot(t,lam_x1,'-g','LineWidth',1.5);
291 plot(t,lam_x2,'-m','LineWidth',1.5);
292 plot(t,lam_x3,'-r','LineWidth',1.5);
293 plot(t,lam_x4,'—b','LineWidth',1.5);
294 title('Co-States vs. Time','Interpreter','latex');
295 xlabel('Time (sec)','Interpreter','latex');
296 ylabel('Co-States','Interpreter','latex');
297 legend2=legend('$\lambda_{x_{1}}$','$\lambda_{x_{2}}$',...
298               '$\lambda_{x_{3}}$','$\lambda_{x_{4}}$');
299 set(legend2,'Interpreter','latex');
300 hold off;
301
302 fprintf('Time taken to solve the Collocation Problem = %.4f',TimeTaken)

```

```

1  function obj = LinearTangentObj(z)
2  % Computes the objective function of the problem
3
4  global psStuff nstates ncontrols a
5
6  %-----%
7  % Radau pseudospectral method quantities required: %
8  %   - Differentiation matrix (psStuff.D) %
9  %   - Legendre-Gauss-Radau weights (psStuff.w) %
10 %   - Legendre-Gauss-Radau points (psStuff.tau) %
11 %-----%
12 D = psStuff.D; tau = psStuff.tau; w = psStuff.w;
13
14 %-----%
15 % Decompose the NLP decision vector into pieces containing %
16 %   - the state %
17 %   - the control %
18 %   - the initial time %
19 %   - the final time %
20 %-----%
21 N = length(tau)-1;
22 stateIndices = 1:nstates*(N+1);
23 controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
24 t0Index = controlIndices(end)+1;
25 tfIndex = t0Index+1;
26 stateVector = z(stateIndices);
27 controlVector = z(controlIndices);
28 t0 = z(t0Index);
29 tf = z(tfIndex);
30
31 %-----%
32 % Reshape the state and control parts of the NLP decision vector %
33 % to matrices of sizes (N+1) by nstates and (N+1) by ncontrols, %
34 % respectively. The state is approximated at the N LGR points %
35 % plus the final point. Thus, each column of the state vector is %
36 % length N+1. The LEFT-HAND SIDE of the defect constraints, D*X, %
37 % uses the state at all of the points (N LGR points plus final %
38 % point). The RIGHT-HAND SIDE of the defect constraints, %
39 % (tf-t0)F/2, uses the state and control at only the LGR points. %
40 % Thus, it is necessary to extract the state approximations at %
41 % only the N LGR points. Finally, in the Radau pseudospectral %
42 % method, the control is approximated at only the N LGR points. %
43 %-----%
44 statePlusEnd = reshape(stateVector,N+1,nstates);
45 control = reshape(controlVector,N,ncontrols);

```

```
46 stateLGR = statePlusEnd(1:end-1,:);
47
48 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49 % Identify the components of the state column-wise from stateLGR. %
50 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
51 x1 = stateLGR(:,1);
52 x2 = stateLGR(:,2);
53 x3 = stateLGR(:,3);
54 x4 = stateLGR(:,4);
55 u = control;
56
57 % Cost function
58 J = tf;
59 obj = J;
60
61 end
```



```

1 function C = LinearTangentFun(z)
2
3 %-----%
4 % Objective and constraint functions for the orbit-raising %
5 % problem. This function is designed to be used with the NLP %
6 % solver SNOPT. %
7 %-----%
8 % DO NOT FOR ANY REASON ALTER THE LINE OF CODE BELOW! %
9 global psStuff nstates ncontrols a
10 % DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE! %
11 %-----%
12
13 %-----%
14 % Radau pseudospectral method quantities required: %
15 % - Differentiation matrix (psStuff.D) %
16 % - Legendre-Gauss-Radau weights (psStuff.w) %
17 % - Legendre-Gauss-Radau points (psStuff.tau) %
18 %-----%
19 D = psStuff.D; tau = psStuff.tau; w = psStuff.w;
20
21 %-----%
22 % Decompose the NLP decision vector into pieces containing %
23 % - the state %
24 % - the control %
25 % - the initial time %
26 % - the final time %
27 %-----%
28 N = length(tau)-1;
29 stateIndices = 1:nstates*(N+1);
30 controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
31 t0Index = controlIndices(end)+1;
32 tfIndex = t0Index+1;
33 stateVector = z(stateIndices);
34 controlVector = z(controlIndices);
35 t0 = z(t0Index);
36 tf = z(tfIndex);
37
38 %-----%
39 % Reshape the state and control parts of the NLP decision vector %
40 % to matrices of sizes (N+1) by nstates and (N+1) by ncontrols, %
41 % respectively. The state is approximated at the N LGR points %
42 % plus the final point. Thus, each column of the state vector is %
43 % length N+1. The LEFT-HAND SIDE of the defect constraints, D*X, %
44 % uses the state at all of the points (N LGR points plus final %
45 % point). The RIGHT-HAND SIDE of the defect constraints, %

```

```

46 % (tf-t0)F/2, uses the state and control at only the LGR points. %
47 % Thus, it is necessary to extract the state approximations at %
48 % only the N LGR points. Finally, in the Radau pseudospectral %
49 % method, the control is approximated at only the N LGR points. %
50 %-----%
51 statePlusEnd = reshape(stateVector,N+1,nstates);
52 control = reshape(controlVector,N,ncontrols);
53 stateLGR = statePlusEnd(1:end-1,:);
54
55 %-----%
56 % Identify the components of the state column-wise from stateLGR. %
57 %-----%
58 x1 = stateLGR(:,1);
59 x2 = stateLGR(:,2);
60 x3 = stateLGR(:,3);
61 x4 = stateLGR(:,4);
62 u = control;
63
64 %-----%
65 % Compute the right-hand side of the differential equations at %
66 % the N LGR points. Each component of the right-hand side is %
67 % stored as a column vector of length N, that is each column has %
68 % the form %
69 %          [ f_i(x_1,u_1,t_1) ] %
70 %          [ f_i(x_2,u_2,t_2) ] %
71 %          . %
72 %          . %
73 %          . %
74 %          [ f_i(x_N,u_N,t_N) ] %
75 % where "i" is the right-hand side of the ith component of the %
76 % vector field f. It is noted that in MATLABB the calculation of %
77 % the right-hand side is vectorized. %
78 %-----%
79 diffeqRHS = [x3, x4, a*cos(u), a*sin(u)];
80
81 %-----%
82 % Compute the left-hand side of the defect constraints, recalling %
83 % that the left-hand side is computed using the state at the LGR %
84 % points PLUS the final point. %
85 %-----%
86 diffeqLHS = D*statePlusEnd;
87
88 %-----%
89 % Construct the defect constraints at the N LGR points. %
90 % Remember that the right-hand side needs to be scaled by the %

```

