PROJECT
SPACECRAFT ORBIT TRANSFER

OBJECTIVE

The of this project is to develop a solution to the optimal control problem arising in the Spacecraft Orbit Transfer using Legendre-Gauss-Radau collocation.

SPACECRAFT DYNAMICS

Considering a spacecraft, modeled as a point mass *P* of mass *m*, moving relative to an inertial reference frame I. Furthermore, assume that the particle is moving in a plane that is itself fixed in I. Next, let $\{e_x, e_y, e_z\}$ be a right-handed orthonormal basis fixed in I.

Assuming now that the position of the spacecraft is denoted $r_{P/O}$, where *O* is fixed in I and denotes the location of the Sun (that is, the position of the spacecraft is measured relative to the Sun). Suppose now that $r_{P/O}$ is parameterized in terms of a basis $\{u_r, u_\theta, u_z\}$ where $\{u_r, u_\theta, u_z\}$ rotates about the e_z direction such that θ is the angle from e_x to u_r. Finally, assume that the distance from *O* to *P* is denoted r. A schematic of the geometry of the problem is given in Fig. 1.

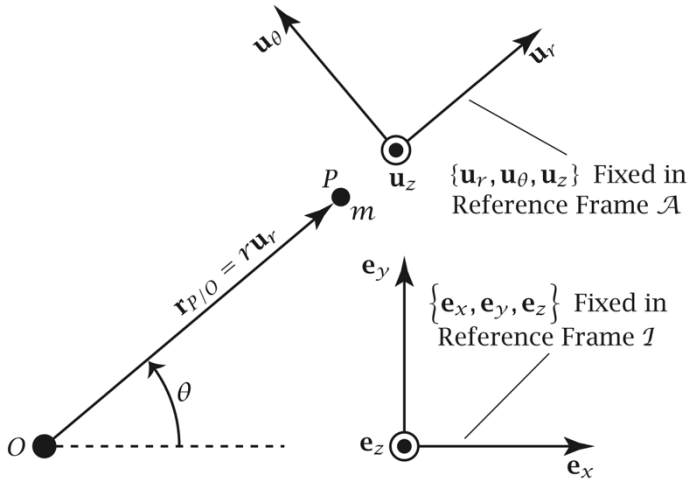


Figure 1: Schematic of particle moving in an inertially fixed plane.

Assuming now that the following two forces act on the spacecraft: (1) gravitational force and (2) thrust force. The gravitational force is given as

$$G = -m\mu \frac{r_{P/O}}{\left\| r_{P/O} \right\|^3}$$

while the thrust force is given as

$$T = Tw$$

where w is a unit vector that lies an angle β from the direction u_θ (in other words, β is the angle between u_θ and w).

OPTIMAL CONTROL PROBLEM

First, the spacecraft starts in a heliocentric circular orbit of radius $r_0 = 1$ and terminates in a heliocentric circular orbit of radius $r_f = 1.5$. It is assumed that the initial longitude (where θ denotes the longitude) is zero while the terminal longitude is free. Furthermore, the initial mass is unity (that is, $m_0 = 1$) while the terminal mass is free. This problem is solved for two objectives, 1) To minimize the time taken to transfer the spacecraft from the initial orbit to the terminal orbit and 2) To maximize the mass of the spacecraft at final time. In order to achieve these objectives in placing the space vehicle in a higher energy orbit, the optimal control formulation leads to the following set of necessary conditions of optimality.

$$\dot{X} = f(t, X, u) \quad \text{(i)}$$

$$\dot{\lambda} = \frac{dH}{dX} = g(t, X, u, \lambda) \quad \text{(ii)}$$

$$\frac{dH}{du} = 0 = \varphi(t, X, u, \lambda) \quad \text{(iii)}$$

Where X and λ represents the 4x1 state and the co-state vectors respectively and u, is the thrust directional angle. Equation (i) and (ii) are differential equations, and equation (iii) is just an algebraic equation that leads to the necessary conditions for optimal control.

The set of 5 first order differential equations that represent equation (i) which were derived using Newton's second law and Lagrange method are,

$$\dot{r} = v_r \quad (1)$$

$$\dot{\theta} = \frac{v_\theta}{r} \quad (2)$$

$$\dot{v}_r - \frac{{v_\theta}^2}{r} + \frac{\mu}{r^2} - \frac{T \sin\beta}{m} = 0 \quad (3)$$

$$\dot{v}_\theta + \frac{v_r v_\theta}{r} - \frac{T \cos\beta}{m} = 0 \quad (4)$$

$$\dot{m} = -\frac{T}{v_e} \quad (5)$$

The known boundary conditions at the initial and final time are,

$$r_0 = 1; \quad (6)$$

$$\theta_0 = 0; \quad (7)$$

$$v_r(0) = 0; \quad (8)$$

$$v_\theta(0) = \sqrt{\frac{\mu}{r(0)}}; \quad (9)$$

$$m_0 = 1; \quad (10)$$

$$r_f = 1.5; \quad (11)$$

$$\theta_f = free; \quad (12)$$

$$v_r(f) = 0; \quad (13)$$

$$v_\theta(f) = \sqrt{\frac{\mu}{r(f)}}; \quad (14)$$

$$m_f = free; \quad (15)$$

The objective in this optimal control problem is represented using an objective function J as follows.

(1) For the objective to maximize mass, performance Index J is,

$$J = \int_{m_0}^{m_f} -1 \, dm$$

(1) For the objective to minimize time, performance Index J is,

$$J = \int_{t_0}^{t_f} dt$$

In the previous solution using indirect and direct shooting the only control was the thrust angle β. Now the problem is slightly changed so that both the thrust magnitude, T, and the thrust angle β are controls. Using this formulation, the differential equations remain same, the only difference being that now the formulation contains two controls. While the thrust angle is free, the thrust magnitude is constrained as follows:

$$0 \leq T \leq T_{max}$$

Where T_{max} is the max value of the thrust available and $T_{max} = 0.1405$

NUMERICAL SOLUTION TO THE OPTIMAL CONTROL PROBLEM

Tabular Summary of the Results
For the objective to maximize mass at final time
(1) Using U1 U2 ,and T as input.

Degree of the Polynomial in each interval. (N)	Number of Intervals. (K)	No of iterations Performed	Objective function to maximize mass (J)	Final time. (tf)	Total CPU Seconds in IPOPT (without NLP function evaluation)	Total CPU seconds in NLP function evaluation	Result
3	2	41	0.9084	7.1676	0.541	0.136	Optimal Solution Found
3	4	640	0.9079	8.8213	4.755	2.154	Optimal Solution Found

3	8	96	0.9077	5.8175	0.834	0.331	Optimal Solution Found
3	16	93	0.9077	6.4541	0.509	0.218	Optimal Solution Found
3	32	112	0.9077	6.9504	0.917	0.308	Optimal Solution Found
4	2	570	0.9094	11.1036	3.318	0.891	Optimal solution found
4	4	334	0.9077	5.3019	1.962	0.538	Optimal solution found
4	8	101	0.9077	7.3107	0.679	0.179	Optimal solution found
4	16	136	0.9077	6.6956	1.654	0.639	Optimal solution found
4	32	214	0.9077	6.9774	1.877	0.462	Optimal solution found

Discussion –

The results show that the estimate of the optimal solution gets better as the number of intervals increases. As the number of LGR points increases the number of state estimates increases hence improving the accuracy of the estimate. It can be inferred from the table above that the estimates using 2 intervals give the least accurate estimates of the optimal solution and the intervals 16 and 32 give the best possible estimate of the control. The input plots of T shows that the torque is zero most of the time indicating that the optimal solution is conserving fuel. And T switches to its maximum value twice. The final trajectory is reached at a delayed time at around 7 seconds. The plot also shows that the spacecraft takes an extended trajectory. And U1 and U2 are seen to be display a polar behavior when U1 is 0 while U2 is 1 most of the time indicating their sine and cosine properties when throttle angle is 0. The polynomial of degree 4 is seen to give a better estimate compared to polynomial degree 3

Using Beta and T as control

Degree of the	**Number of**	**No of iterations Performed**	**Objective function**	**Final time. (tf)**	**Total CPU Seconds in**	**Total CPU seconds in**	**Result**

Polynomial in each interval. (N)	**Intervals. (K)**		**to maximize mass (J)**		**IPOPT (without NLP function evaluation)**	**NLP function evaluation**	
3	2	491	0.9100	10.7531	3.262	0.720	Optimal solution found
3	4	80	0.9079	8.8213	0.322	0.149	Optimal solution found
3	8	998	0.9077	7.4139	6.532	1.839	Optimal solution found
3	16	210	0.9077	7.7128	1.334	0.434	Optimal solution found
3	32	228	0.9077	7.8986	1.574	0.627	Optimal solution found
4	2	210	0.9061	16.5409	1.624	0.404	Optimal solution found
4	4	147	0.9076	7.3519	1.070	0.220	Optimal solution found
4	8	708	0.9077	6.7487	5.335	1.608	Optimal solution found
4	16	134	0.9077	7.8309	1.005	0.301	Optimal solution found
4	32	216	0.9077	7.7029	1.417	0.505	Optimal solution found

Discussion –

A better estimate was obtained with U1 and U2 as control compared to using a single control angle beta. The plots for beta showed more fluctuation while using just beta as control. But this fluctuation was indicated when torque T was zero which implies that fluctuation in beta had no effect on the optimality. The number of iterations for each setting also increased in this approach as indicated in the table above.

RESULTS –

(2) Objective – To minimize final time tf

Degree of the Polynomial in each interval. (N)	Number of Intervals. (K)	No of iterations Performed	Objective function to minimize time (J)	Final mass. mass(tf)	Total CPU Seconds in IPOPT (without NLP function evaluation)	Total CPU seconds in NLP function evaluation	Result
3	2	42	3.4519	0.7558	0.922	0.107	Optimal solution found
3	4	43	3.2469	0.7568	0.410	0.127	Optimal solution found
3	8	93	3.2919	0.7692	0.567	0.177	Optimal solution found
3	16	99	3.2481	0.7567	0.787	0.198	Optimal solution found
3	32	89	3.2481	0.7567	1.466	0.357	Optimal solution found
4	2	109	3.4276	0.7568	0.662	0.179	Optimal solution found
4	4	79	3.2478	0.7567	0.802	0.421	Optimal solution found
4	8	92	3.2542	0.7582	0.769	0.263	Optimal solution found
4	16	135	3.2585	0.7593	0.860	0.271	Optimal solution found
4	32	69	3.2585	0.7562	0.833	0.315	Optimal solution found

Discussion –

The Tabular column above indicates that IPOPT was able to find the optimal solution. The objective function had a very close estimate to the optimal time found using the indirect methods. The estimates to final time were seen to be more accurate as the number of intervals were increased. The input plot shows a switch in the control U1 and U2 indicating a switch in beta. The torque T always remained constant. This shows that at the optimal solution to achieve this objective the throttle must be always at its maximum. The switch in beta shows that the direction of the throttle angle show switch to the opposite direction. This makes the spacecraft to slow down as it approaches the final trajectories and to satisfy the specified boundary conditions.

For Beta and T as control inputs

Degree of the Polynomial in each interval. (N)	**Number of Intervals. (K)**	**No of iterations Performed**	**Objective function to minimize time (J)**	**Final mass. mass(tf)**	**Total CPU Seconds in IPOPT (without NLP function evaluation)**	**Total CPU seconds in NLP function evaluation**	**Result**
3	2	40	3.2456	0.7569	0.543	0.184	Optimal control found
3	4	56	3.2471	0.7568	0.447	0.130	Optimal solution found
3	8	84	3.2637	0.7556	0.539	0.165	Optimal solution found
3	16	98	3.3011	0.7644	0.749	0.184	Optimal solution found
3	32	142	3.2709	0.7589	0.949	0.286	Optimal solution found
4	2	58	3.3174	0.7824	0.442	0.132	Optimal solution found
4	4	80	3.2910	0.7738	0.659	0.199	Optimal solution found

4	8	105	3.2996	0.7717	0.763	0.277	Optimal solution found
4	16	167	3.2633	0.7556	1.331	0.296	Optimal solution found
4	32	254	3.2769	0.7601	1.873	0.590	Optimal solution found

Discussion –

A closer estimate was obtained with U1 and U2 as control compared to using a single control angle beta. Final times were estimated to be closer with polynomial of degree 3 compared to polynomial of degree 4.

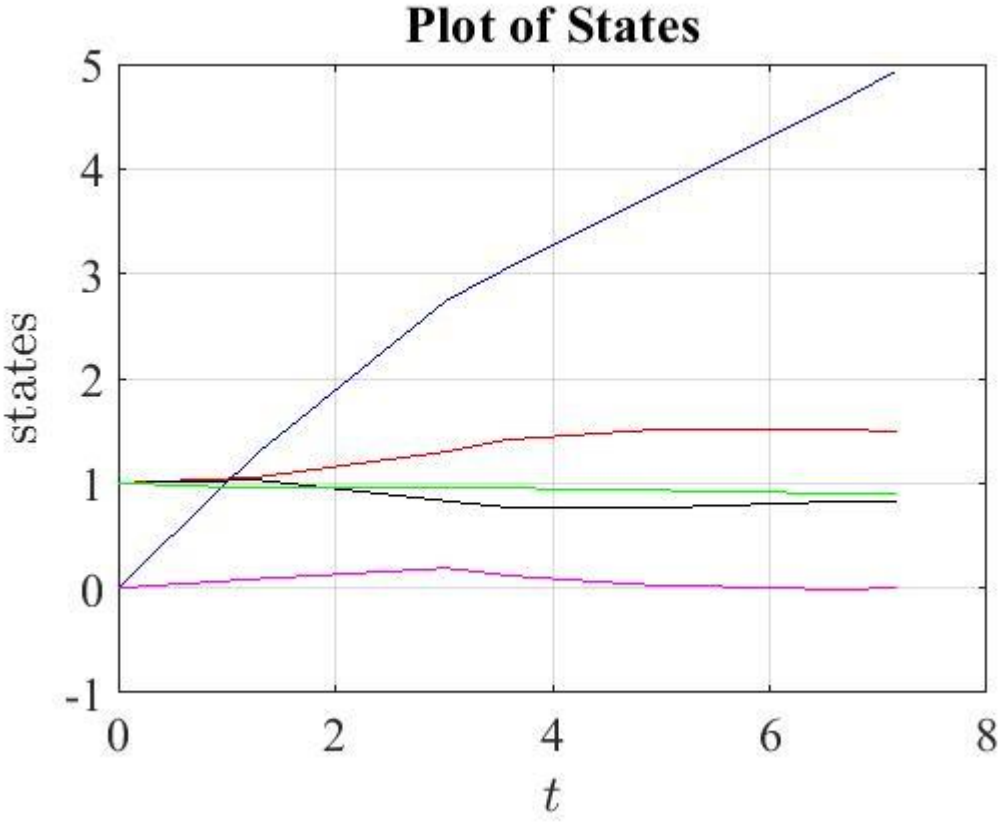
Conclusion –

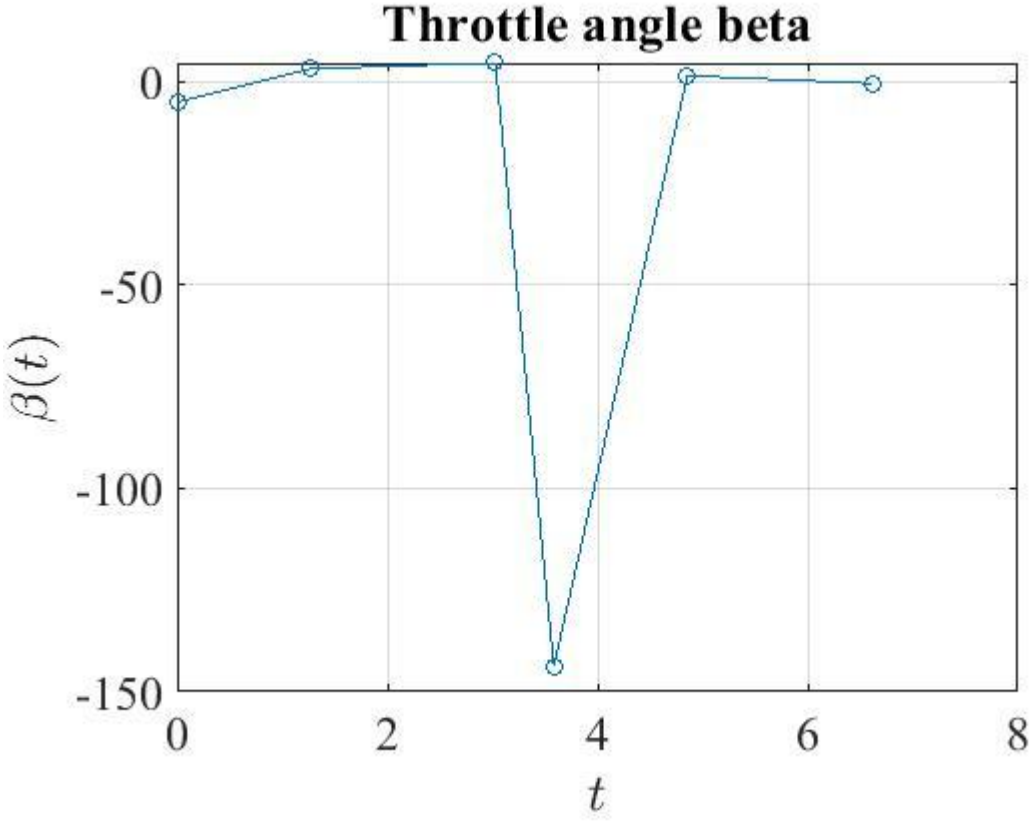
The computational time was reduced significantly while using Lagrange-Radau-Collocation at different intervals and degree of the polynomial when compared to the indirect and direct shooting methods in the midterm. It was observed that the thrust angle was not well defined when beta was used as control. And the estimates obtained using U1 and U2 as control were more accurate compared to beta as control. Faster convergence to optimality was observed using LGR collocation compared to the results obtained from direct and indirect shooting methods form the midterm exam.

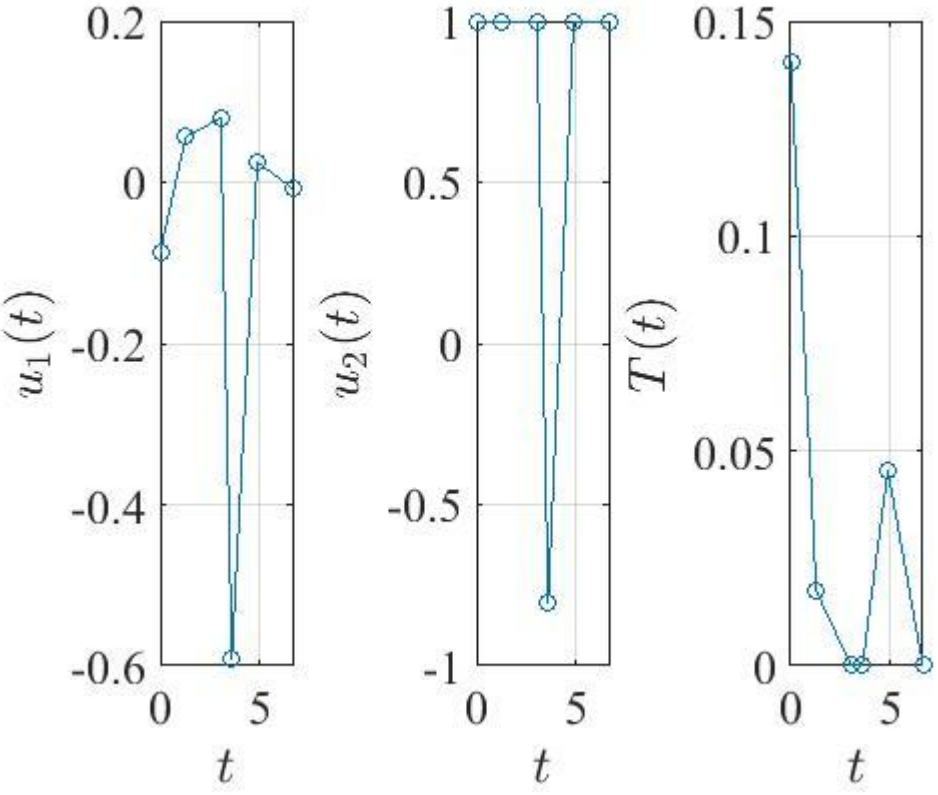
Graphs –

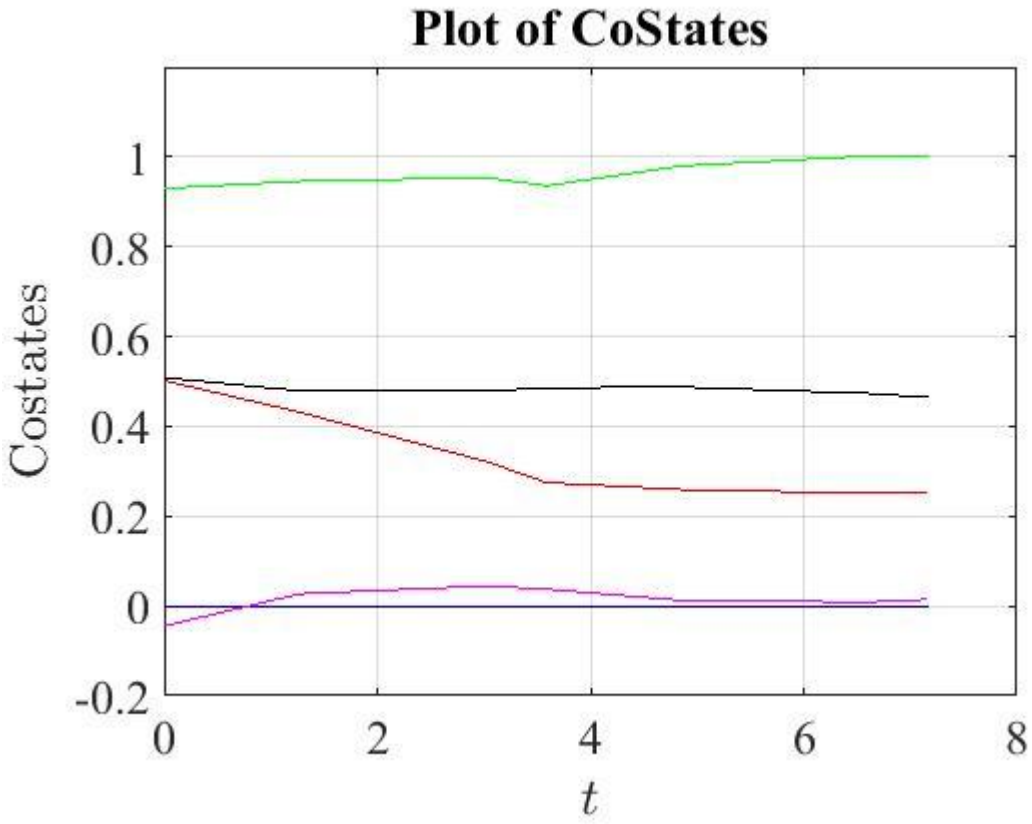
Objective – To maximize mass at final time (M_tf) using U1, U2 and T as control

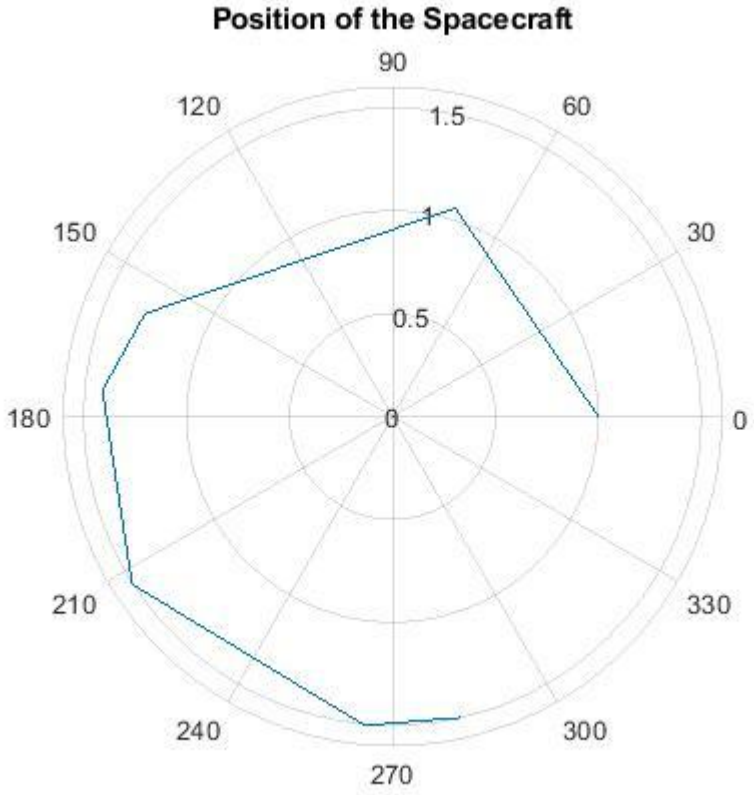
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 2



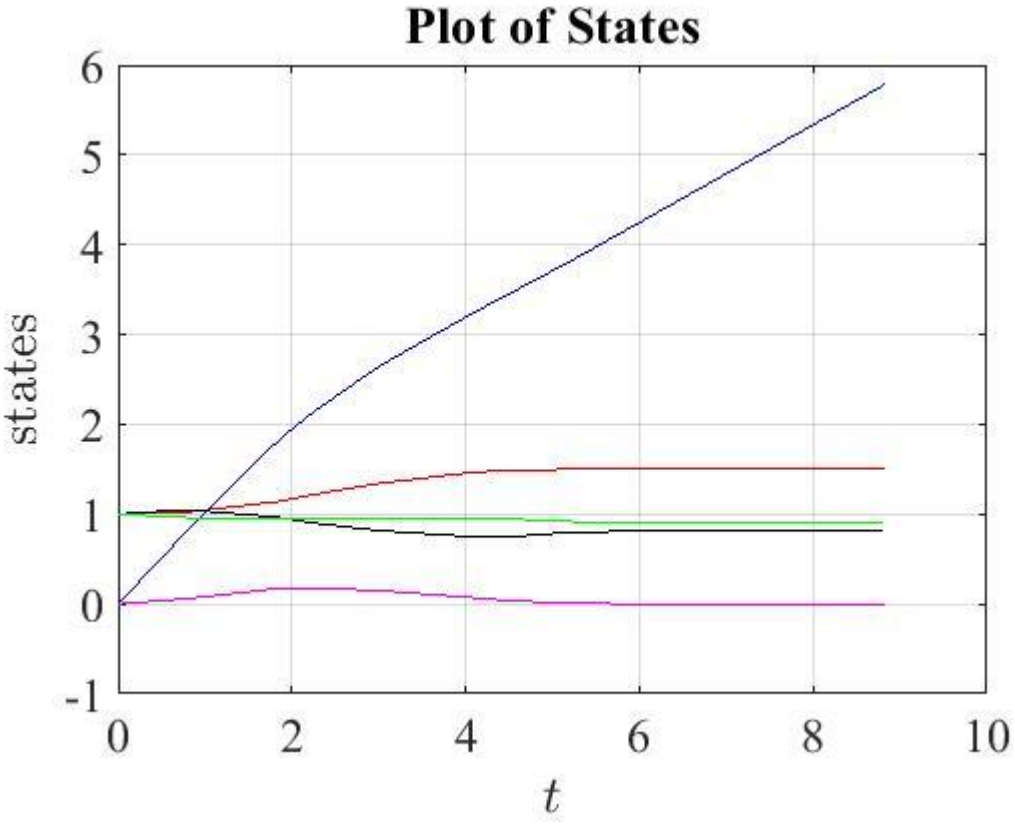

Throttle angle beta
0
-50
-100
-150
$\beta(t)$
0
2
4
6
8
t

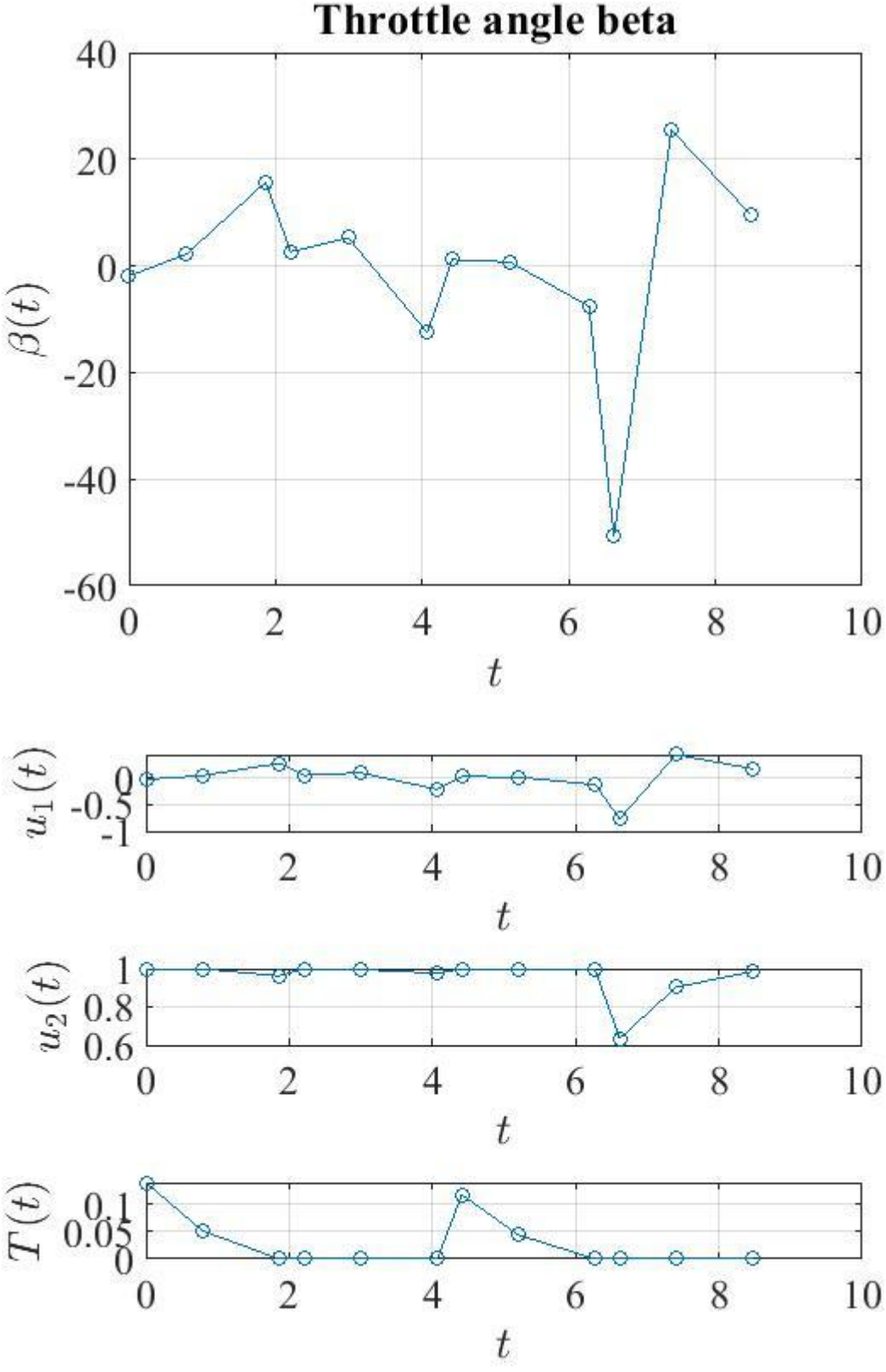

0.2
0
-0.2
-0.4
-0.6
$u_1(t)$
0
5
t
1
0.5
0
-0.5
-1
$u_2(t)$
0
5
t
0.15
0.1
0.05
0
$T(t)$
0
5
t

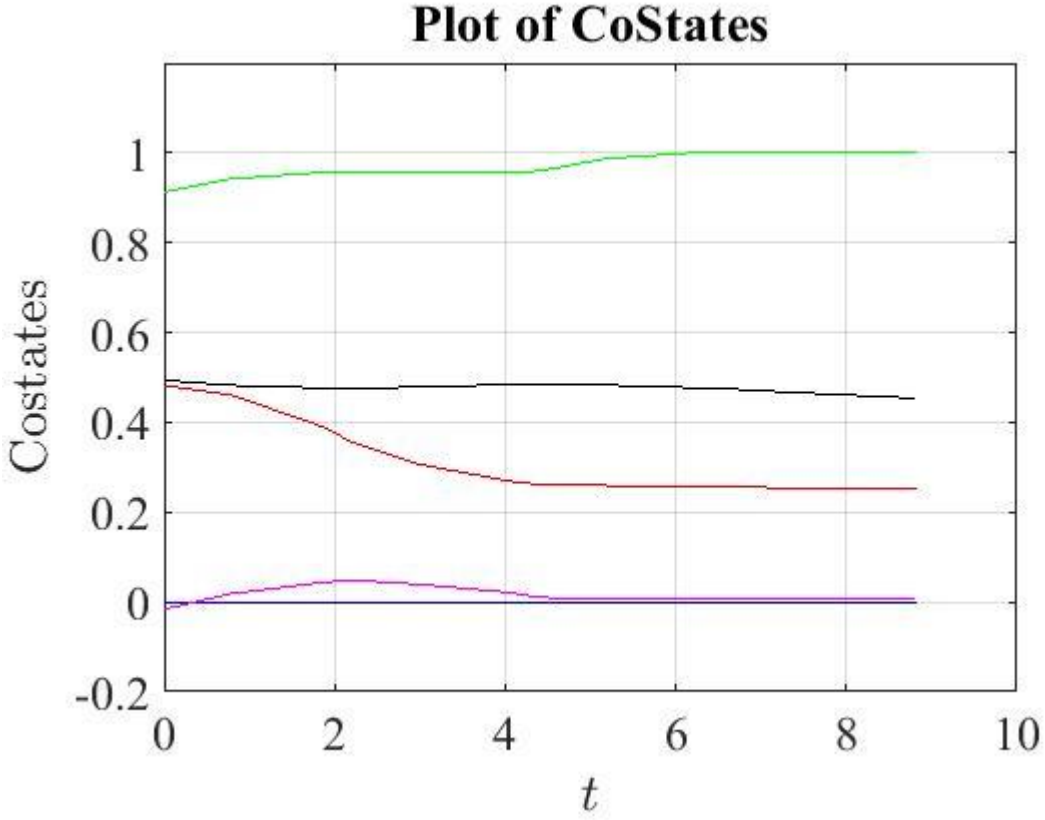

Plot of CoStates
Costates
1
0.8
0.6
0.4
0.2
0
-0.2
0
2
4
6
8
t

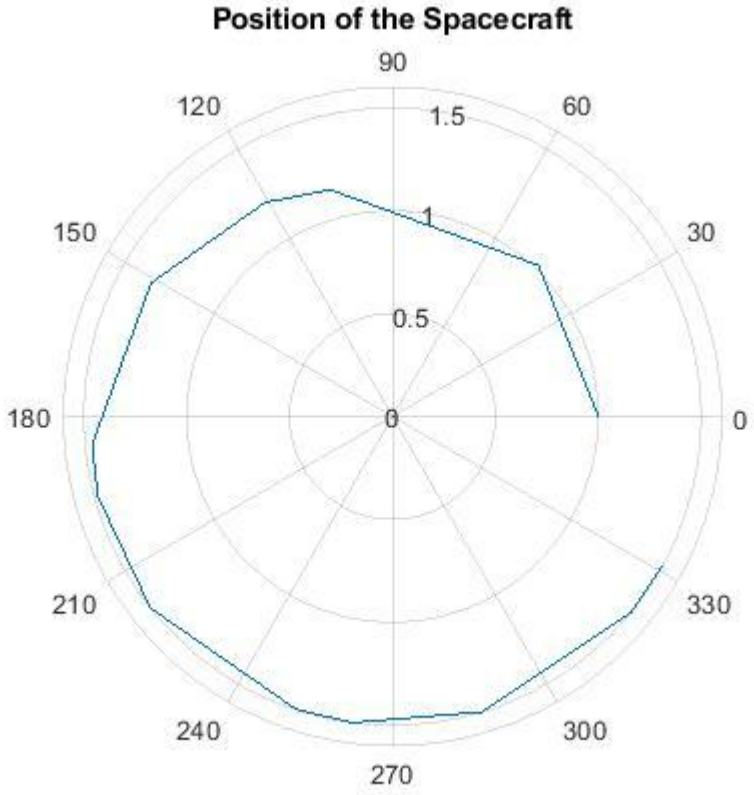

Position of the Spacecraft
90
120
60
1.5
150
30
1
0.5
180
0
210
330
240
300
270

For no of degrees of the polynomial (N) = 3 and No of intervals (K) = 4

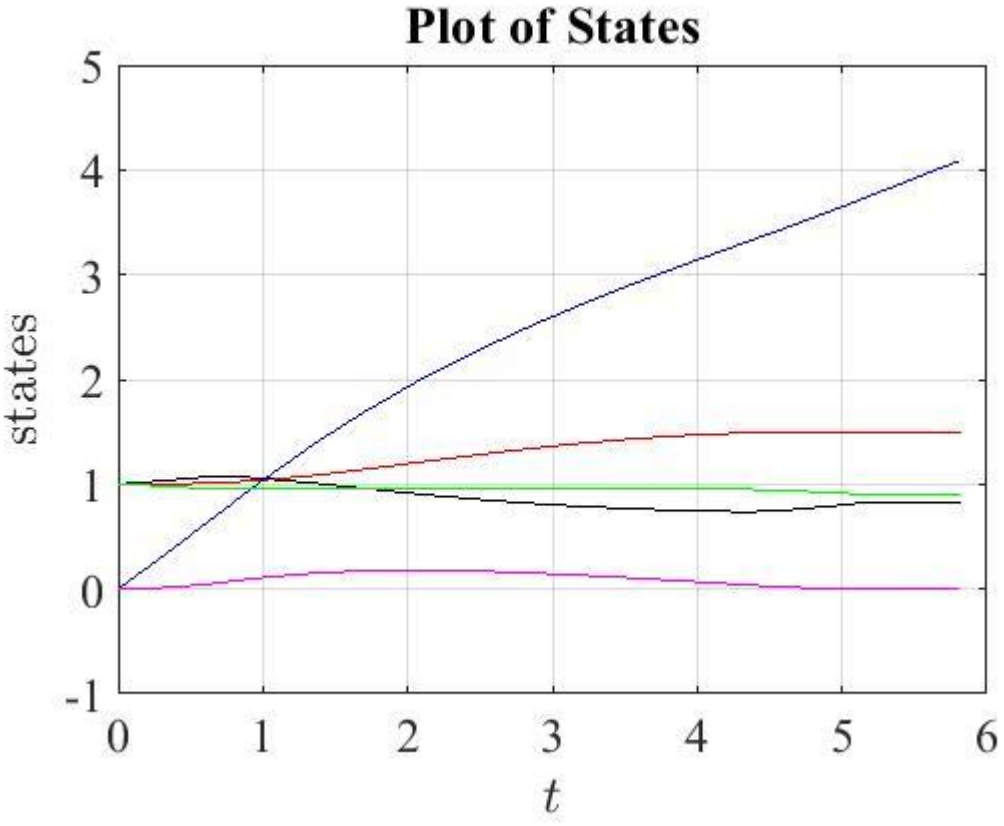


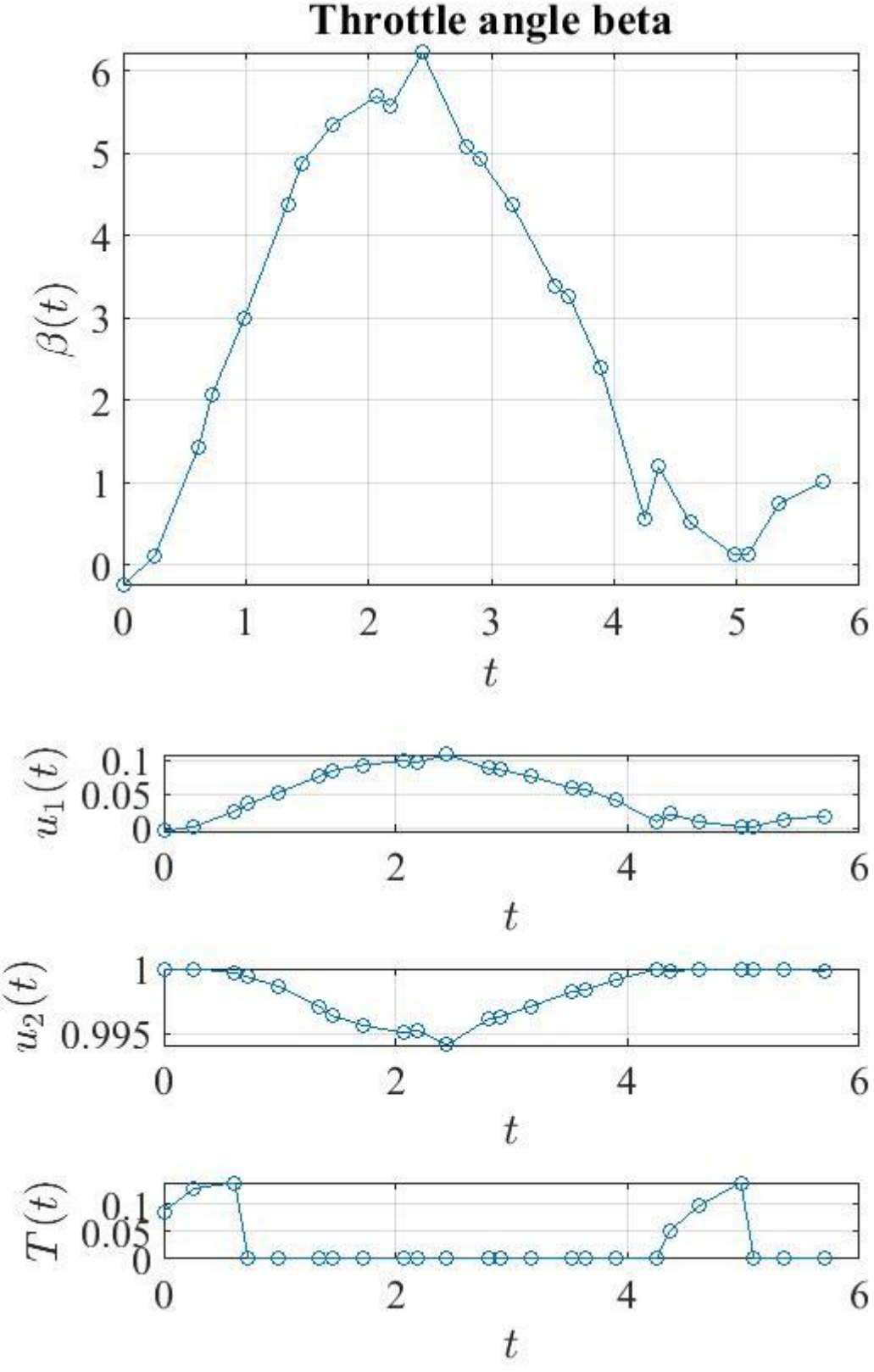

Throttle angle beta
40
20
0
-20
-40
-60
$\beta(t)$
0
2
4
6
8
10
t
$u_1(t)$
0
-0.5
-1
$u_2(t)$
1
0.8
0.6
$T(t)$
0.1
0.05
0