```
91 % factor (tf-t0)/2 because the rate of change of the state is %
92 % being taken with respect to  $\tau \in [-1, +1]$ . Thus, we have %
93 %  $\frac{dt}{t} \frac{d\mathbf{u}}{d\mathbf{u}} = (tf-t0)/2$ . %
94 %-----%
95 defects = diffeqLHS-(tf-t0)*diffeqRHS/2;
96
97 %-----%
98 % Reshape the defect constraints into a column vector. %
99 %-----%
100 defects = reshape(defects,N*nstates,1);
101
102 %-----%
103 % Construct the objective function plus constraint vector. %
104 %-----%
105 J = tf;
106
107 C = [J; defects];
```

```
1 function constraints = LinearTangentCon(Z)
2 % computes the constraints
3
4 output      = LinearTangentFun(Z);
5 constraints = output;
6
7 end

1 function grd = LinearTangentGrd(Z)
2 % computes the gradient
3
4 output = LinearTangentObj_Jac(Z);
5 grd    = output;
6
7 end

1 function jac = LinearTangentJac(Z)
2 % computes the jacobian
3
4 [jac, ~] = LinearTangentFun_Jac(Z);
5
6 end

1 function jacpat = LinearTangentJacPat(S_jac)
2 % computes the jacobian structure
3
4 jacpat = S_jac;
5
6 end
```

1.3.3 Results

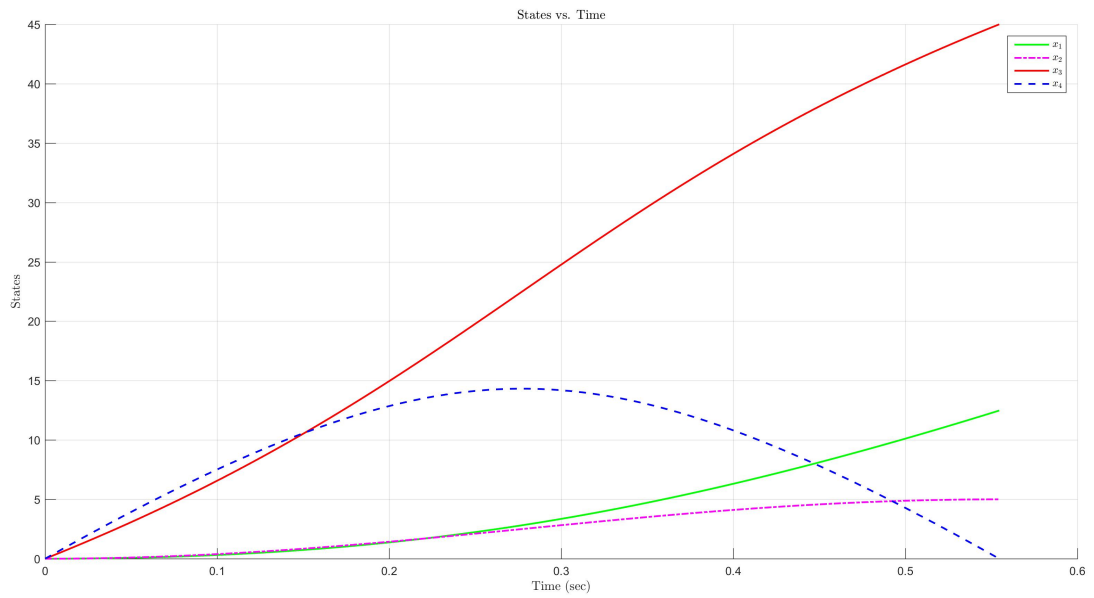


Figure 4: Linear Tangent Steering - Collocation - States

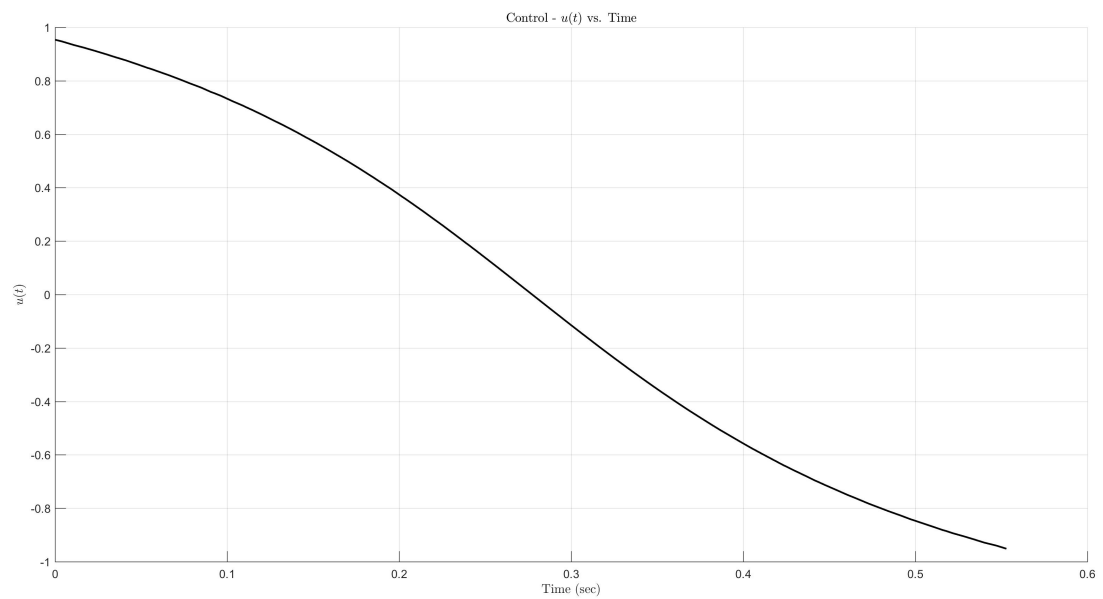


Figure 5: Linear Tangent Steering - Collocation - Control

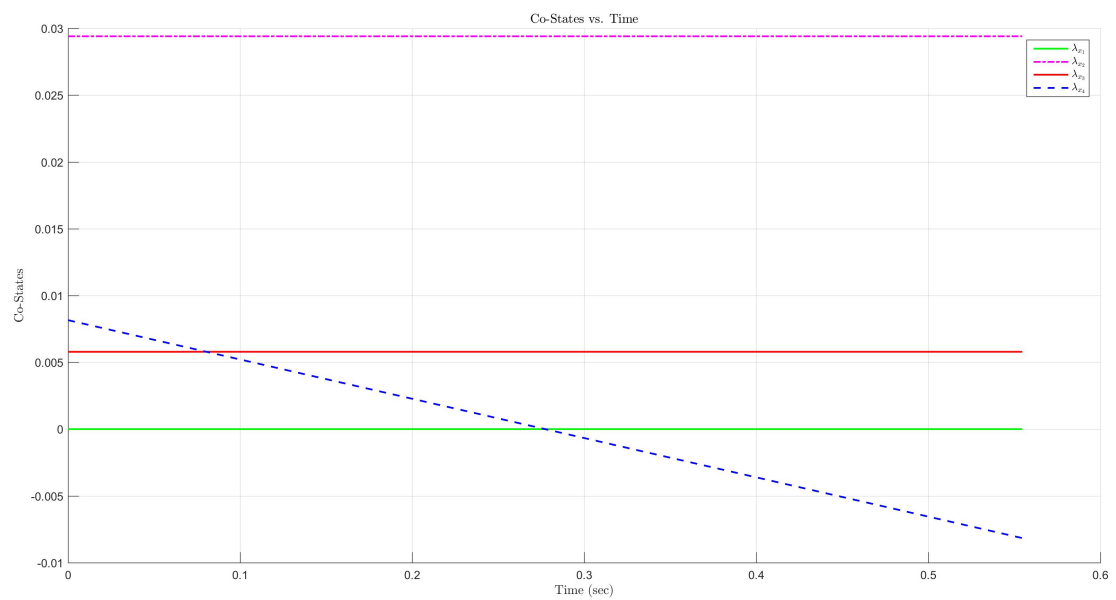


Figure 6: Linear Tangent Steering - Collocation - CoStates

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,

$$J = t_f, \quad (43)$$

Subject to the dynamic constraints,

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= u_1/L, \\ \dot{x}_3 &= x_4, \\ \dot{x}_4 &= u_2/I_\theta, \\ \dot{x}_5 &= x_6, \\ \dot{x}_6 &= u_3/I_\phi, \end{aligned} \quad (44)$$

With the control inequalities,,

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

With the following boundary conditions,

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

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

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

$$I_\phi = I_\theta \sin^2(x_5), \quad (48)$$

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,

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

Subject to the dynamic and path constraints constraints,

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

With the following state constraints,

$$\underline{Z}_{min} \leq \underline{Z} \leq \underline{Z}_{max}, \quad (51)$$

For the Linear Tangent Steering Problem we have,

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

Where,

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

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

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

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