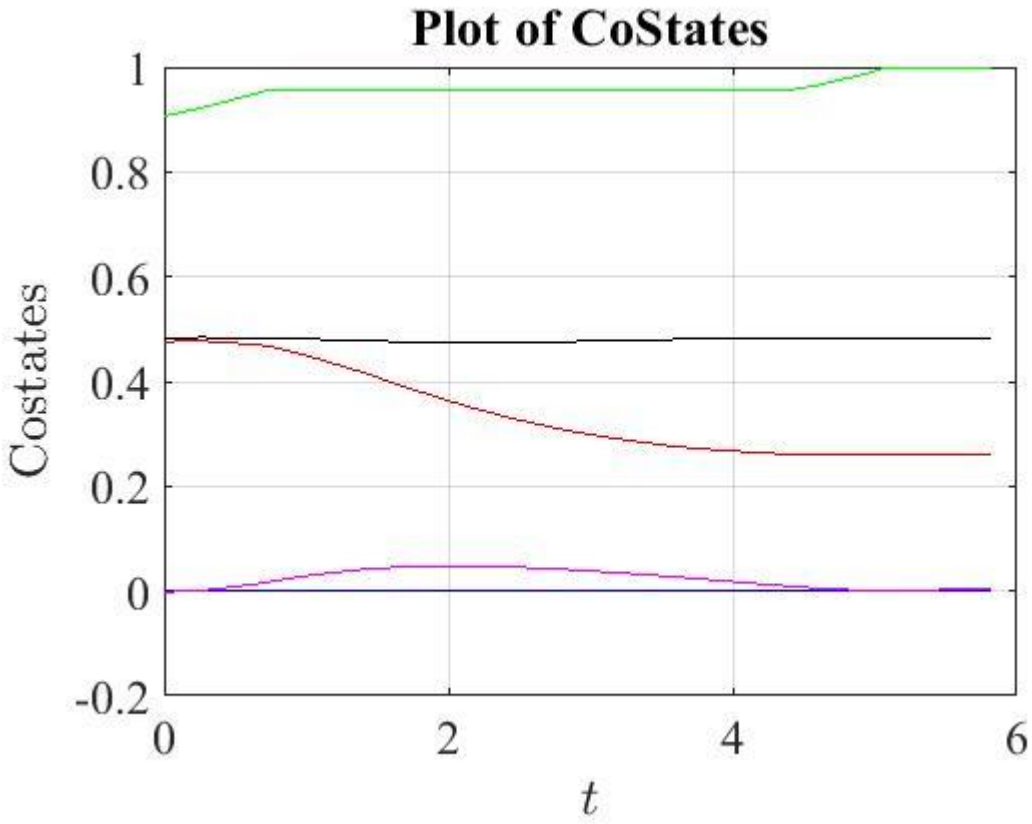

Plot of CoStates
Costates
t
1
0.8
0.6
0.4
0.2
0
-0.2
0
2
4
6
8
10

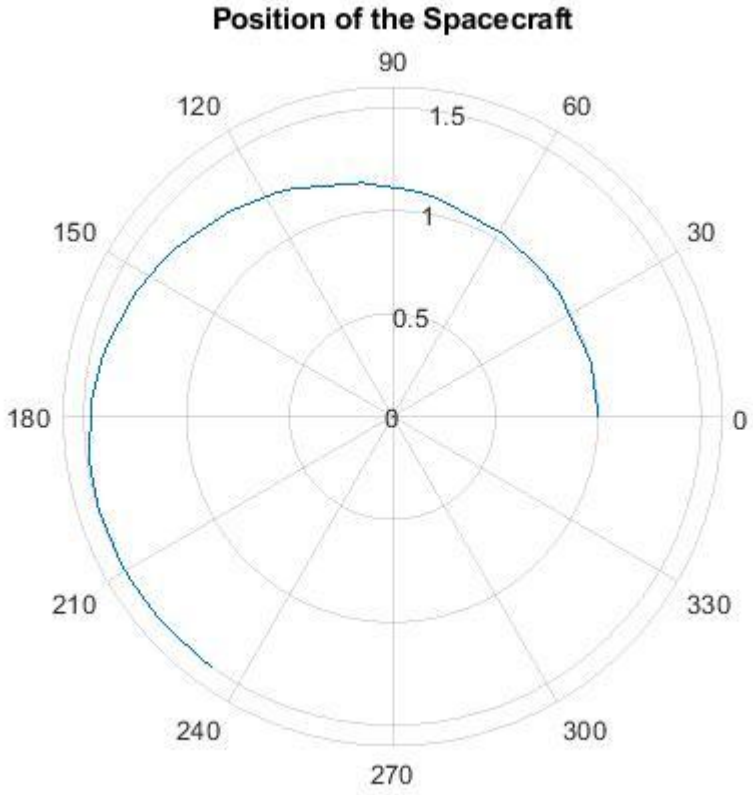

Position of the Spacecraft
90
120
60
150
30
180
0
210
330
240
300
270
1.5
1
0.5
0

For no of degrees of the polynomial (N) = 3 and No of intervals (K) = 8



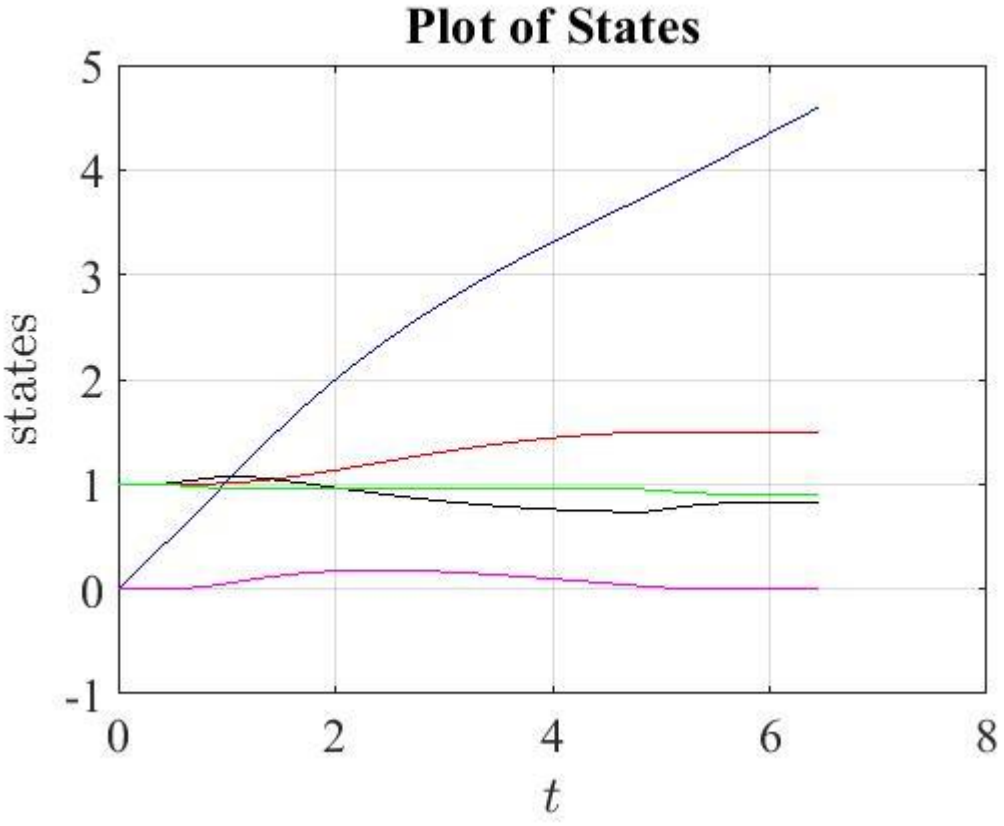

Throttle angle beta
β(t)
t
u1(t)
t
u2(t)
t
T(t)
t

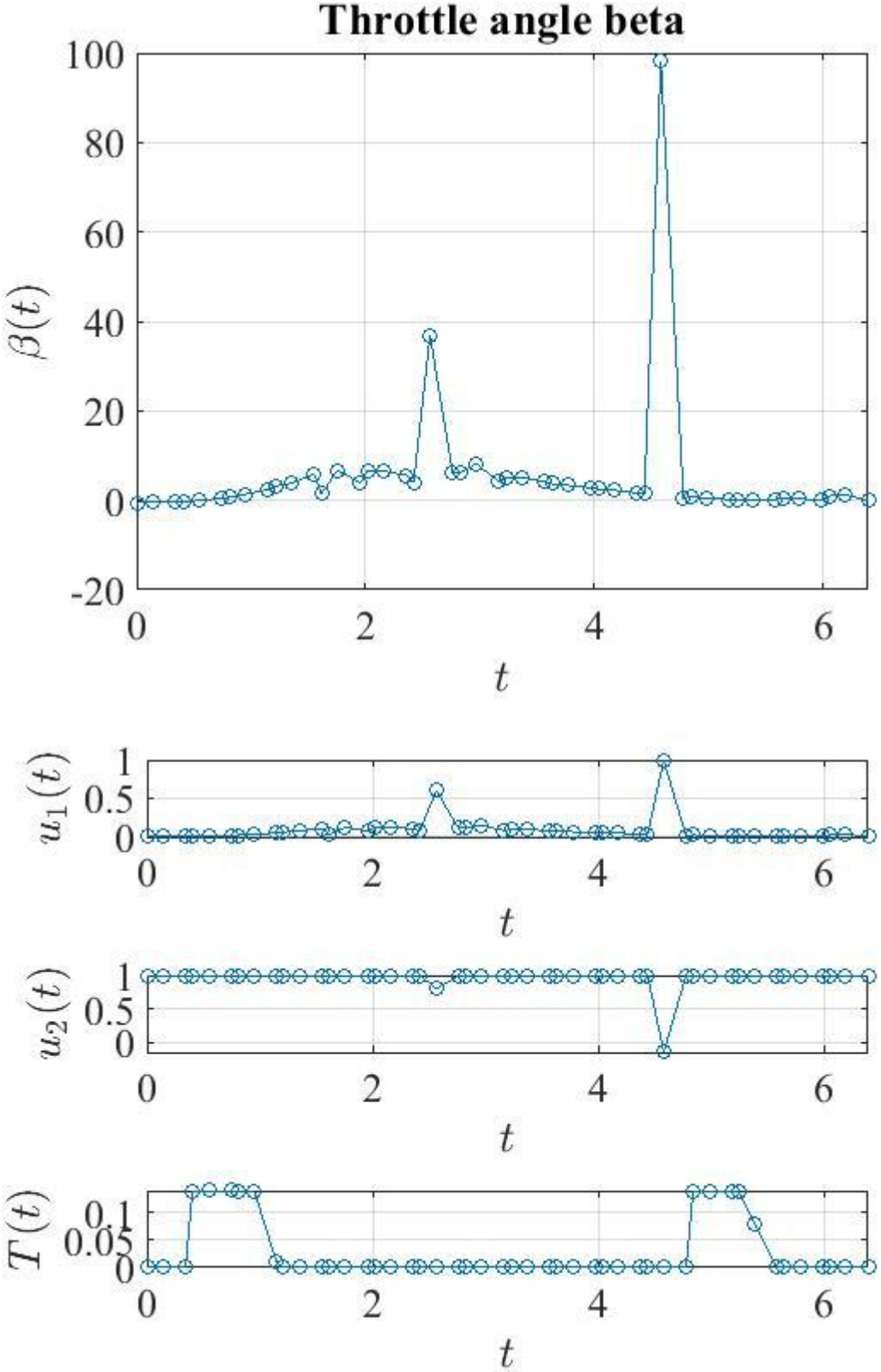


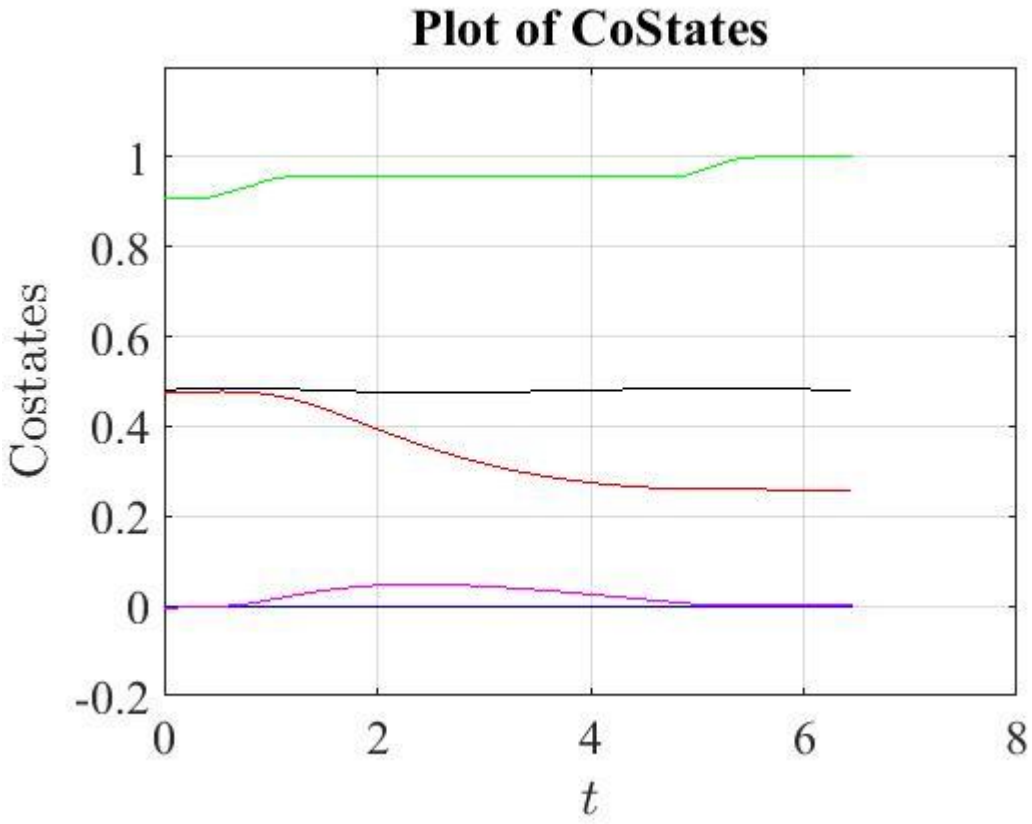


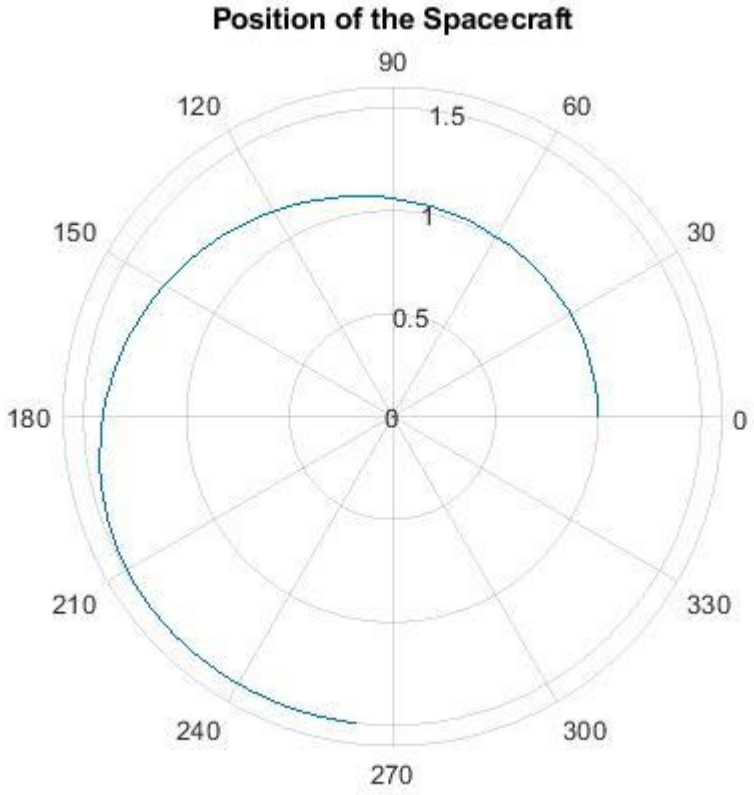
Published with MATLAB® R2021a

For no of degrees of the polynomial (N) = 3 and No of intervals (K) = 16



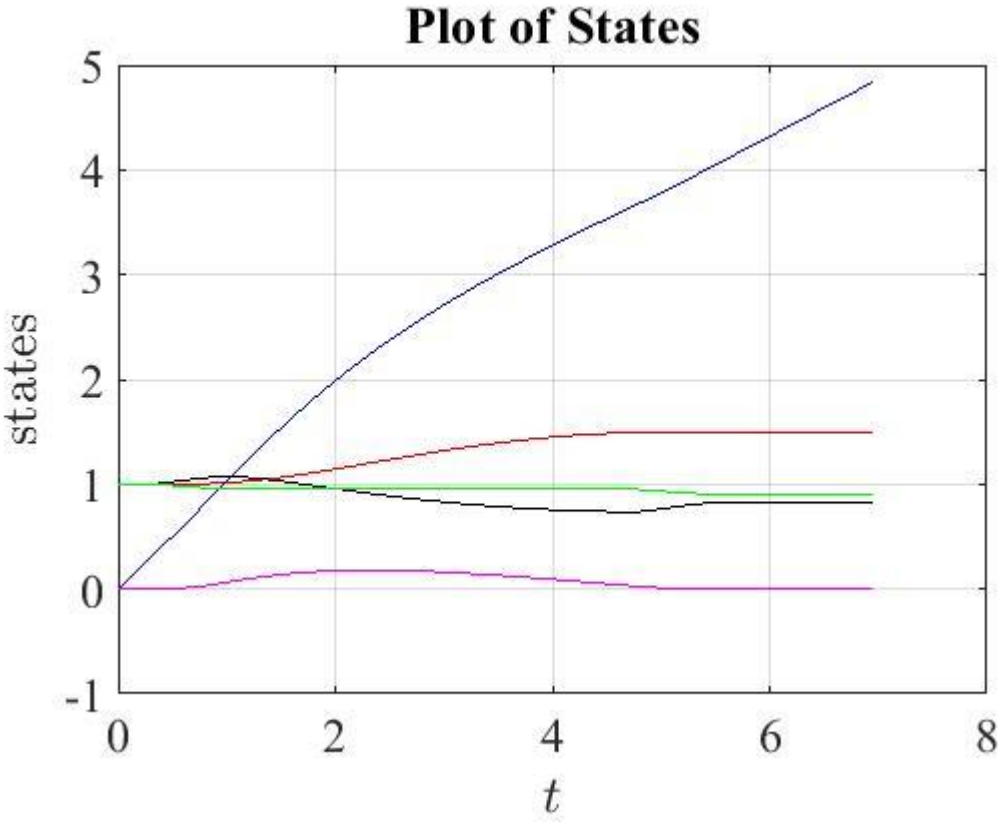

Throttle angle beta
100
80
60
40
20
0
-20
$\beta(t)$
0
2
4
6
t
1
0.5
0
$u_1(t)$
0
2
4
6
t
1
0.5
0
$u_2(t)$
0
2
4
6
t
0.1
0.05
0
$T(t)$
0
2
4
6
t

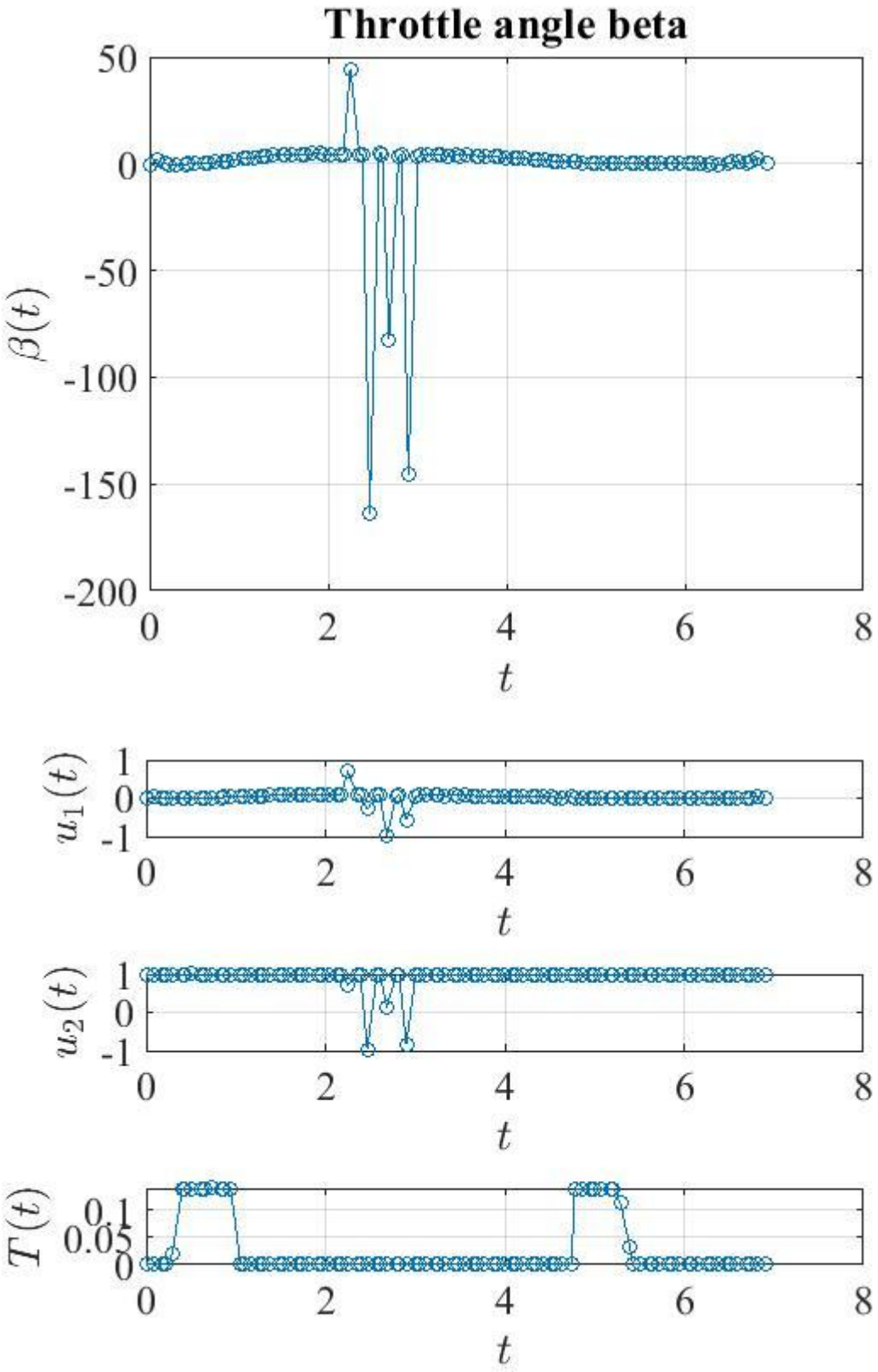


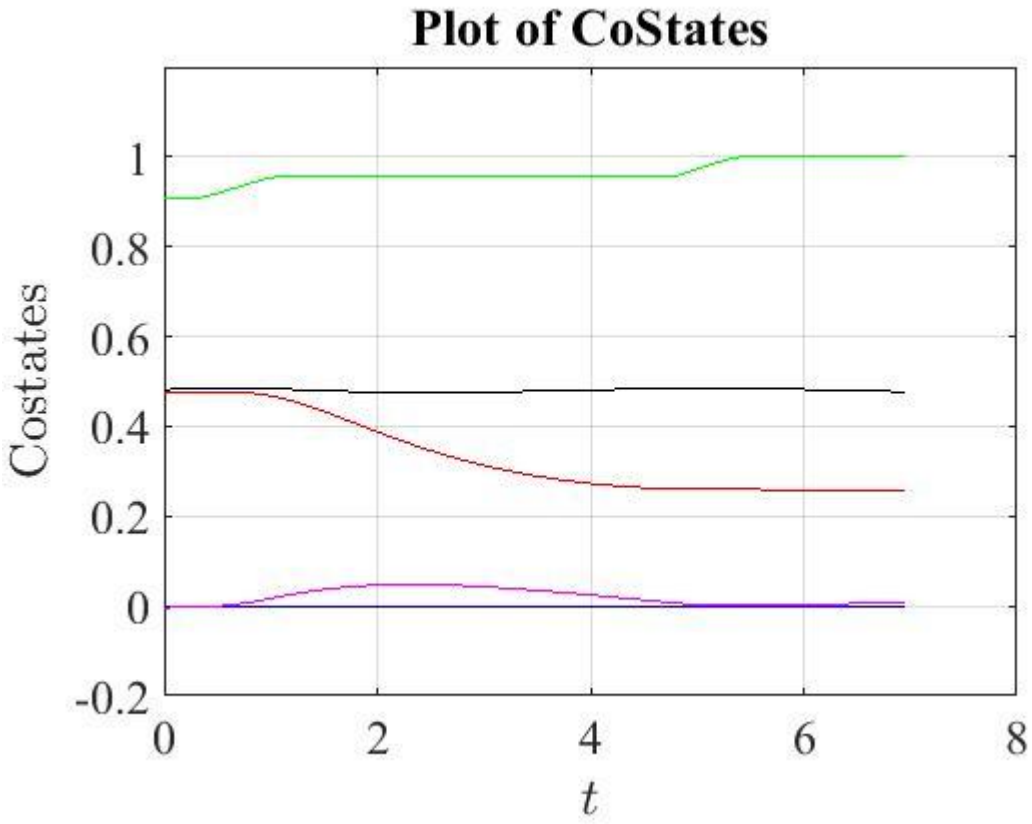


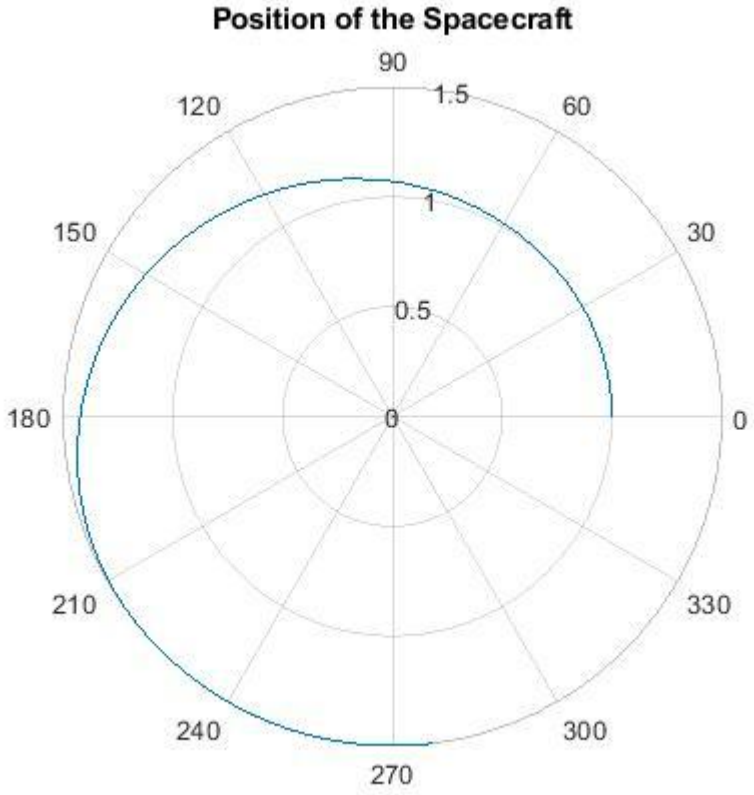
Published with MATLAB® R2021a

For no of degrees of the polynomial (N) = 3 and No of intervals (K) = 32



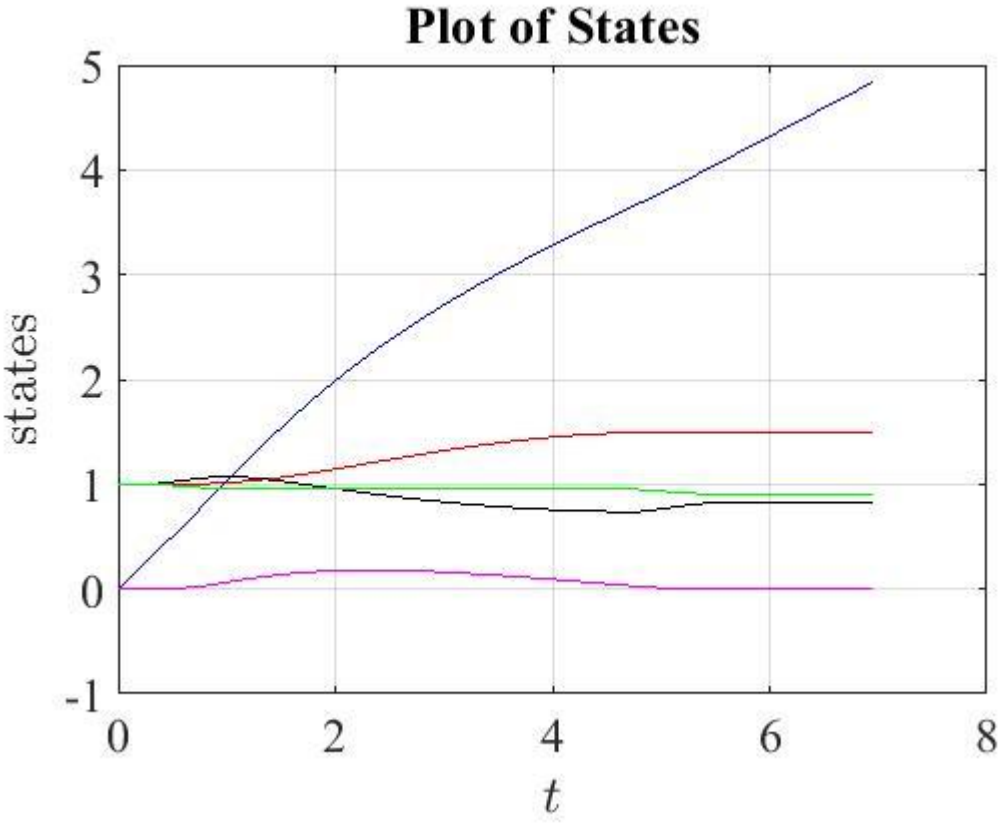

Throttle angle beta
50
0
-50
-100
-150
-200
$\beta(t)$
0
2
4
6
8
t
1
0
-1
$u_1(t)$
$u_2(t)$
0.1
0.05
$T(t)$

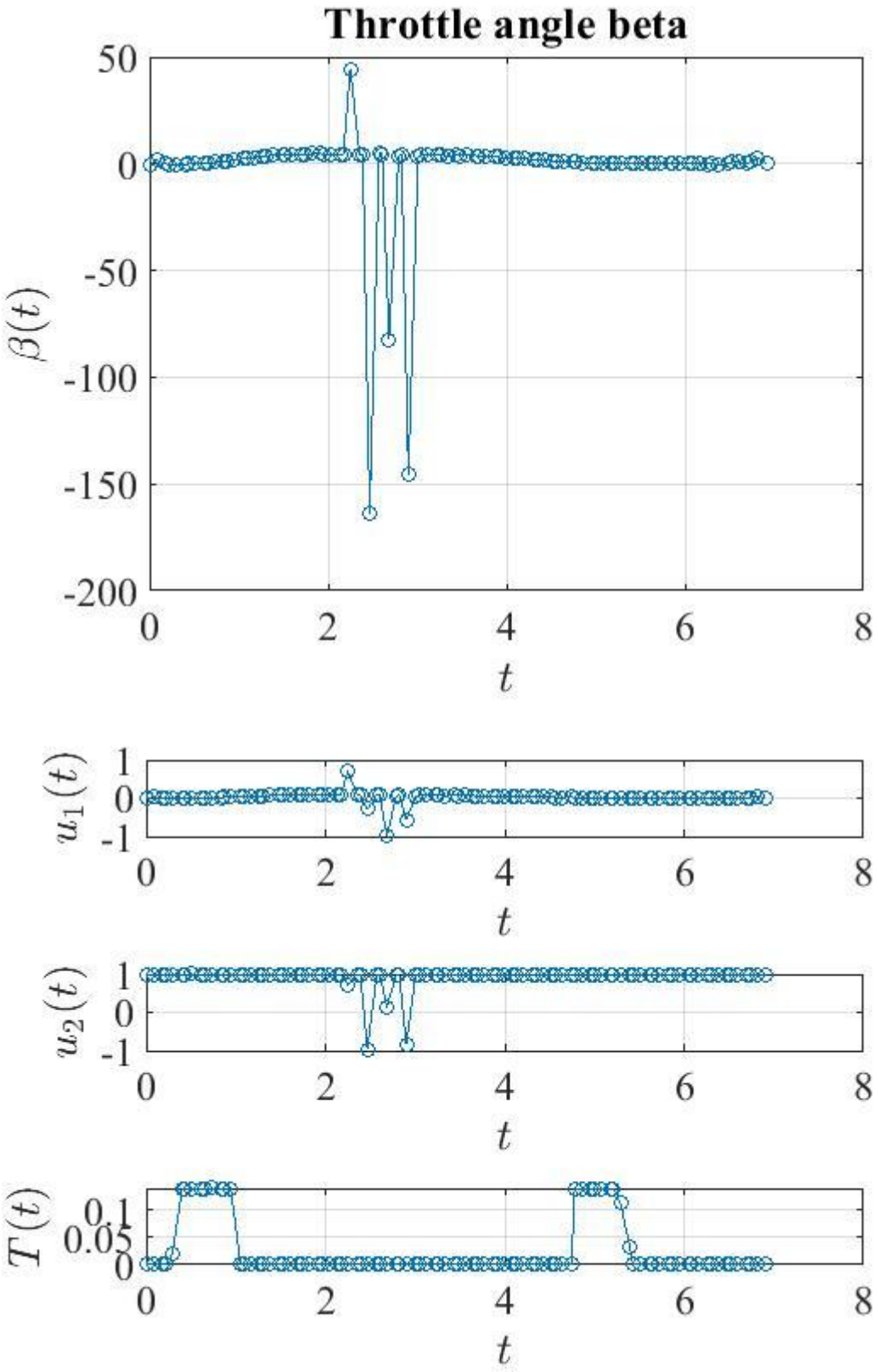


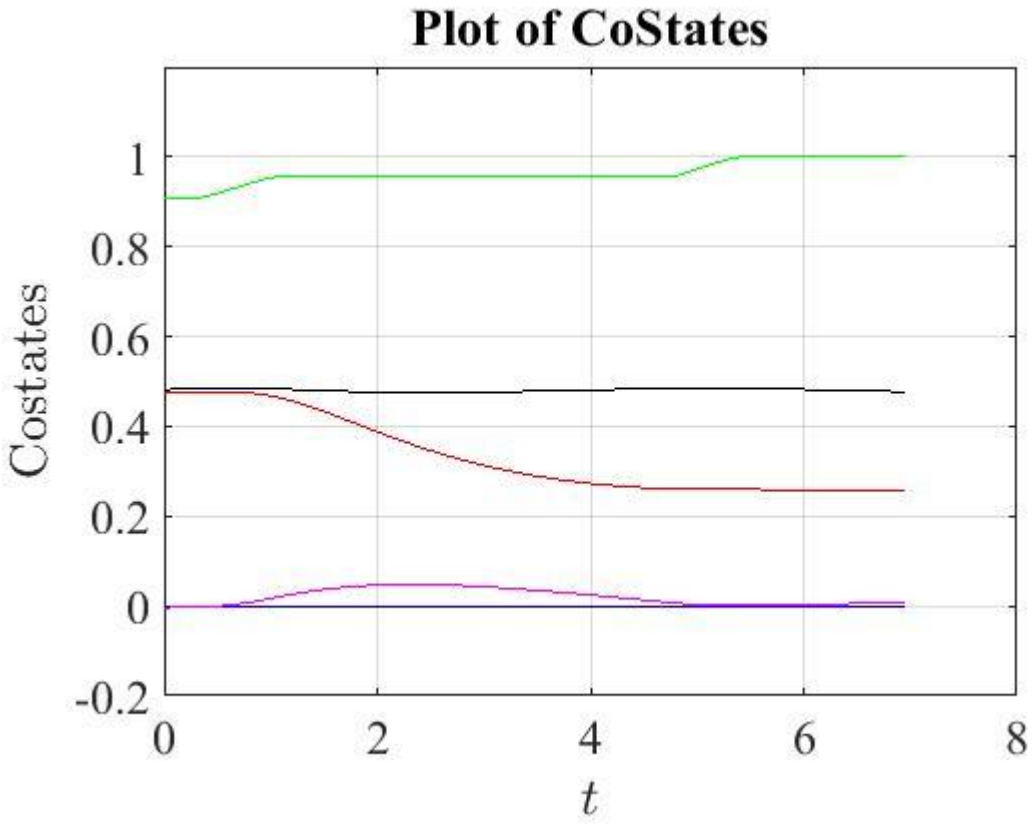


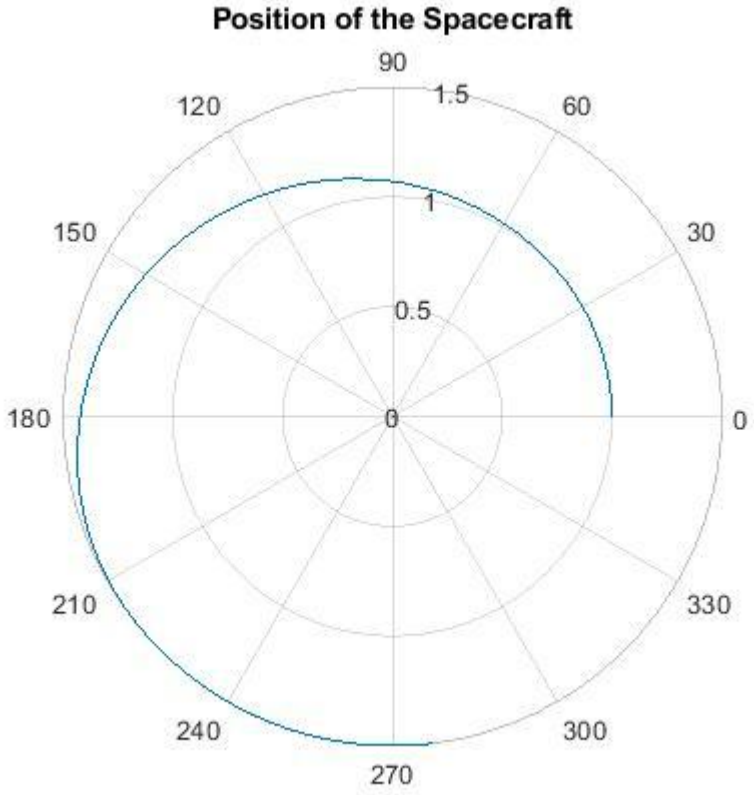
Published with MATLAB® R2021a

For no of degrees of the polynomial (N) = 3 and No of intervals (K) = 32



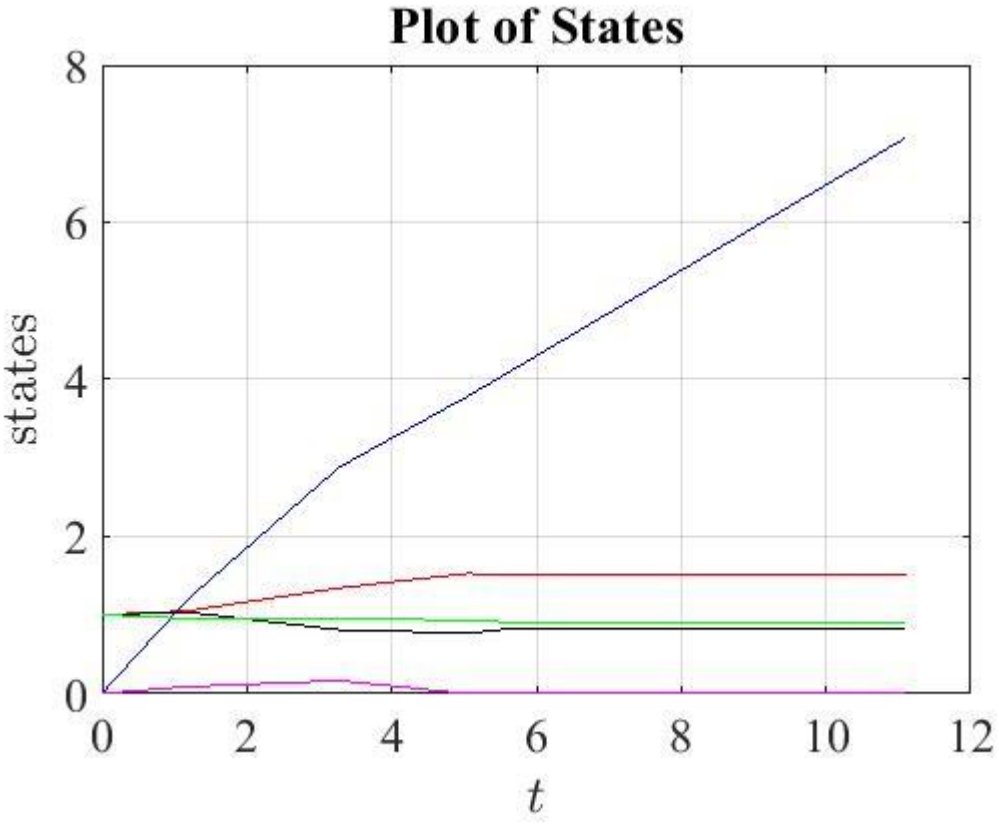

Throttle angle beta
50
0
-50
-100
-150
-200
$\beta(t)$
0
2
4
6
8
t
1
0
-1
$u_1(t)$
$u_2(t)$
0.1
0.05
$T(t)$