$$g(\underline{Z}) = \begin{bmatrix} \underline{\Delta X_1} \\ \underline{\Delta X_2} \\ \underline{\Delta X_3} \\ \underline{\Delta X_4} \\ \underline{\Delta X_5} \\ \underline{\Delta X_6} \end{bmatrix} \quad (55)$$

The $\underline{\Delta X_1}$, $\underline{\Delta X_2}$, $\underline{\Delta X_3}$, $\underline{\Delta X_4}$, $\underline{\Delta X_5}$ and $\underline{\Delta X_6}$ are defined as follows,

$$\begin{aligned} \underline{\Delta X_1} &= D\underline{X}_{1:N+1} - \frac{t_f}{2} [x_{2:N+1}] &= \underline{0}, \\ \underline{\Delta X_2} &= D\underline{X}_{2:N+1} - \frac{t_f}{2} [u_{1:N}/L] &= \underline{0}, \\ \underline{\Delta X_3} &= D\underline{X}_{3:N+1} - \frac{t_f}{2} [x_{4:N+1}] &= \underline{0}, \\ \underline{\Delta X_4} &= D\underline{X}_{4:N+1} - \frac{t_f}{2} [u_{2:N}/I_\theta] &= \underline{0}, \\ \underline{\Delta X_5} &= D\underline{X}_{5:N+1} - \frac{t_f}{2} [x_{6:N+1}] &= \underline{0}, \\ \underline{\Delta X_6} &= D\underline{X}_{6:N+1} - \frac{t_f}{2} [u_{3:N}/I_\phi] &= \underline{0}, \end{aligned} \quad (56)$$

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

The \underline{g}_{min} and \underline{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;

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

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

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,

$$\frac{\partial F}{\partial \underline{Z}} = \left[\begin{array}{cccccccccccc} \frac{\partial F}{\partial \underline{X}_1} & \frac{\partial F}{\partial \underline{X}_2} & \frac{\partial F}{\partial \underline{X}_3} & \frac{\partial F}{\partial \underline{X}_4} & \frac{\partial F}{\partial \underline{X}_5} & \frac{\partial F}{\partial \underline{X}_6} & \frac{\partial F}{\partial \underline{U}_1} & \frac{\partial F}{\partial \underline{U}_2} & \frac{\partial F}{\partial \underline{U}_3} & \frac{\partial F}{\partial t_0} & \frac{\partial F}{\partial t_f} \end{array} \right] \quad (59)$$

The Constraint Jacobian is given as,

$$\frac{\partial \underline{g}}{\partial \underline{Z}} = \left[\begin{array}{cccccccccccc} \frac{\partial \underline{g}}{\partial \underline{X}_1} & \frac{\partial \underline{g}}{\partial \underline{X}_2} & \frac{\partial \underline{g}}{\partial \underline{X}_3} & \frac{\partial \underline{g}}{\partial \underline{X}_4} & \frac{\partial \underline{g}}{\partial \underline{X}_5} & \frac{\partial \underline{g}}{\partial \underline{X}_6} & \frac{\partial \underline{g}}{\partial \underline{U}_1} & \frac{\partial \underline{g}}{\partial \underline{U}_2} & \frac{\partial \underline{g}}{\partial \underline{U}_3} & \frac{\partial \underline{g}}{\partial t_0} & \frac{\partial \underline{g}}{\partial t_f} \end{array} \right] \quad (60)$$

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