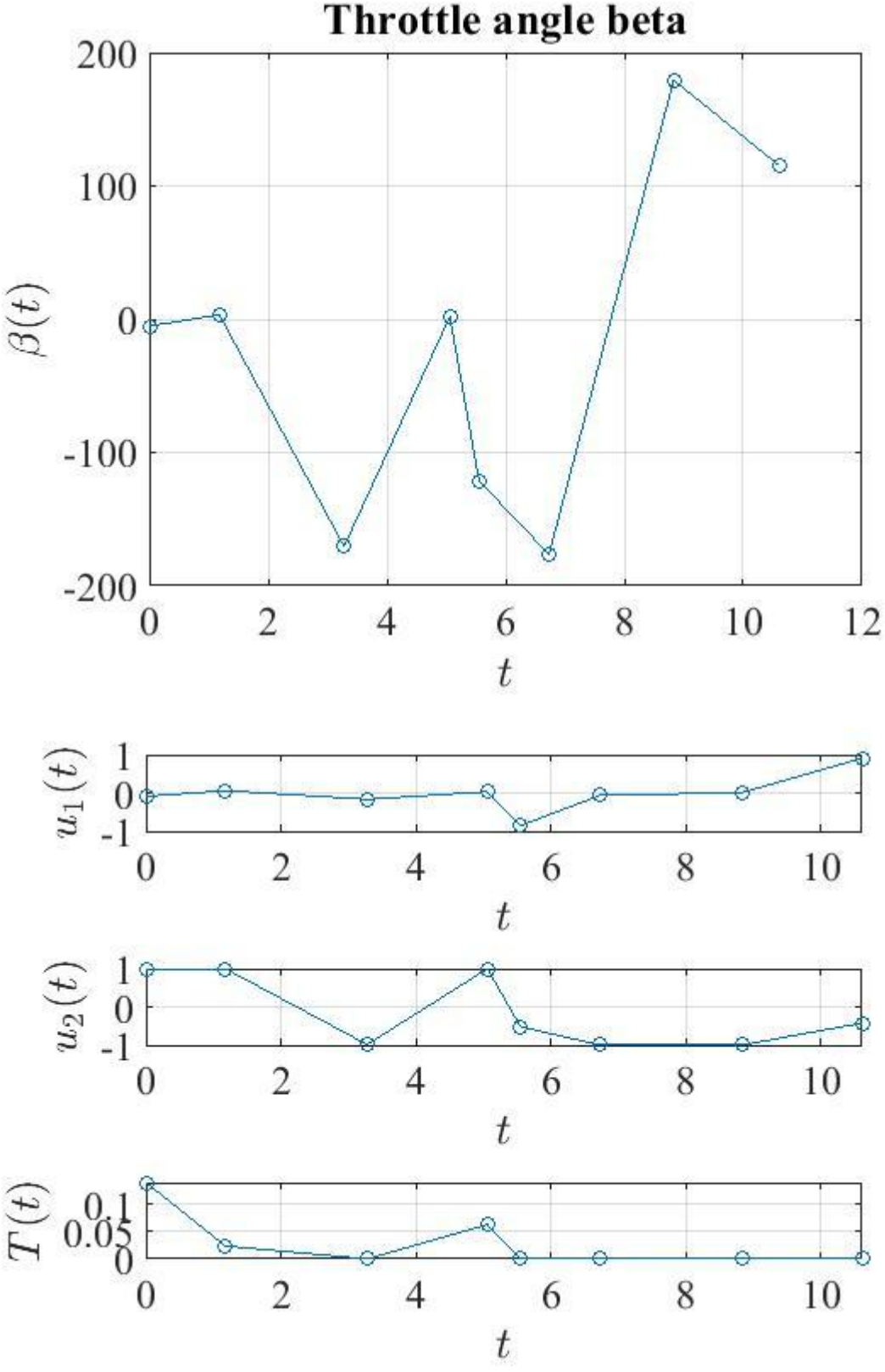


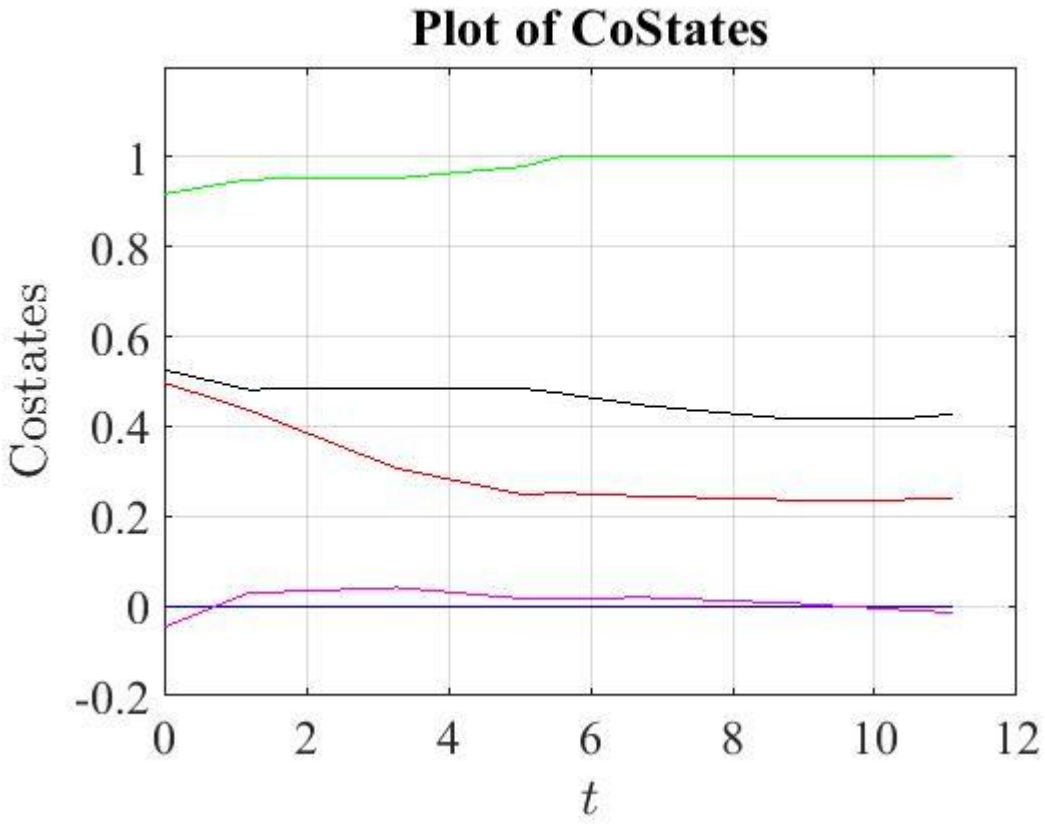


Published with MATLAB® R2021a

For no of degrees of the polynomial (N) = 4 and No of intervals (K) = 2

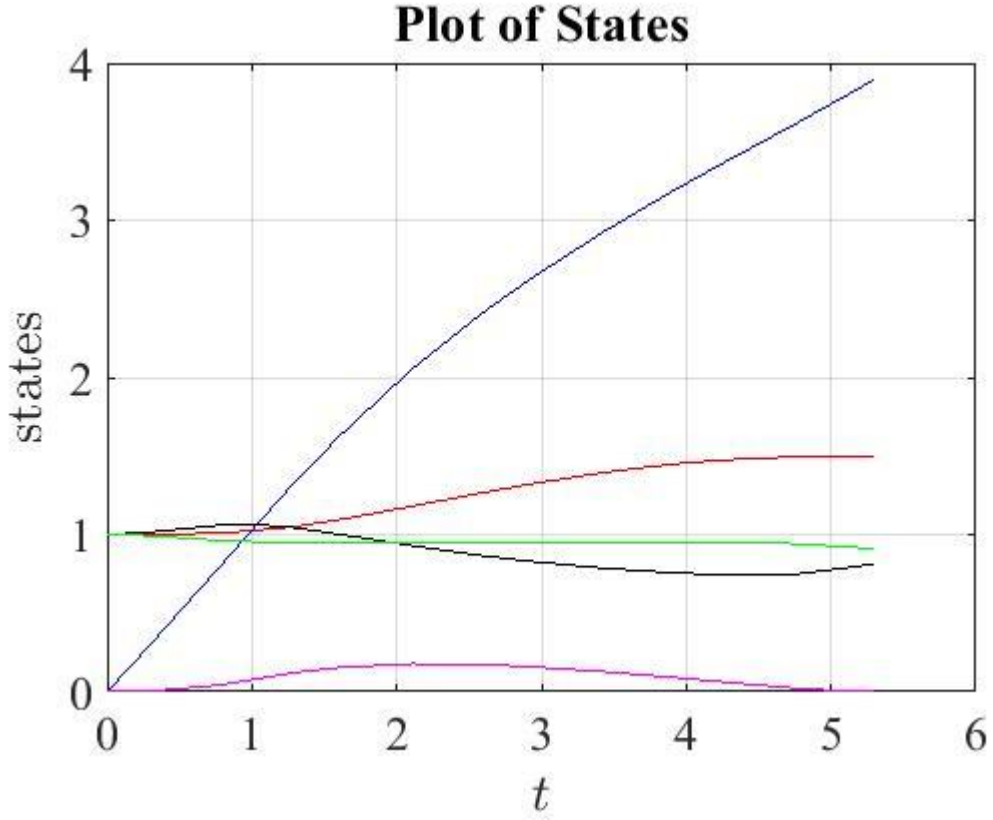


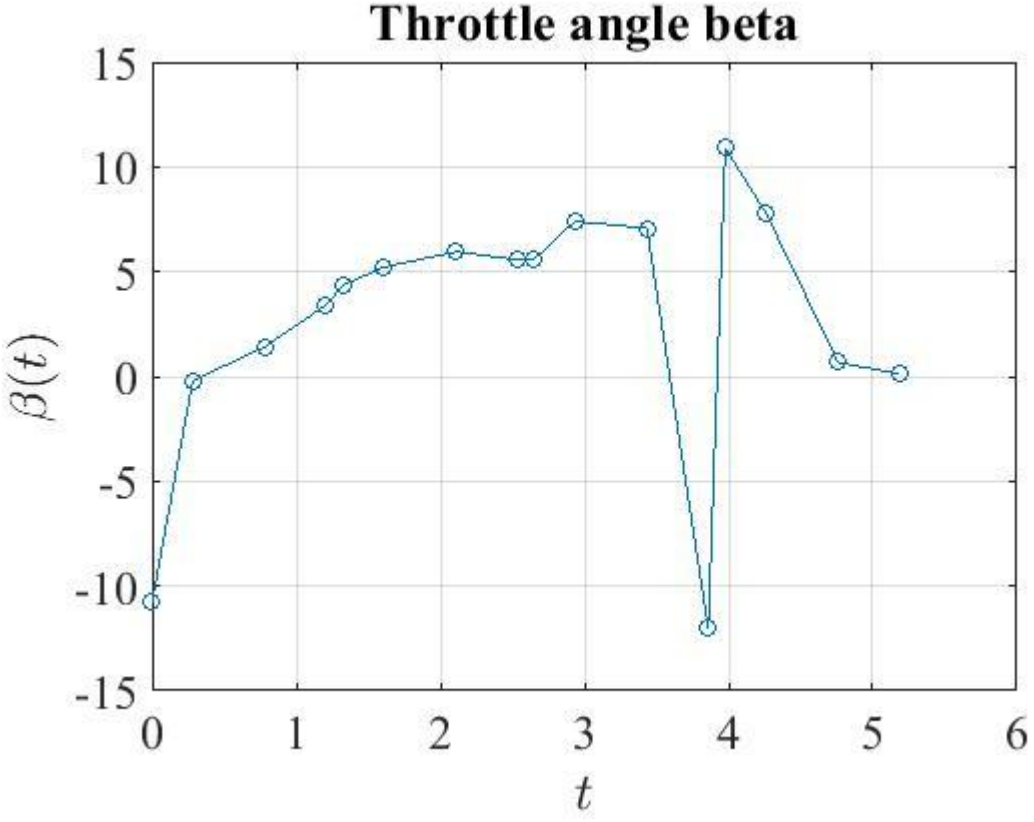

Throttle angle beta
200
100
0
-100
-200
β(t)
0
2
4
6
8
10
12
t
u1(t)
1
0
-1
u2(t)
T(t)
0.1
0.05

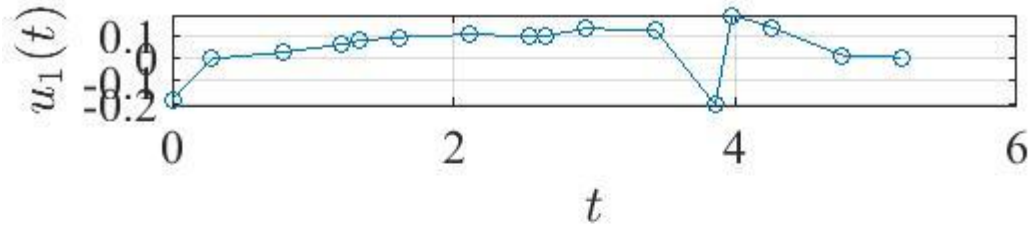


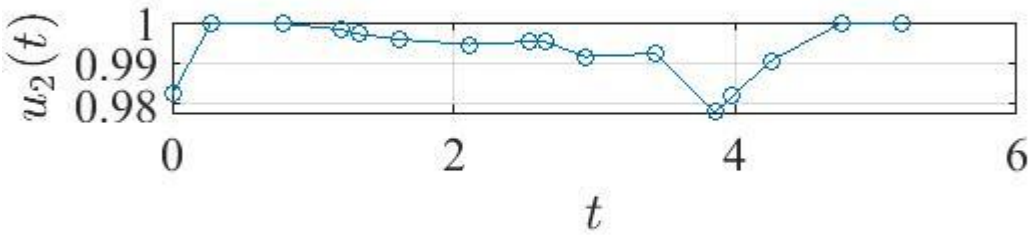
Published with MATLAB® R2021a

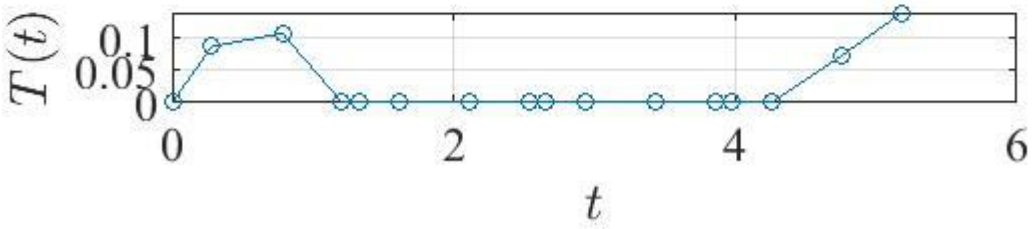
For no of degrees of the polynomial (N) = 4 and No of intervals (K) = 4

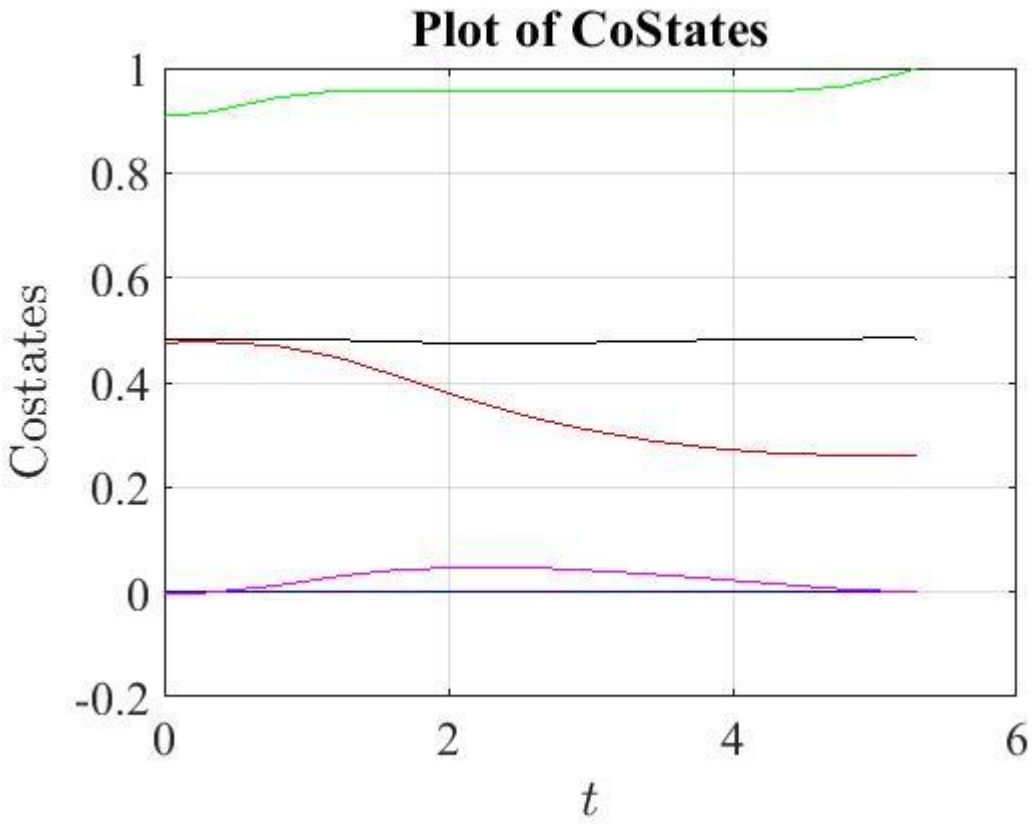

Plot of States
states
t
0
1
2
3
4
5
6


Throttle angle beta
β(t)
t
15
10
5
0
-5
-10
-15
0
1
2
3
4
5
6



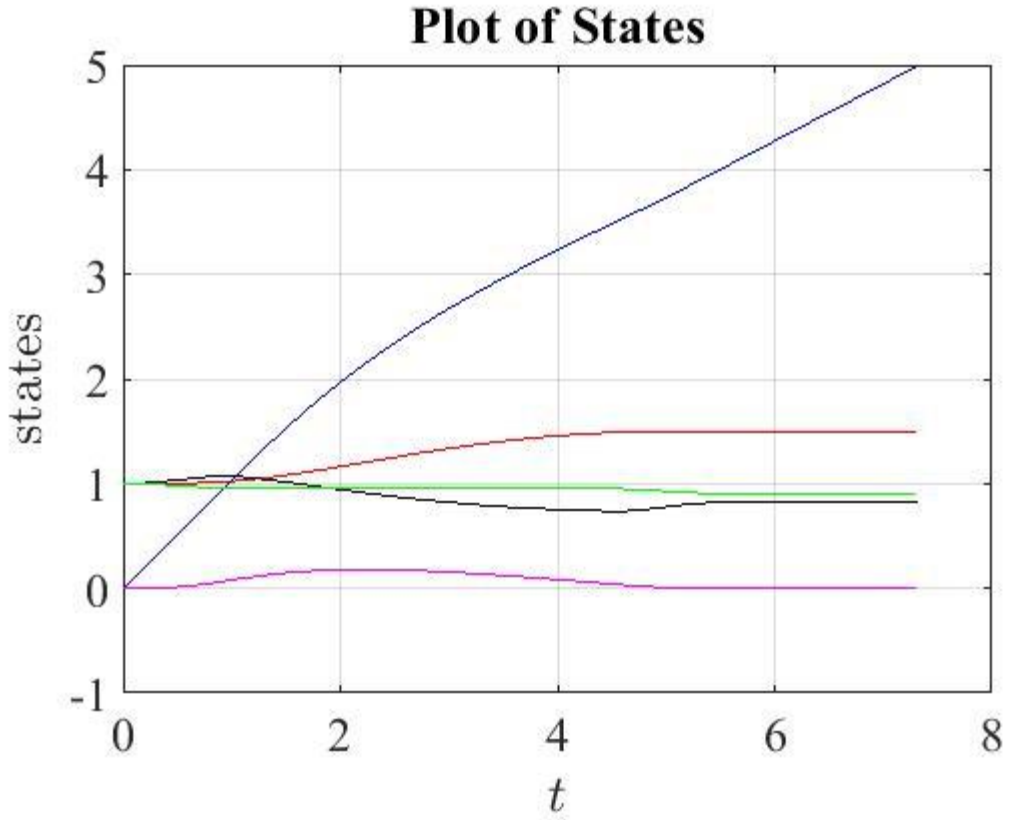


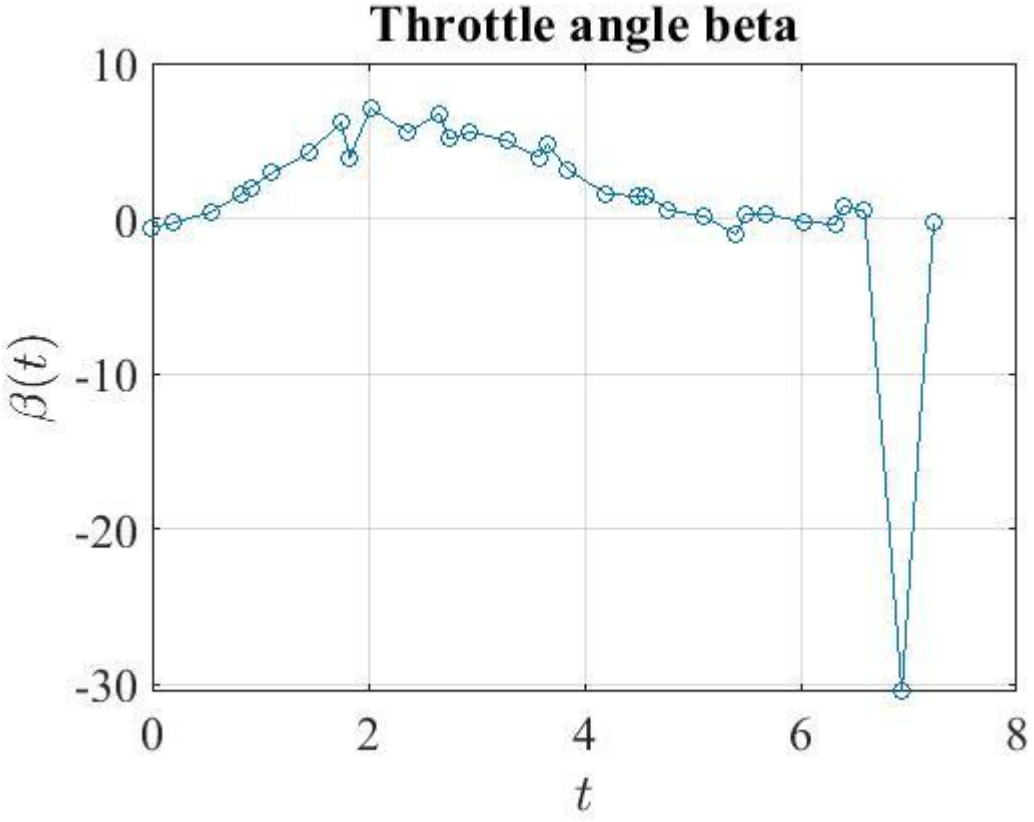


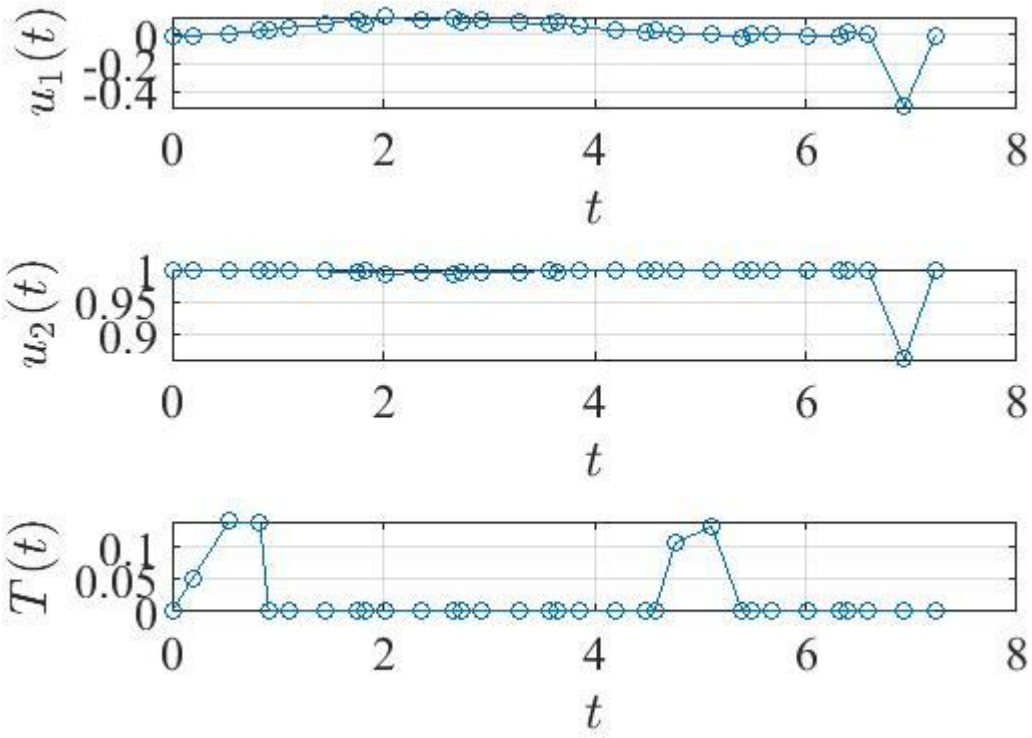


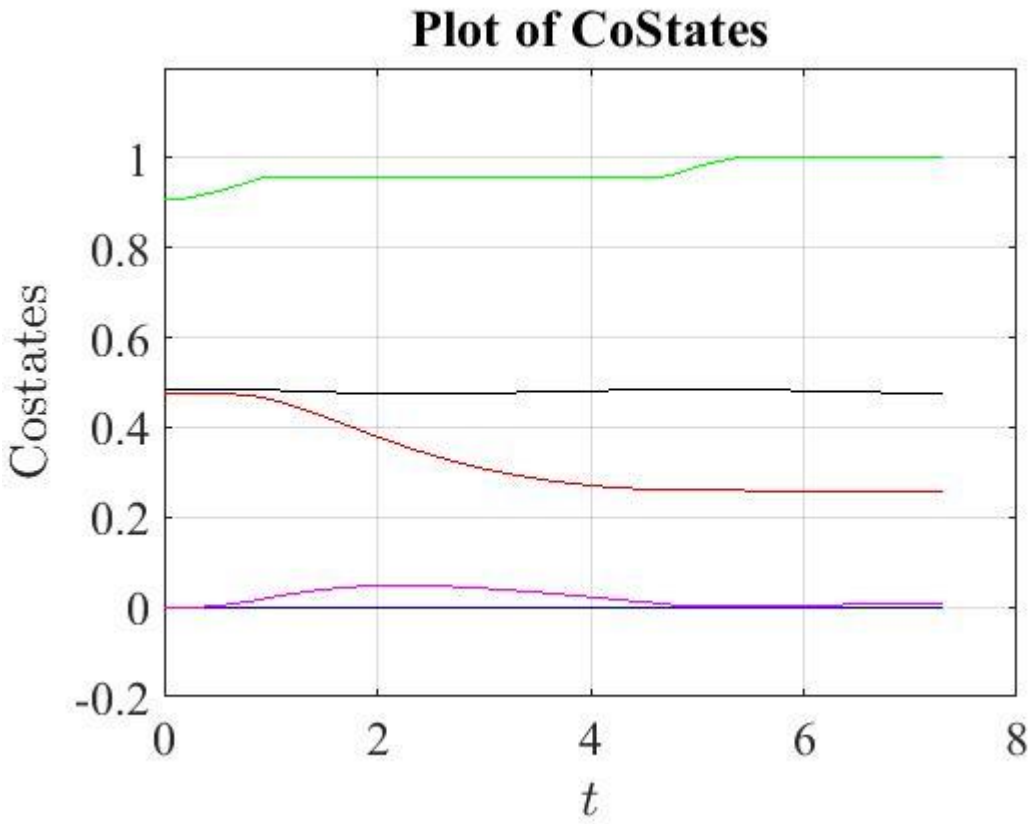
Published with MATLAB® R2021a

For no of degrees of the polynomial (N) = 4 and No of intervals (K) = 8

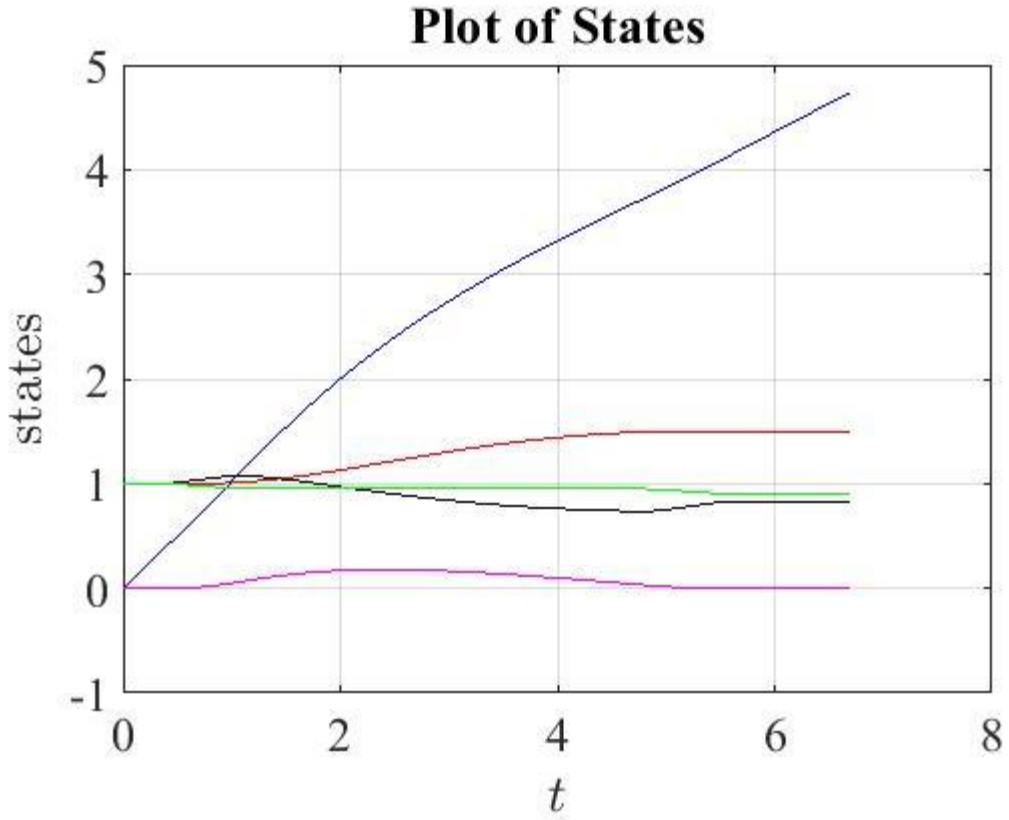


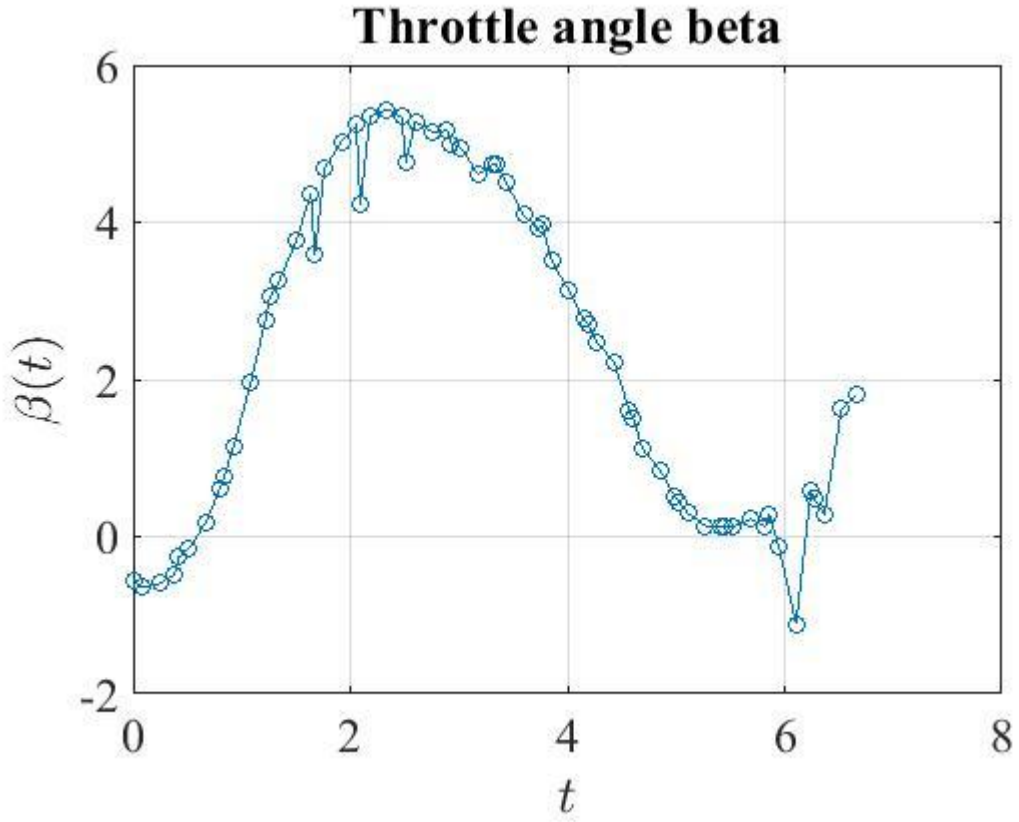


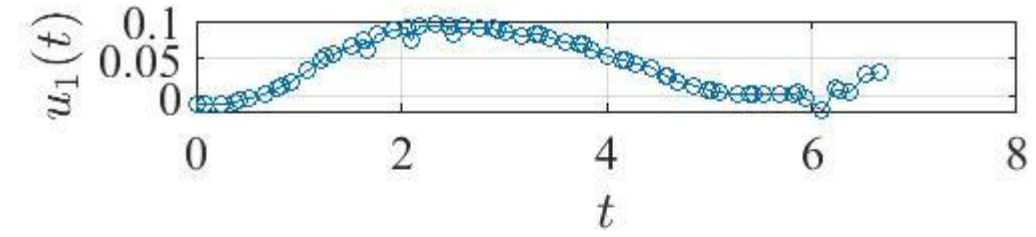


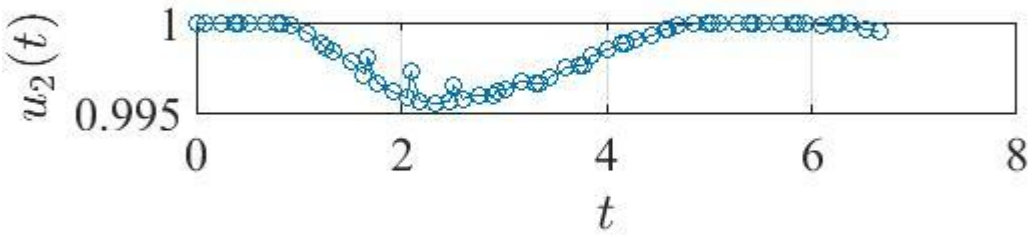


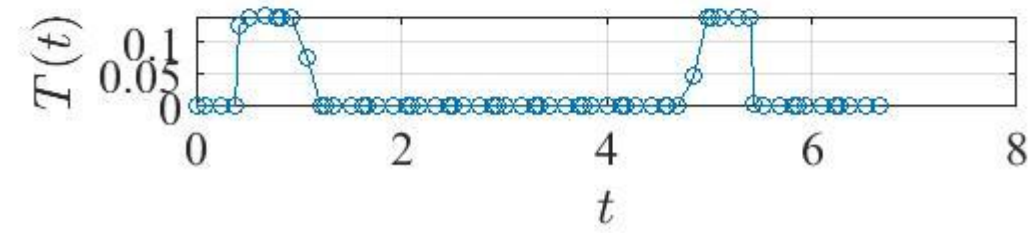
For no of degrees of the polynomial (N) = 4 and No of intervals (K) = 16

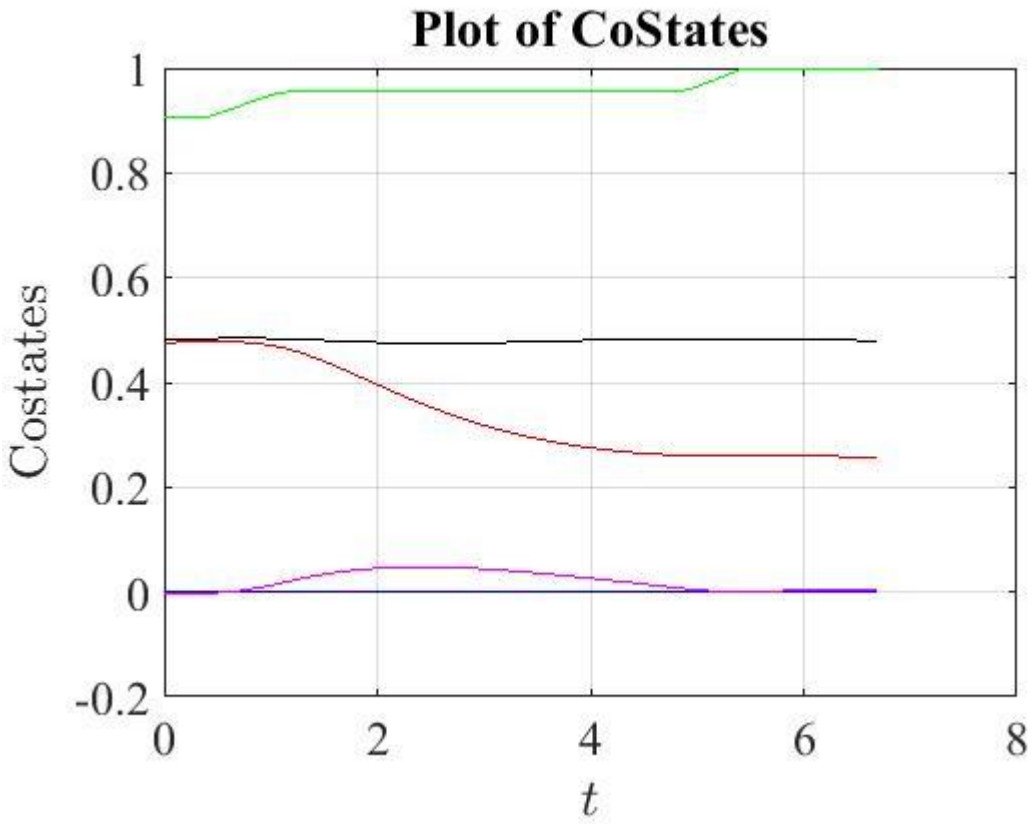






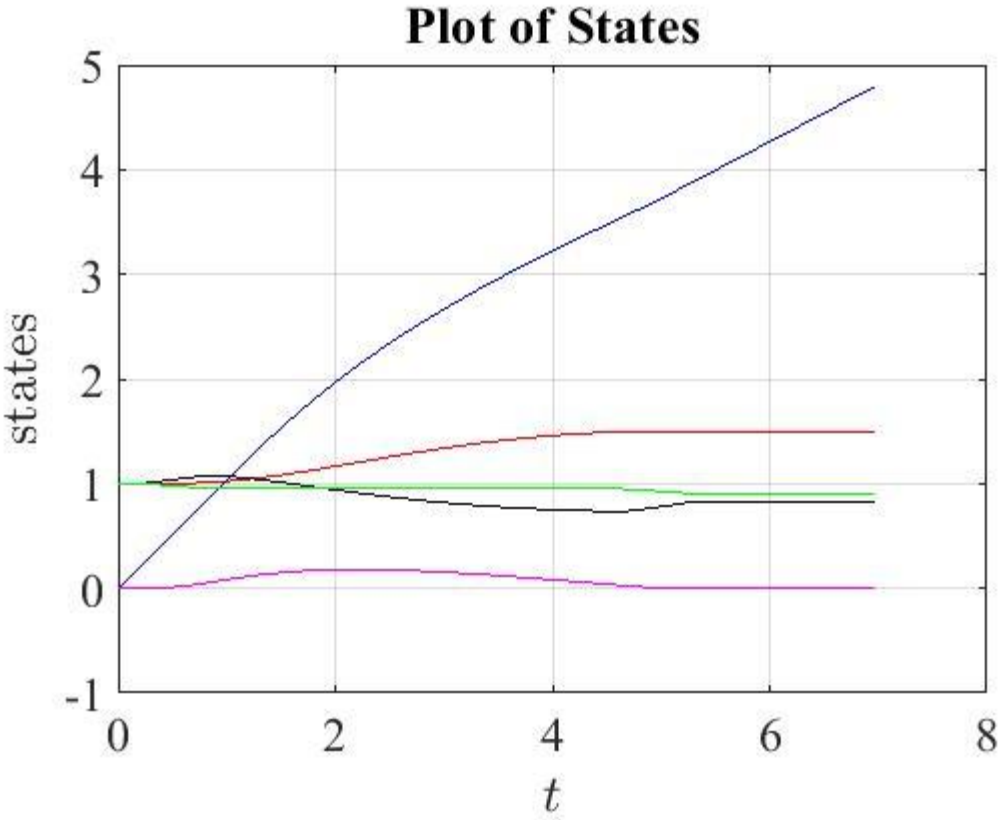


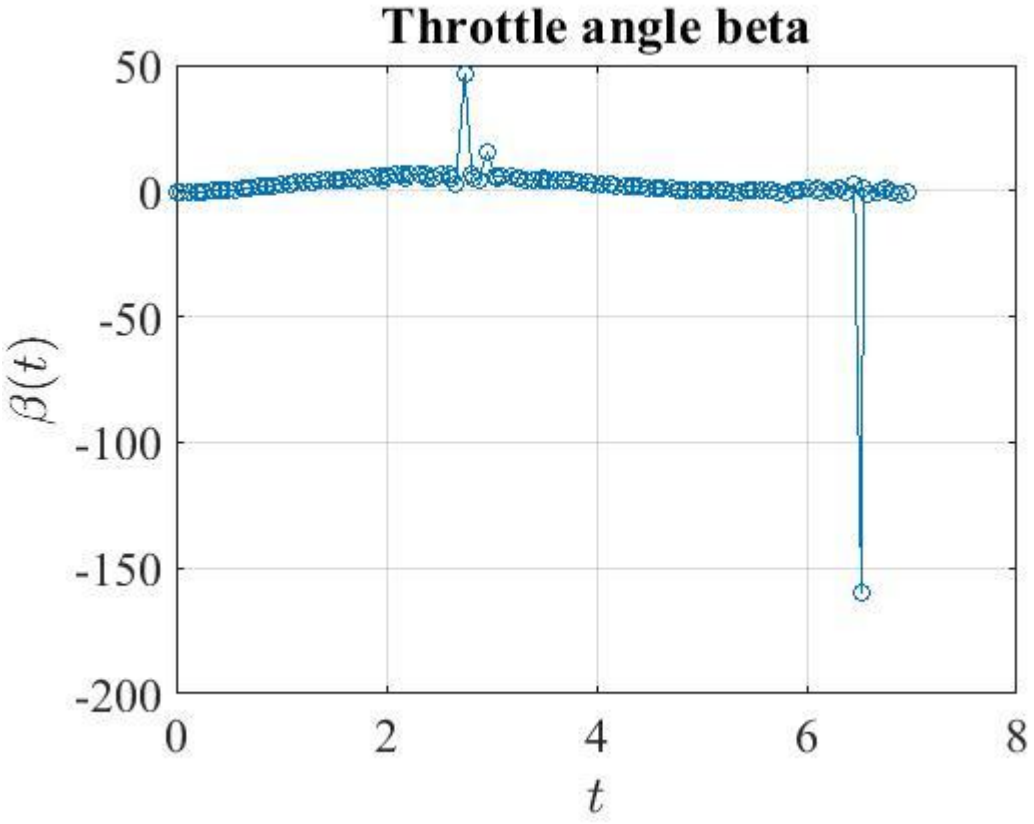


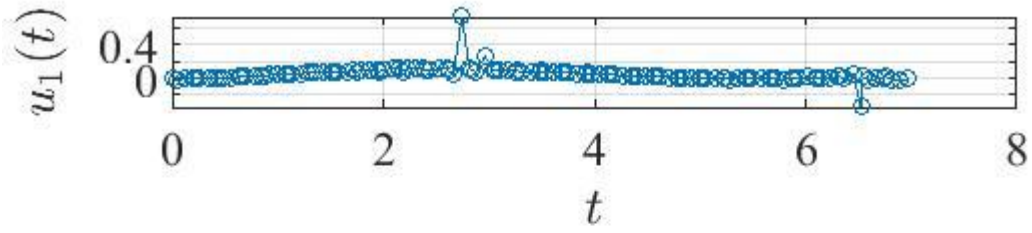


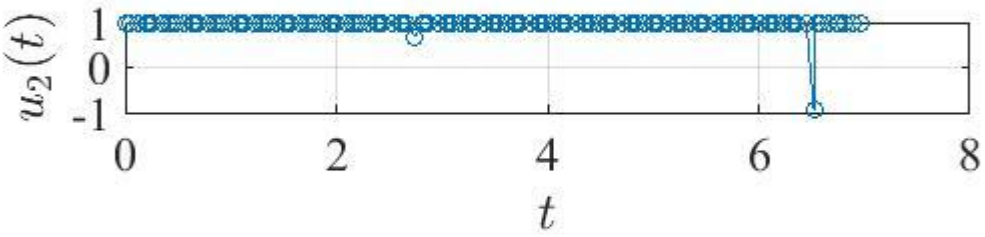
Published with MATLAB® R2021a

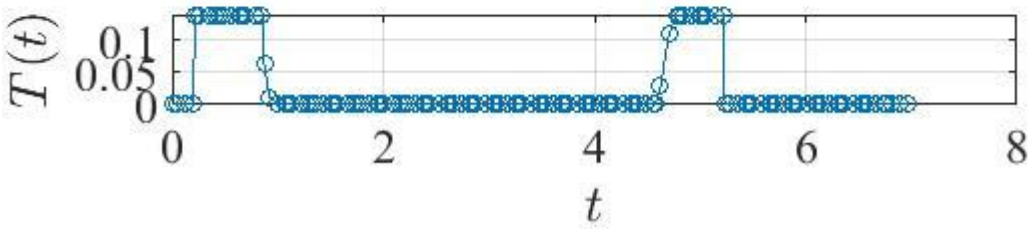
For no of degrees of the polynomial (N) = 4 and No of intervals (K) = 32