```

1  %-----%
2  %                               Robot Arm Problem                               %
3
4  clear all;
5  close all;
6  clc;
7
8  %-----%
9  % Solve the following optimal control problem:                                %
10 % Minimize t_f                                                                %
11 % subject to the differential equation constraints                            %
12 % dx1/dt      = x2                                                            %
13 % dx2/dt      = u1/L                                                          %
14 % dx3/dt      = x4                                                            %
15 % dx4/dt      = u2/I_Theta                                                    %
16 % dx5/dt      = x6                                                            %
17 % dx6/dt      = au3/I_Phi                                                    %
18 %-----%
19 % BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                        %
20 %-----%
21 global psStuff nstates ncontrols L
22 global iGfun jGvar
23 %-----%
24 % END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!!                        %
25 %-----%
26
27
28 %-----%
29 %                               Define the constants for the problem          %
30 %-----%
31
32 L = 5;
33
34 %-----%
35 % Define the sizes of quantities in the optimal control problem             %
36 %-----%
37 nstates = 6;
38 ncontrols = 3;
39
40 %-----%
41 % Define bounds on the variables in the optimal control problem             %
42 %-----%
43 x1_0      = 4.5;

```

```

44 x1_f          = 4.5;
45
46 x2_0          = 0;
47 x2_f          = 0;
48
49 x3_0          = 0;
50 x3_f          = 2*pi/3;
51
52 x4_0          = 0;
53 x4_f          = 0;
54
55 x5_0          = pi/4;
56 x5_f          = pi/4;
57
58 x6_0          = 0;
59 x6_f          = 0;
60
61 x1min         = 0;          x1max         = L;
62 x2min         = -100;       x2max         = 100;
63 x3min         = -pi;        x3max         = pi;
64 x4min         = -100;       x4max         = 100;
65 x5min         = 0;          x5max         = pi;
66 x6min         = -100;       x6max         = 100;
67
68 u1min         = -1;         u1max         = 1;
69 u2min         = -1;         u2max         = 1;
70 u3min         = -1;         u3max         = 1;
71
72 t0min         = 0;          t0max         = 0;
73 tfmin         = 0;          tfmax         = 100;
74
75 %-----%
76 % In this section, we define the three type of discretizations %
77 % that can be employed. These three approaches are as follows: %
78 % (1) p-method = global pseudospectral method %
79 %              = single interval and the degree of the %
80 %              polynomial in the interval can be varied %
81 % (2) h-method = fixed-degree polynomial in each interval %
82 %              and the number of intervals can be varied %
83 % (3) hp-method = can vary BOTH the degree of the polynomial %
84 %              in each interval and the number of intervals %
85 % %
86 % For simplicity in this tutorial, we will allow for either a %
87 % p-method or an h-method. Regardless of which method is being %
88 % employed, the user needs to specify the following parameters: %

```

```

89 %      (a) N = Polynomial Degree %
90 %      (b) meshPoints = Set of Monotonically Increasing Mesh Points %
91 %                  on the Interval  $\tau \in [-1, +1]$ . %
92 % %
93 % When using a p-method, the parameters N and meshPoints must be %
94 % specified as follows: %
95 %      (i) meshPoints = [-1 1] %
96 %      (ii) N = Choice of Polynomial Degree (e.g., N=10, N=20) %
97 % When using an h-method, the parameters N and meshPoints must be %
98 % specified as follows: %
99 %      (i) meshPoints =  $[\tau_1, \tau_2, \tau_3, \dots, \tau_N]$  %
100 %                  where  $\tau_1 = -1$ ,  $\tau_N = 1$  and %
101 %                   $(\tau_2, \dots, \tau_{N-1})$  are %
102 %                  monotonically increasing on the open %
103 %                  interval  $(-1, +1)$ . %
104 % %
105 %      Compute Points, Weights, and Differentiation Matrix %
106 % %
107 % %
108 % Choose Polynomial Degree and Number of Mesh Intervals %
109 % numIntervals = 1  $\implies$  p-method %
110 % numIntervals > 1  $\implies$  h-method %
111 % %
112 N = 4;
113 numIntervals = 10;
114 % %
115 % DO NOT ALTER THE LINE OF CODE SHOWN BELOW! %
116 % %
117 meshPoints = linspace(-1,1,numIntervals+1).';
118 polyDegrees = N*ones(numIntervals,1);
119 [tau,w,D] = lgrPS(meshPoints,polyDegrees);
120 psStuff.tau = tau; psStuff.w = w; psStuff.D = D; NLGR = length(w);
121 % %
122 % DO NOT ALTER THE LINES OF CODE SHOWN ABOVE! %
123 % %
124 % %
125 % %
126 % Set the bounds on the variables in the NLP. %
127 % %
128 zx1min = x1min*ones(length(tau),1);
129 zx1max = x1max*ones(length(tau),1);
130 zx1min(1) = x1_0; zx1max(1) = x1_0;
131 zx1min(NLGR+1) = x1_f; zx1max(NLGR+1) = x1_f;
132
133 zx2min = x2min*ones(length(tau),1);

```



```

134 zx2max = x2max*ones(length(tau),1);
135 zx2min(1) = x2_0; zx2max(1) = x2_0;
136 zx2min(NLGR+1) = x2_f; zx2max(NLGR+1) = x2_f;
137
138 zx3min = x3min*ones(length(tau),1);
139 zx3max = x3max*ones(length(tau),1);
140 zx3min(1) = x3_0; zx3max(1) = x3_0;
141 zx3min(NLGR+1) = x3_f; zx3max(NLGR+1) = x3_f;
142
143 zx4min = x4min*ones(length(tau),1);
144 zx4max = x4max*ones(length(tau),1);
145 zx4min(1) = x4_0; zx4max(1) = x4_0;
146 zx4min(NLGR+1) = x4_f; zx4max(NLGR+1) = x4_f;
147
148 zx5min = x5min*ones(length(tau),1);
149 zx5max = x5max*ones(length(tau),1);
150 zx5min(1) = x5_0; zx5max(1) = x5_0;
151 zx5min(NLGR+1) = x5_f; zx5max(NLGR+1) = x5_f;
152
153 zx6min = x6min*ones(length(tau),1);
154 zx6max = x6max*ones(length(tau),1);
155 zx6min(1) = x6_0; zx6max(1) = x6_0;
156 zx6min(NLGR+1) = x6_f; zx6max(NLGR+1) = x6_f;
157
158 zu1min = u1min*ones(length(tau)-1,1);
159 zu1max = u1max*ones(length(tau)-1,1);
160
161 zu2min = u2min*ones(length(tau)-1,1);
162 zu2max = u2max*ones(length(tau)-1,1);
163
164 zu3min = u3min*ones(length(tau)-1,1);
165 zu3max = u3max*ones(length(tau)-1,1);
166
167 zmin = [zx1min; zx2min; zx3min; zx4min; zx5min; zx6min; zu1min; zu2min
        ;...
        zu3min; t0min; tfmin];
168
169 zmax = [zx1max; zx2max; zx3max; zx4max; zx5max; zx6max; zu1max; zu2max
        ;...
        zu3max; t0max; tfmax];
170
171
172 %-----%
173 % Set the bounds on the constraints in the NLP. %
174 %-----%
175 defectMin = zeros(nstates*(length(tau)-1),1);
176 defectMax = zeros(nstates*(length(tau)-1),1);

```

```

177 pathMin = []; pathMax = [];
178 eventMin = []; eventMax = [];
179 objMin = 0; objMax = inf;
180 Fmin = [objMin; defectMin; pathMin; eventMin];
181 Fmax = [objMax; defectMax; pathMax; eventMax];
182
183 %-----%
184 % Supply an initial guess for the NLP. %
185 %-----%
186 x1guess = x1_0*ones(NLGR+1,1)+randn(NLGR+1,1);
187 x2guess = x2_0*ones(NLGR+1,1)+randn(NLGR+1,1);
188 x3guess = x3_0*ones(NLGR+1,1)+randn(NLGR+1,1);
189 x4guess = x4_0*ones(NLGR+1,1)+randn(NLGR+1,1);
190 x5guess = x5_0*ones(NLGR+1,1)+randn(NLGR+1,1);
191 x6guess = x6_0*ones(NLGR+1,1);
192 u1guess = ((u1min+u1max)/2)*ones(NLGR,1)+randn(NLGR,1);
193 u2guess = ((u2min+u2max)/2)*ones(NLGR,1)+randn(NLGR,1);
194 u3guess = ((u3min+u3max)/2)*ones(NLGR,1)+randn(NLGR,1);
195 t0guess = 0;
196 tfguess = 1+randn(1,1);
197 z0 = [x1guess; x2guess; x3guess; x4guess; x5guess; x6guess; u1guess;...
198       u2guess; u3guess; t0guess; tfguess];
199
200 %-----%
201 % Generate derivatives and sparsity pattern using Adigator %
202 %-----%
203 % - Constraint Funtcion Derivatives
204 xsize = size(z0);
205 x      = adigatorCreateDerivInput(xsize, 'z0');
206 output = adigatorGenJacFile('RobotArmFun', {x});
207 S_jac  = output.JacobianStructure;
208 [iGfun, jGvar] = find(S_jac);
209
210 % - Objective Funtcion Derivatives
211 xsize = size(z0);
212 x      = adigatorCreateDerivInput(xsize, 'z0');
213 output = adigatorGenJacFile('RobotArmObj', {x});
214 grd_structure = output.JacobianStructure;
215
216 %-----%
217 % Set IPOPT callback functions %
218 %-----%
219 funcs.objective = @(Z)RobotArmObj(Z);
220 funcs.gradient  = @(Z)RobotArmGrd(Z);
221 funcs.constraints = @(Z)RobotArmCon(Z);

```

```

222 funcs.jacobian      = @(Z)RobotArmJac(Z);
223 funcs.jacobianstructure = @(S)RobotArmJacPat(S_jac);
224 options.ipopt.hessian_approximation = 'limited-memory';
225
226 %-----%
227 % Set IPOPT Options %
228 %-----%
229 options.ipopt.tol = 1e-5;
230 options.ipopt.linear_solver = 'mumps';
231 options.ipopt.max_iter = 20000;
232 options.ipopt.mu_strategy = 'adaptive';
233 options.ipopt.ma57_automatic_scaling = 'yes';
234 options.ipopt.print_user_options = 'yes';
235 options.ipopt.output_file = ['LinearTangent', 'IPOPTinfo.txt'];
236 options.ipopt.print_level = 5; % set print level default
237
238 options.lb = zmin; % Lower bound on the variables.
239 options.ub = zmax; % Upper bound on the variables.
240 options.cl = Fmin; % Lower bounds on the constraint functions.
241 options.cu = Fmax; % Upper bounds on the constraint functions.
242
243 %-----%
244 % Call IPOPT
245 %-----%
246 tic; % Timing the Process
247
248 [z, info] = ipopt(z0, funcs, options);
249
250 TimeTaken = toc;
251
252 %-----%
253 % extract lagrange multipliers from ipopt output, info
254 %-----%
255 Fmul = info.lambda;
256
257 %-----%
258 % Extract the state and control from the decision vector z. %
259 % Remember that the state is approximated at the LGR points %
260 % plus the final point, while the control is only approximated %
261 % at only the LGR points. %
262 %-----%
263 x1 = z(1:NLGR+1);
264 x2 = z(NLGR+2:2*(NLGR+1));
265 x3 = z(2*(NLGR+1)+1:3*(NLGR+1));
266 x4 = z(3*(NLGR+1)+1:4*(NLGR+1));

```