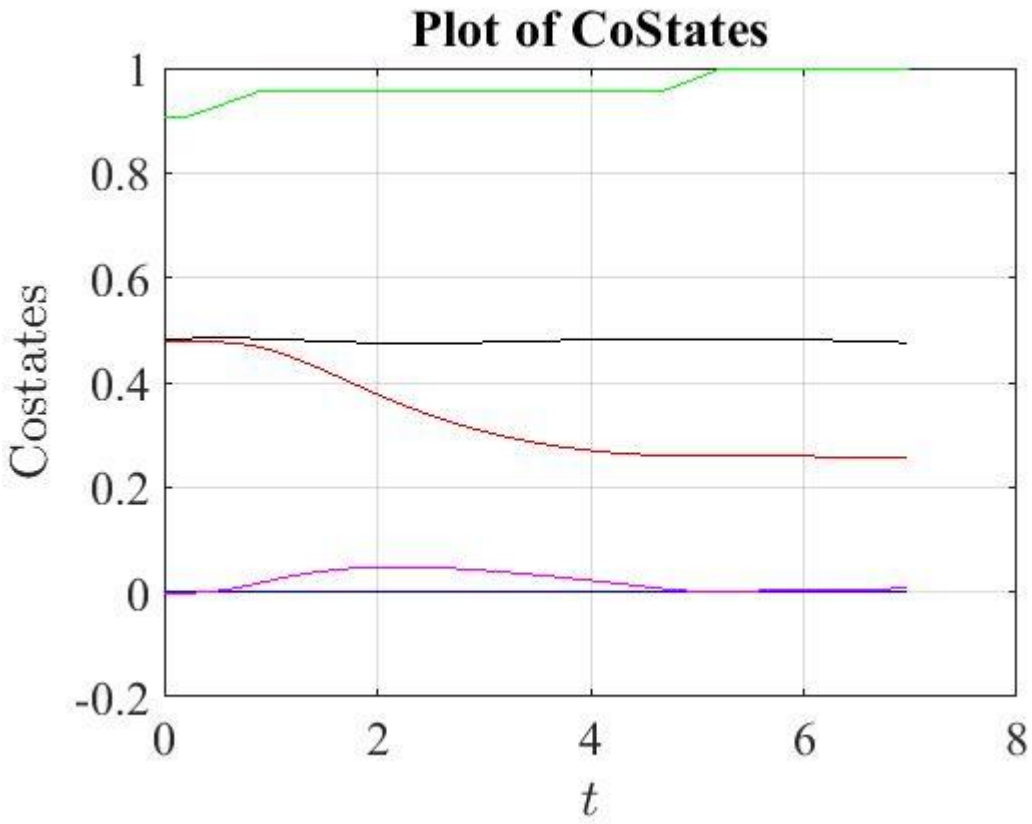








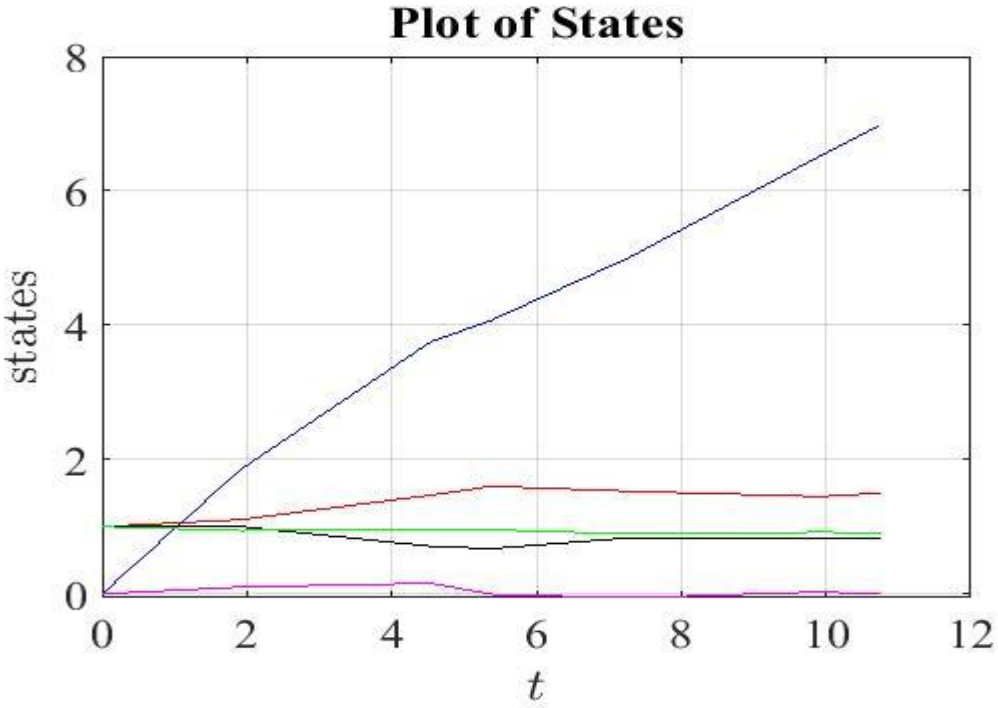


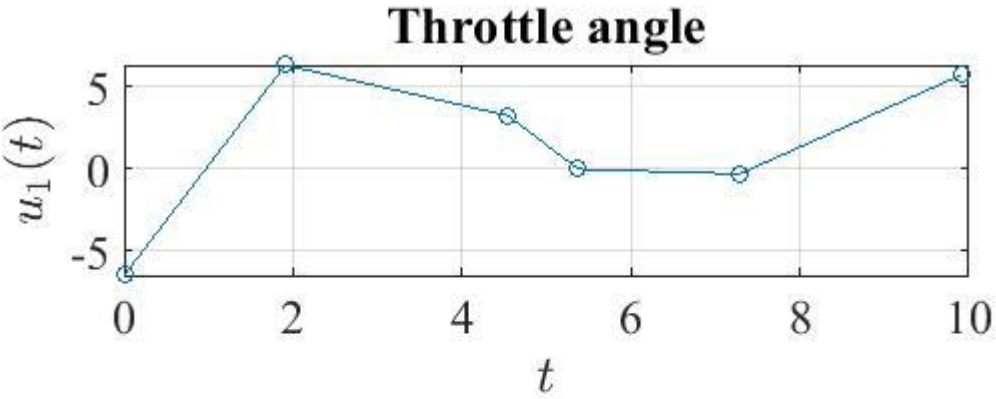


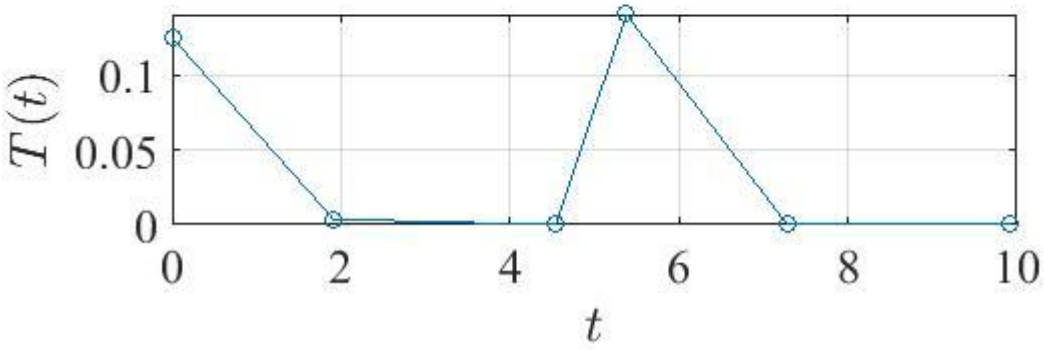
Published with MATLAB® R2021a

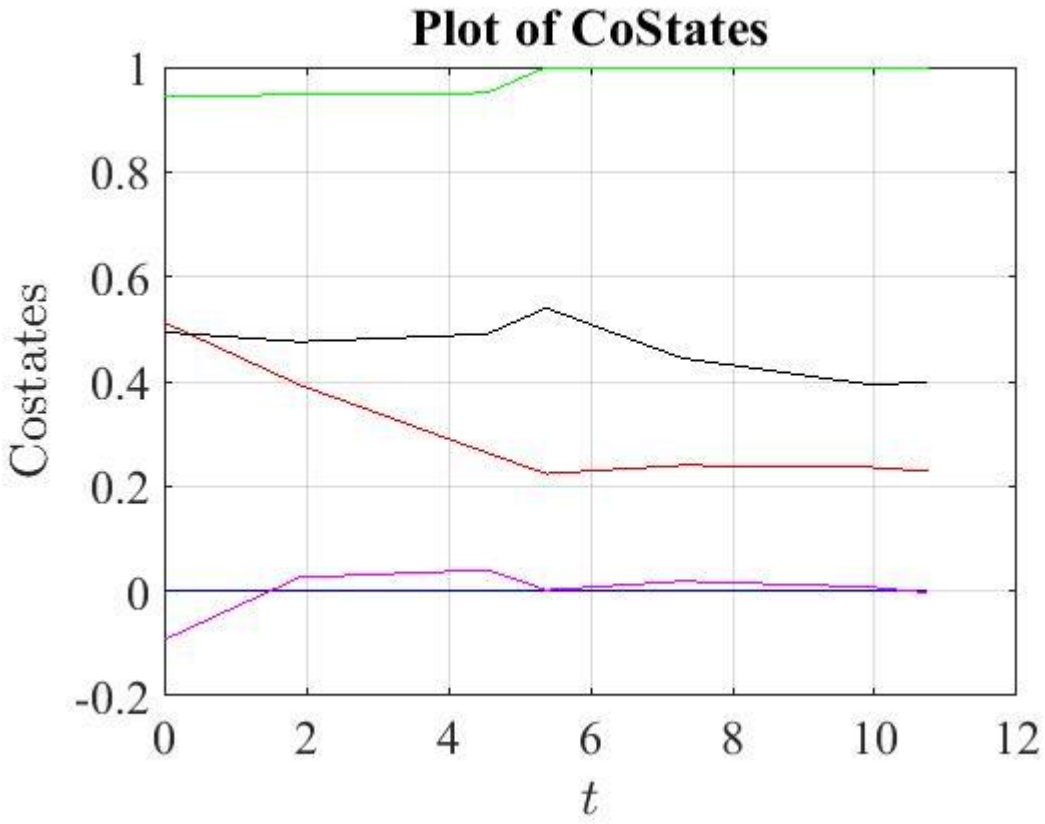
Objective – To maximize mass at final time (M_tf) using beta and T as control

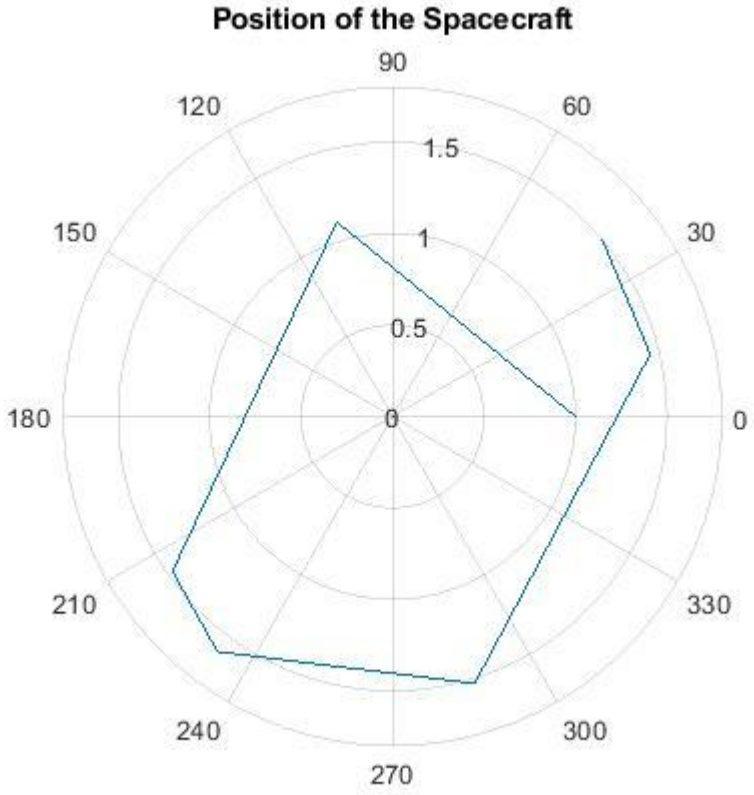
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 2