```

267 x5 = z(4*(NLGR+1)+1:5*(NLGR+1));
268 x6 = z(5*(NLGR+1)+1:6*(NLGR+1));
269 u1 = z(6*(NLGR+1)+1:6*(NLGR+1)+NLGR);
270 u2 = z(6*(NLGR+1)+NLGR+1:6*(NLGR+1)+2*NLGR);
271 u3 = z(6*(NLGR+1)+2*NLGR+1:6*(NLGR+1)+3*NLGR);
272 t0 = z(end-1);
273 tf = z(end);
274 t = (tf-t0)*(tau+1)/2+t0; % Time for Plotting States
275 tLGR = t(1:end-1); % Time for Plotting Control
276
277 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
278 % Extract the Lagrange multipliers corresponding
279 % the defect constraints.
280 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
281 multipliersDefects = Fmul(2:nstates*NLGR+1);
282 multipliersDefects = reshape(multipliersDefects,NLGR,nstates);
283 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
284 % Compute the costates at the LGR points via transformation
285 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
286 costateLGR = inv(diag(w))*multipliersDefects;
287 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
288 % Compute the costate at the tau=+1 via transformation
289 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
290 costateF = D(:,end).'*multipliersDefects;
291 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
292 % Now assemble the costates into a single matrix
293 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
294 costate = [costateLGR; costateF];
295 lam_x1 = costate(:,1); lam_x2 = costate(:,2);
296 lam_x3 = costate(:,3); lam_x4 = costate(:,4);
297 lam_x5 = costate(:,3); lam_x6 = costate(:,4);
298
299 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
300 % plot results
301 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
302 % Plotting States
303 figure(1)
304 hold on
305 grid on
306 plot(t,x1,'-g','LineWidth',1.5);
307 plot(t,x2,'-m','LineWidth',1.5);
308 plot(t,x3,'-r','LineWidth',1.5);
309 plot(t,x4,'-b','LineWidth',1.5);
310 plot(t,x5,'-c','LineWidth',1.5);
311 plot(t,x6,'-m','LineWidth',1.5);

```

```

312 title('States vs. Time','Interpreter','latex');
313 xlabel('Time (sec)','Interpreter','latex');
314 ylabel('States','Interpreter','latex');
315 legend1=legend('$x_{1}$','$x_{2}$','$x_{3}$','$x_{4}$','$x_{5}$',...
316               '$x_{6}$');
317 set(legend1,'Interpreter','latex');
318 hold off;
319
320 % Plotting Control
321 figure(2)
322 hold on
323 grid on
324 plot(tLGR,u1,'-r','LineWidth',1.5);
325 plot(tLGR,u2,'—b','LineWidth',1.5);
326 plot(tLGR,u3,'-g','LineWidth',1.5);
327 title('Control - $u(t)$ vs. Time','Interpreter','latex');
328 xlabel('Time (sec)','Interpreter','latex');
329 ylabel('$u(t)$','Interpreter','latex');
330 legend1=legend('$u_{1}$','$u_{2}$','$u_{3}$');
331 set(legend1,'Interpreter','latex');
332 hold off;
333
334 % Plotting Costates
335 figure(3)
336 hold on
337 grid on
338 plot(t,lam_x1,'-g','LineWidth',1.5);
339 plot(t,lam_x2,'-m','LineWidth',1.5);
340 plot(t,lam_x3,'-r','LineWidth',1.5);
341 plot(t,lam_x4,'—b','LineWidth',1.5);
342 plot(t,lam_x5,'-c','LineWidth',1.5);
343 plot(t,lam_x6,'—m','LineWidth',1.5);
344 title('Co-States vs. Time','Interpreter','latex');
345 xlabel('Time (sec)','Interpreter','latex');
346 ylabel('Co-States','Interpreter','latex');
347 legend2=legend('$\lambda_{x_{1}}$','$\lambda_{x_{2}}$'...
348               '$\lambda_{x_{3}}$','$\lambda_{x_{4}}$','$\lambda_{x_{5}}$'...
349               '$\lambda_{x_{6}}$');
350 set(legend2,'Interpreter','latex');
351 hold off;
352
353 fprintf('Time taken to solve the Collocation Problem = %.4f',TimeTaken)

```

```

1  function obj = RobotArmObj(z)
2  % Computes the objective function of the problem
3
4  global psStuff nstates ncontrols L
5
6  %-----%
7  % Radau pseudospectral method quantities required: %
8  %   - Differentiation matrix (psStuff.D) %
9  %   - Legendre-Gauss-Radau weights (psStuff.w) %
10 %   - Legendre-Gauss-Radau points (psStuff.tau) %
11 %-----%
12 D = psStuff.D; tau = psStuff.tau; w = psStuff.w;
13
14 %-----%
15 % Decompose the NLP decision vector into pieces containing %
16 %   - the state %
17 %   - the control %
18 %   - the initial time %
19 %   - the final time %
20 %-----%
21 N = length(tau)-1;
22 stateIndices = 1:nstates*(N+1);
23 controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
24 t0Index = controlIndices(end)+1;
25 tfIndex = t0Index+1;
26 stateVector = z(stateIndices);
27 controlVector = z(controlIndices);
28 t0 = z(t0Index);
29 tf = z(tfIndex);
30
31 %-----%
32 % Reshape the state and control parts of the NLP decision vector %
33 % to matrices of sizes (N+1) by nstates and (N+1) by ncontrols, %
34 % respectively. The state is approximated at the N LGR points %
35 % plus the final point. Thus, each column of the state vector is %
36 % length N+1. The LEFT-HAND SIDE of the defect constraints, D*X, %
37 % uses the state at all of the points (N LGR points plus final %
38 % point). The RIGHT-HAND SIDE of the defect constraints, %
39 % (tf-t0)F/2, uses the state and control at only the LGR points. %
40 % Thus, it is necessary to extract the state approximations at %
41 % only the N LGR points. Finally, in the Radau pseudospectral %
42 % method, the control is approximated at only the N LGR points. %
43 %-----%
44 statePlusEnd = reshape(stateVector,N+1,nstates);
45 control = reshape(controlVector,N,ncontrols);

```

```
46 stateLGR = statePlusEnd(1:end-1,:);
47
48 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49 % Identify the components of the state column-wise from stateLGR. %
50 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
51 x1 = stateLGR(:,1);
52 x2 = stateLGR(:,2);
53 x3 = stateLGR(:,3);
54 x4 = stateLGR(:,4);
55 x5 = stateLGR(:,5);
56 x6 = stateLGR(:,6);
57 u1 = control(:,1);
58 u2 = control(:,2);
59 u3 = control(:,3);
60
61 % Cost function
62 J = tf;
63 obj = J;
64
65 end
```

```

1  function C = RobotArmFun(z)
2
3  %-----%
4  % Objective and constraint functions for the orbit-raising %
5  % problem. This function is designed to be used with the NLP %
6  % solver SNOPT. %
7  %-----%
8  % DO NOT FOR ANY REASON ALTER THE LINE OF CODE BELOW! %
9  global psStuff nstates ncontrols L
10 % DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE! %
11 %-----%
12
13 %-----%
14 % Radau pseudospectral method quantities required: %
15 % - Differentiation matrix (psStuff.D) %
16 % - Legendre-Gauss-Radau weights (psStuff.w) %
17 % - Legendre-Gauss-Radau points (psStuff.tau) %
18 %-----%
19 D = psStuff.D; tau = psStuff.tau; w = psStuff.w;
20
21 %-----%
22 % Decompose the NLP decision vector into pieces containing %
23 % - the state %
24 % - the control %
25 % - the initial time %
26 % - the final time %
27 %-----%
28 N = length(tau)-1;
29 stateIndices = 1:nstates*(N+1);
30 controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
31 t0Index = controlIndices(end)+1;
32 tfIndex = t0Index+1;
33 stateVector = z(stateIndices);
34 controlVector = z(controlIndices);
35 t0 = z(t0Index);
36 tf = z(tfIndex);
37
38 %-----%
39 % Reshape the state and control parts of the NLP decision vector %
40 % to matrices of sizes (N+1) by nstates and (N+1) by ncontrols, %
41 % respectively. The state is approximated at the N LGR points %
42 % plus the final point. Thus, each column of the state vector is %
43 % length N+1. The LEFT-HAND SIDE of the defect constraints, D*X, %
44 % uses the state at all of the points (N LGR points plus final %
45 % point). The RIGHT-HAND SIDE of the defect constraints, %

```



```

46 % (tf-t0)F/2, uses the state and control at only the LGR points. %
47 % Thus, it is necessary to extract the state approximations at %
48 % only the N LGR points. Finally, in the Radau pseudospectral %
49 % method, the control is approximated at only the N LGR points. %
50 %-----%
51 statePlusEnd = reshape(stateVector,N+1,nstates);
52 control = reshape(controlVector,N,ncontrols);
53 stateLGR = statePlusEnd(1:end-1,:);
54
55 %-----%
56 % Identify the components of the state column-wise from stateLGR. %
57 %-----%
58 x1 = stateLGR(:,1);
59 x2 = stateLGR(:,2);
60 x3 = stateLGR(:,3);
61 x4 = stateLGR(:,4);
62 x5 = stateLGR(:,5);
63 x6 = stateLGR(:,6);
64 u1 = control(:,1);
65 u2 = control(:,2);
66 u3 = control(:,3);
67
68 %-----%
69 % Compute the right-hand side of the differential equations at %
70 % the N LGR points. Each component of the right-hand side is %
71 % stored as a column vector of length N, that is each column has %
72 % the form %
73 %           [ f_i(x_1,u_1,t_1) ] %
74 %           [ f_i(x_2,u_2,t_2) ] %
75 %           . %
76 %           . %
77 %           . %
78 %           [ f_i(x_N,u_N,t_N) ] %
79 % where "i" is the right-hand side of the ith component of the %
80 % vector field f. It is noted that in MATLABB the calculation of %
81 % the right-hand side is vectorized. %
82 %-----%
83
84 % Basic Computation
85 I_Phi=(1/3)*((L-x1).^3+x1.^3);
86 I_Theta=I_Phi.*sin(x5).^2;
87
88 diffeqRHS = [x2, u1/L, x4, u2./I_Theta, x6, u3./I_Phi];
89
90 %-----%

```

```

91 % Compute the left-hand side of the defect constraints, recalling %
92 % that the left-hand side is computed using the state at the LGR %
93 % points PLUS the final point. %
94 % %
95 diffeqLHS = D*statePlusEnd;
96
97 % %
98 % Construct the defect constraints at the N LGR points. %
99 % Remember that the right-hand side needs to be scaled by the %
100 % factor (tf-t0)/2 because the rate of change of the state is %
101 % being taken with respect to  $\tau \in [-1, +1]$ . Thus, we have %
102 %  $dt/d\tau = (tf-t0)/2$ . %
103 % %
104 defects = diffeqLHS - (tf-t0)*diffeqRHS/2;
105
106 % %
107 % Reshape the defect constraints into a column vector. %
108 % %
109 defects = reshape(defects, N*nstates, 1);
110
111 % %
112 % Construct the objective function plus constraint vector. %
113 % %
114 J = tf;
115
116 C = [J; defects];

```

```
1 function constraints = RobotArmCon(Z)
2 % computes the constraints
3
4 output      = RobotArmFun(Z);
5 constraints = output;
6
7 end

1 function grd = RobotArmGrd(Z)
2 % computes the gradient
3
4 output = RobotArmObj_Jac(Z);
5 grd    = output;
6
7 end

1 function jac = RobotArmJac(Z)
2 % computes the jacobian
3
4 [jac, ~] = RobotArmFun_Jac(Z);
5
6 end

1 function jacpat = RobotArmJacPat(S_jac)
2 % computes the jacobian structure
3
4 jacpat = S_jac;
5
6 end
```

2.4 Results

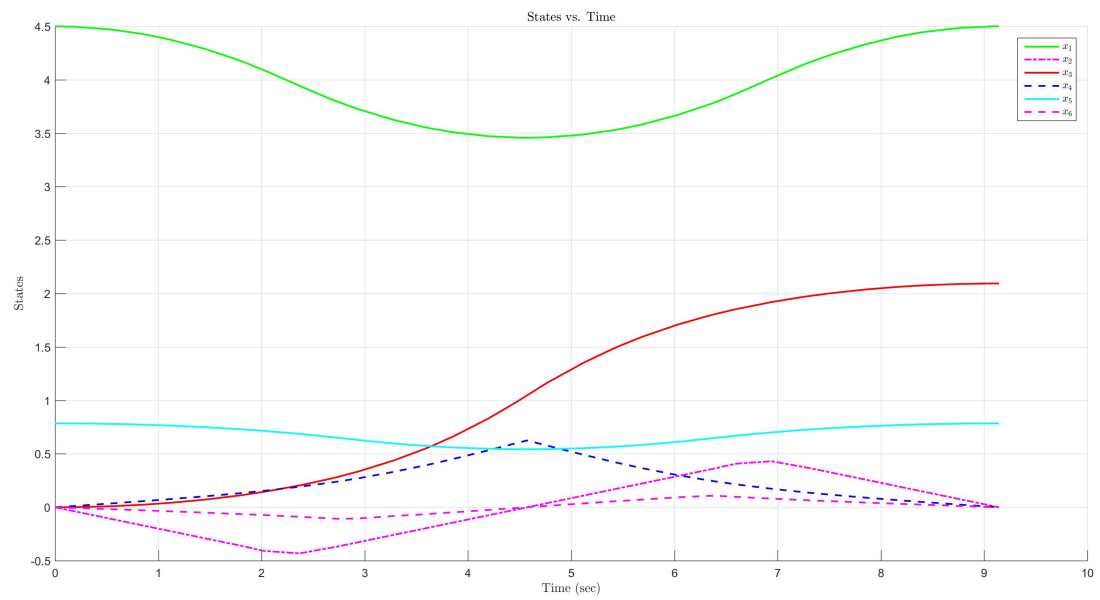


Figure 7: Robot Arm - Collocation - States

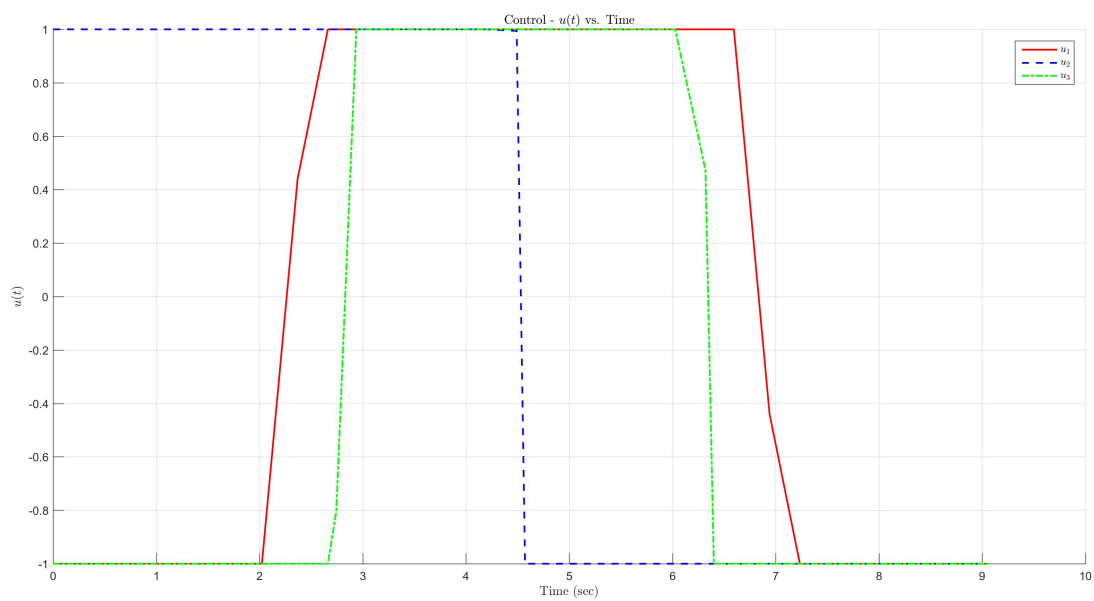


Figure 8: Robot Arm - Collocation - Controls

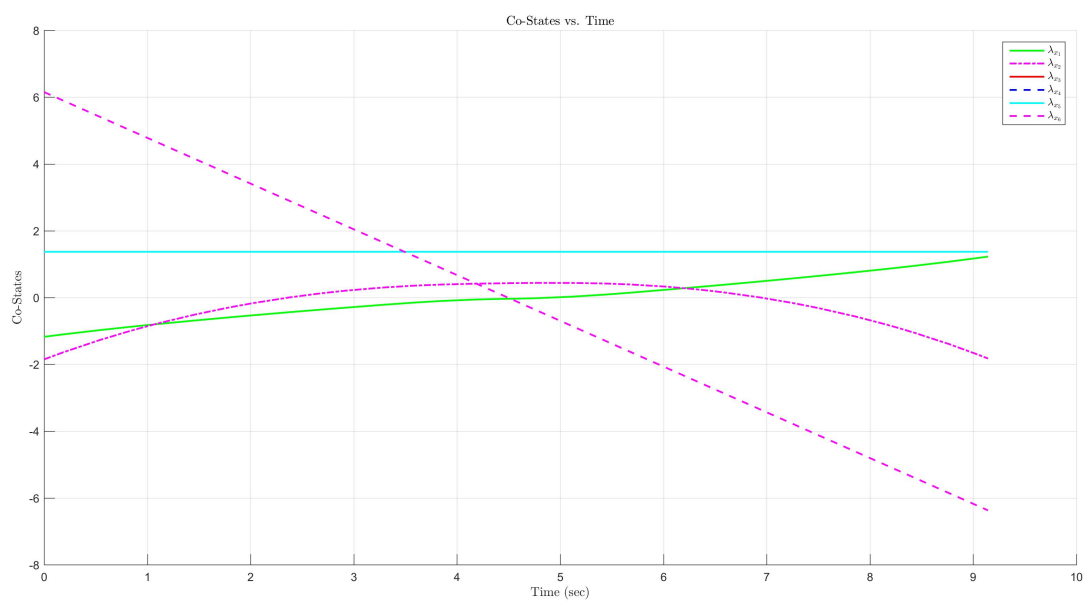


Figure 9: Robot Arm - Collocation - CoStates

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.