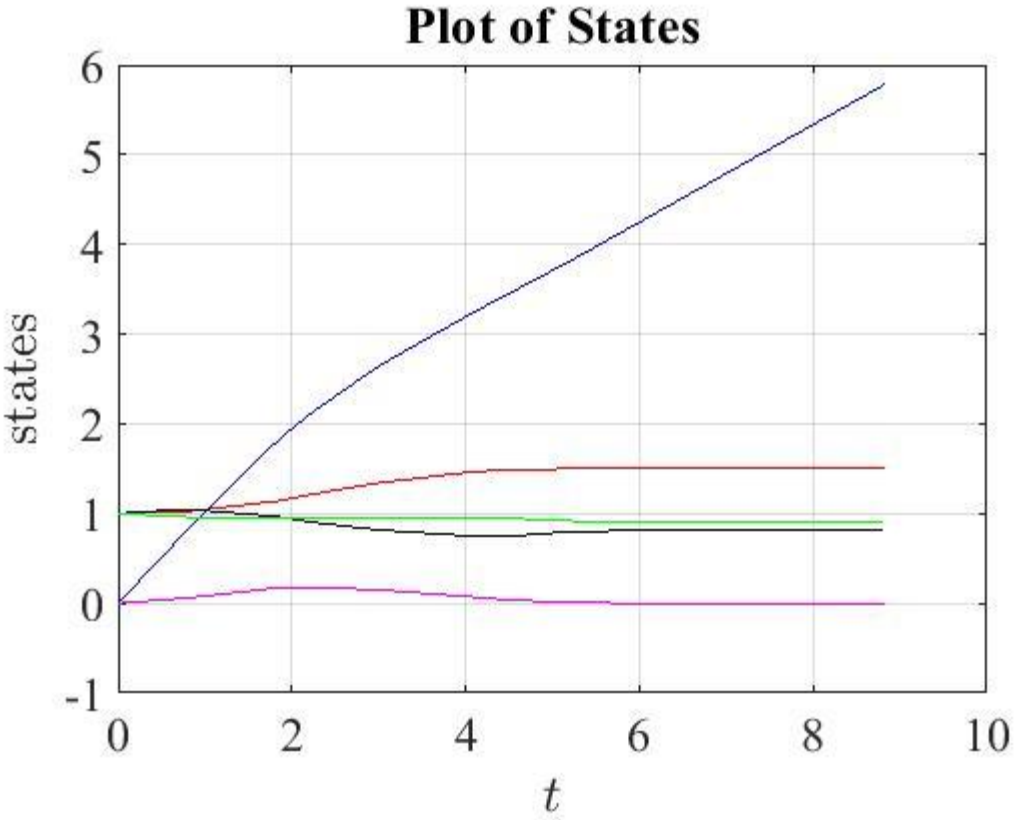


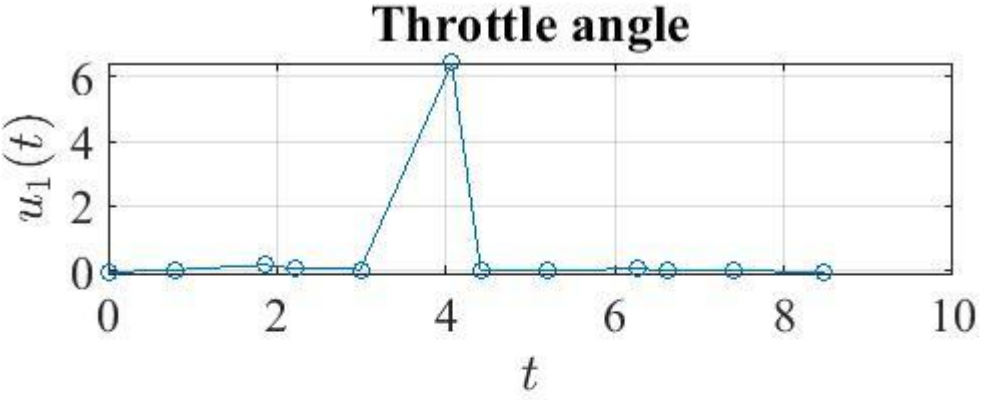


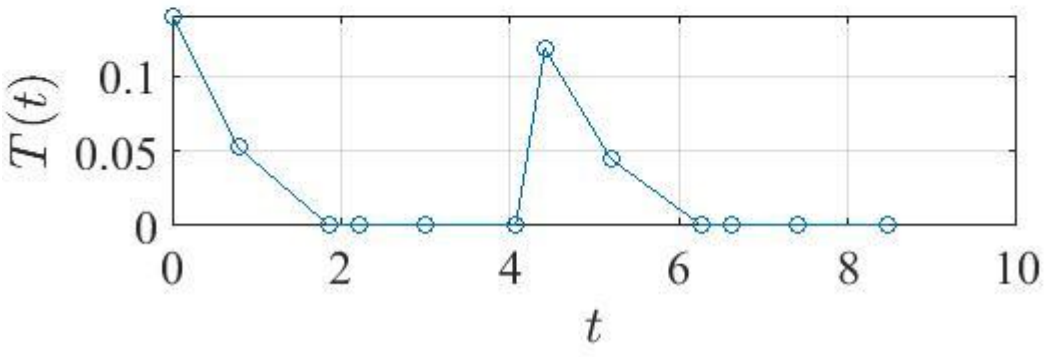


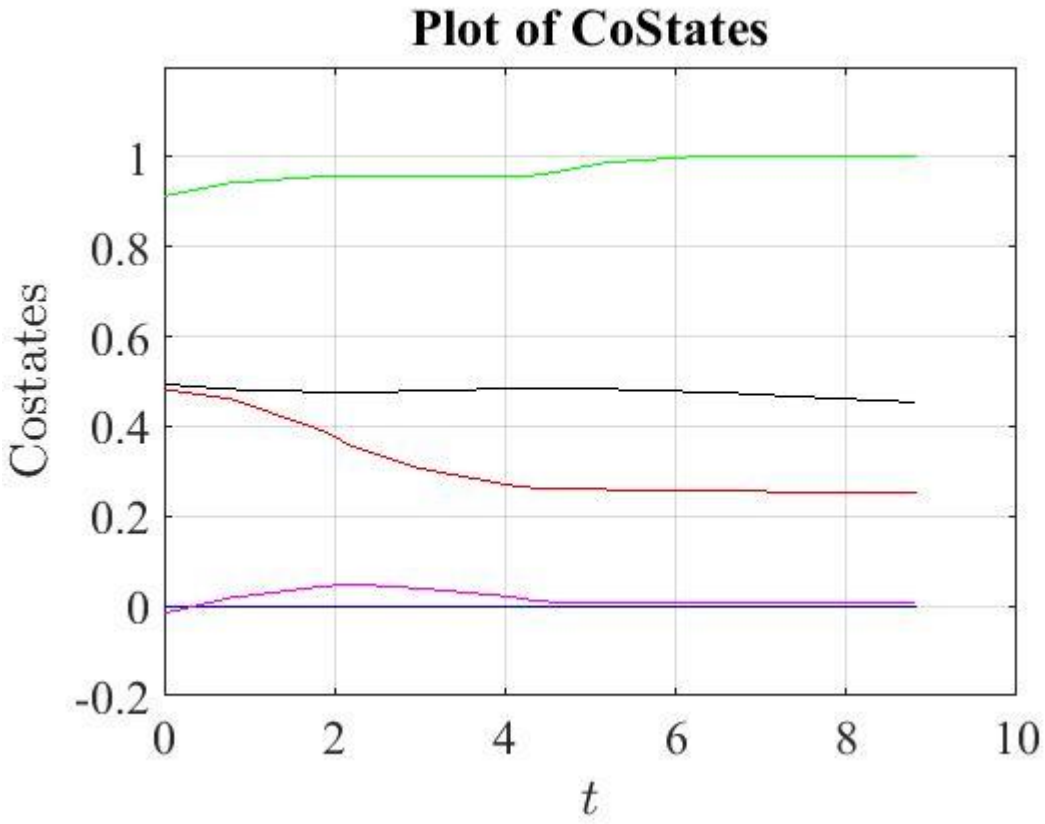
Published with MATLAB® R2021a

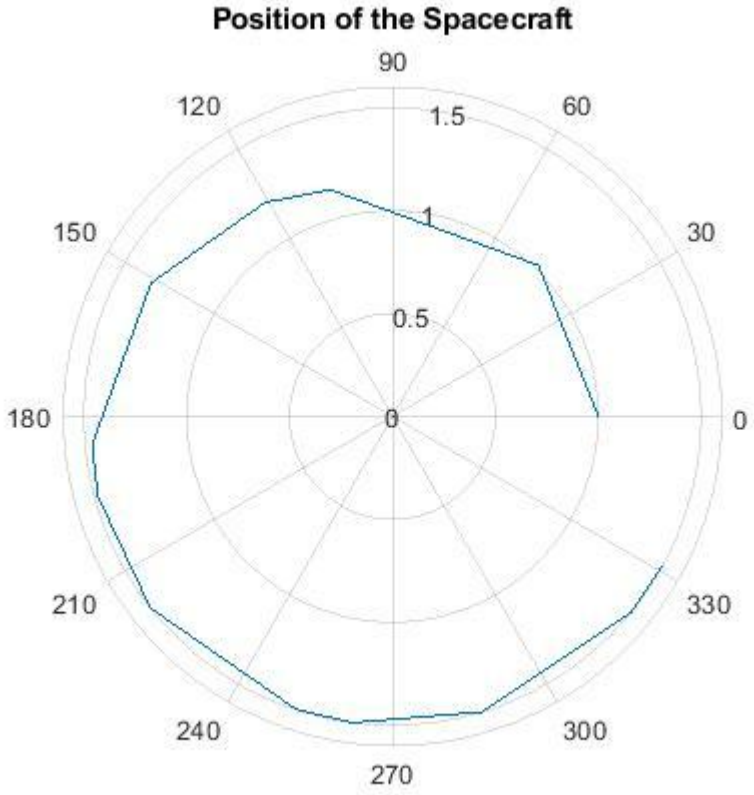
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 4





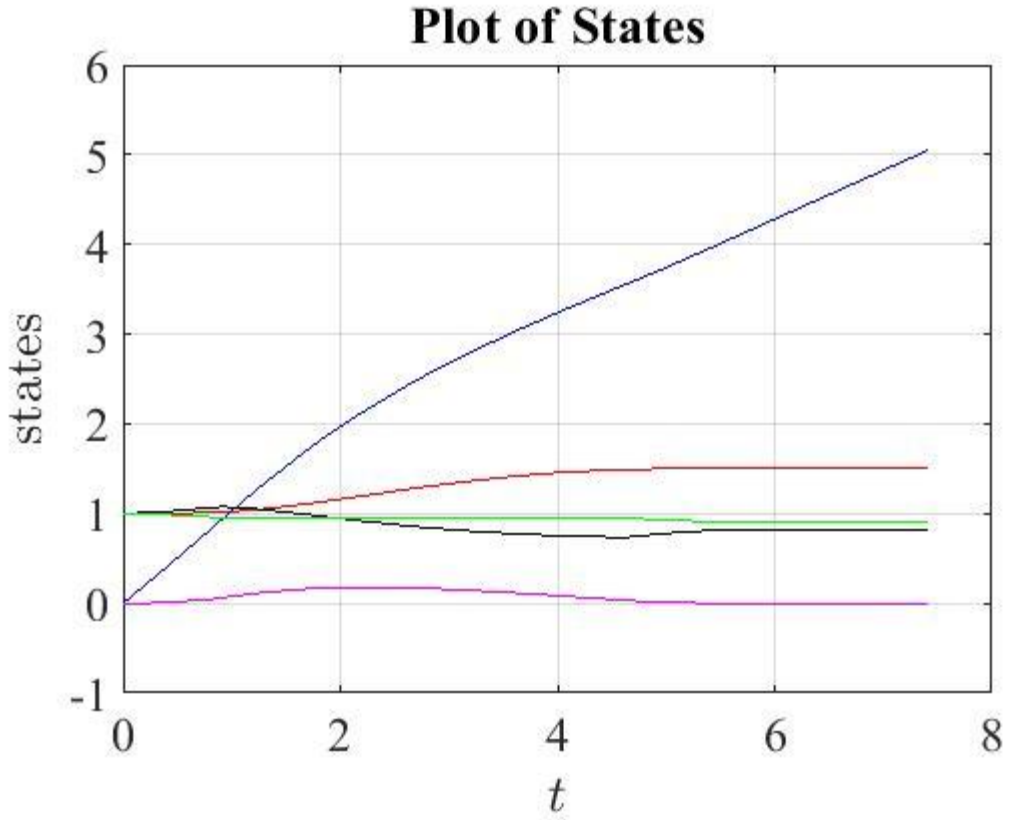


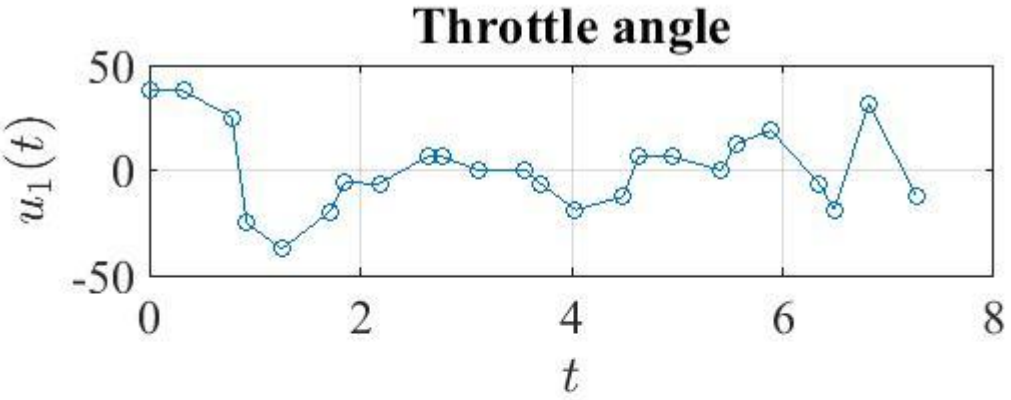


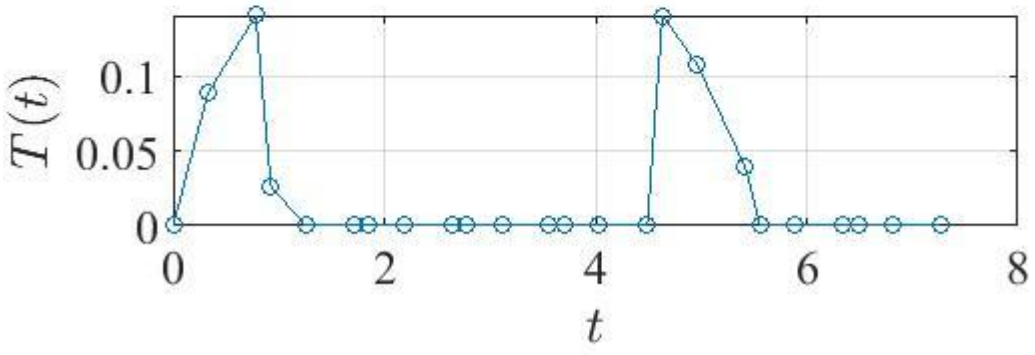


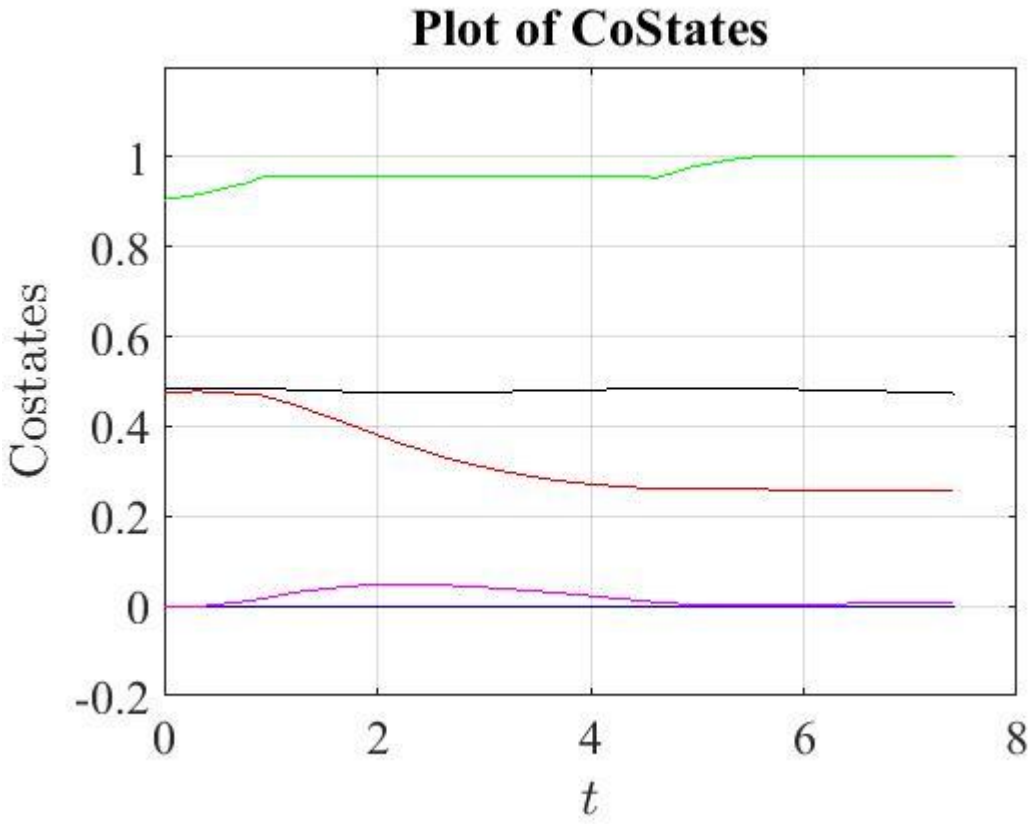
Published with MATLAB® R2021a

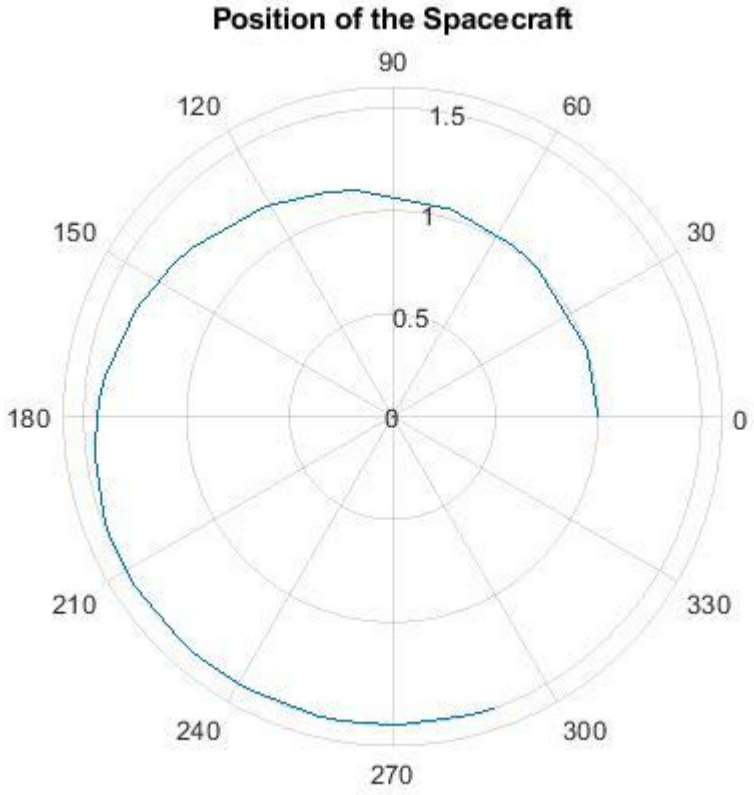
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 8





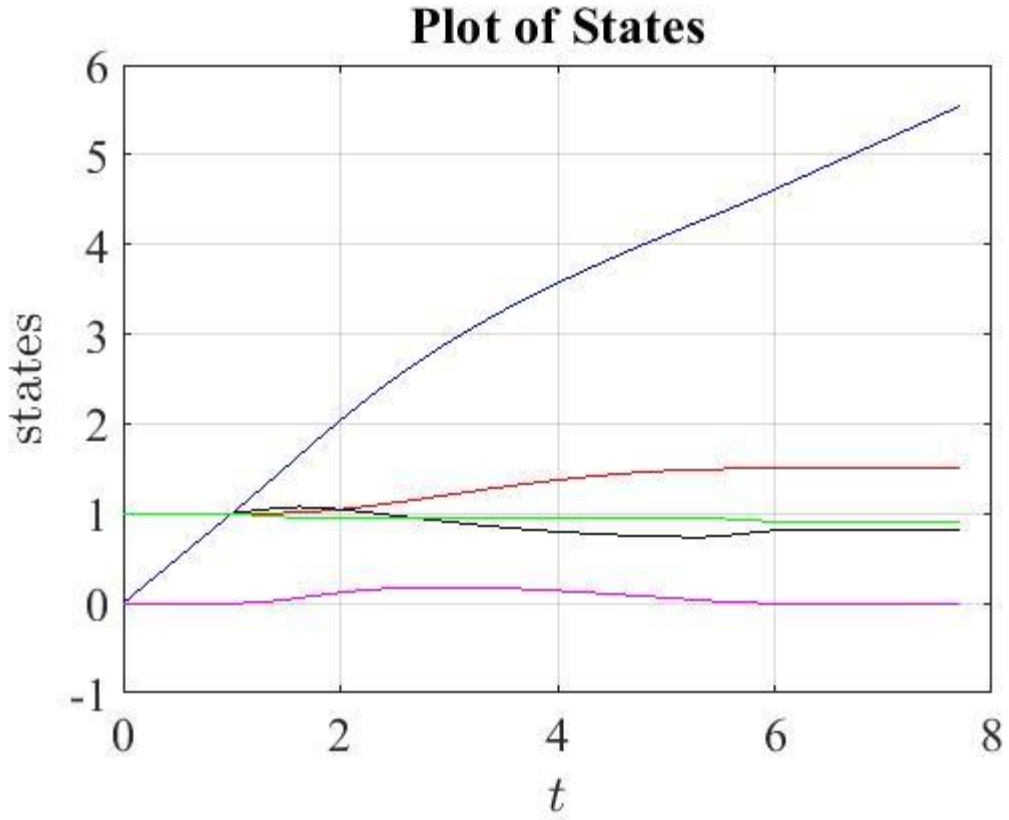


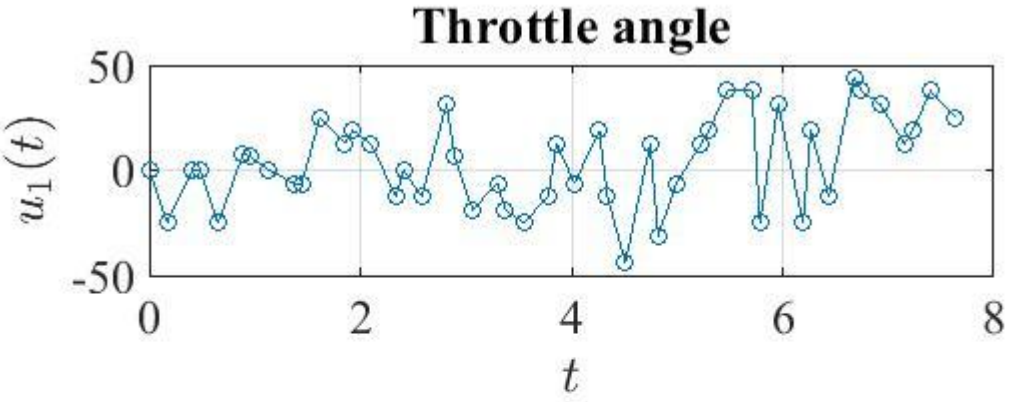


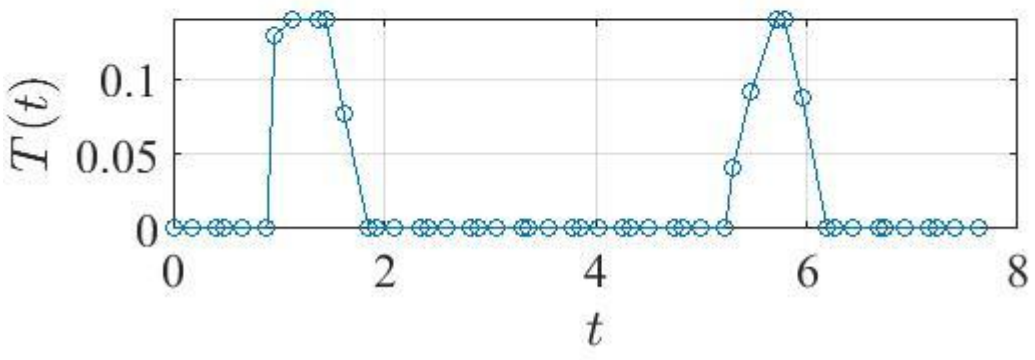


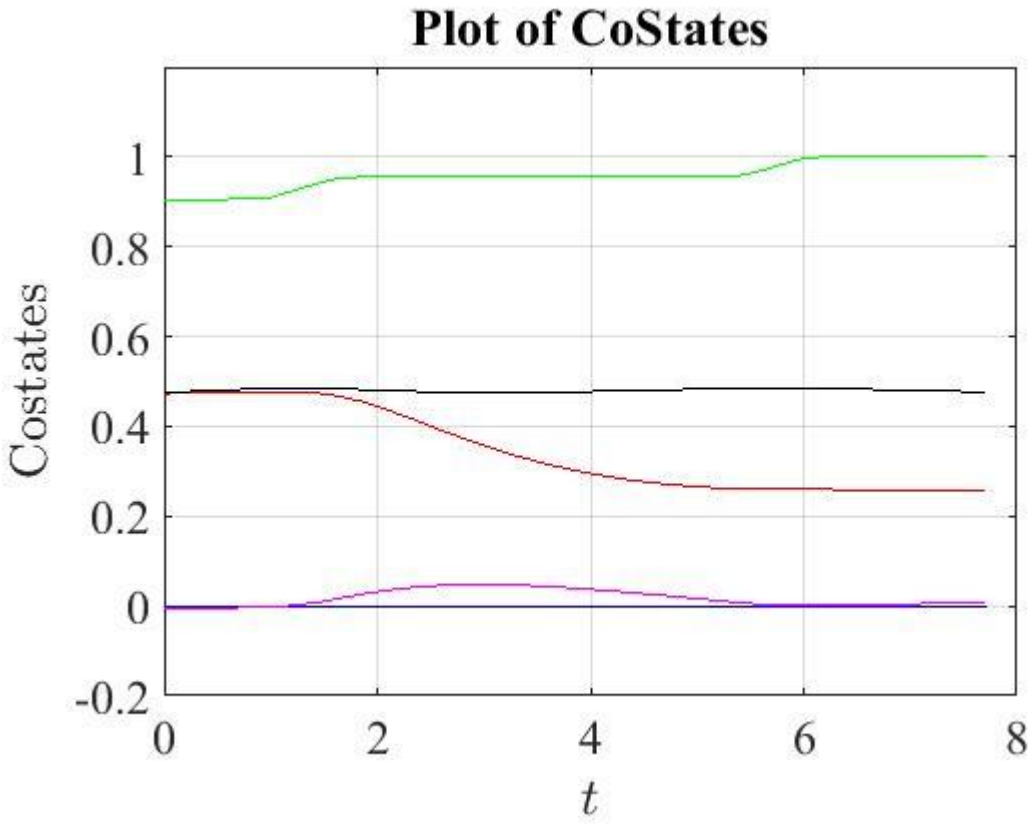
Published with MATLAB® R2021a

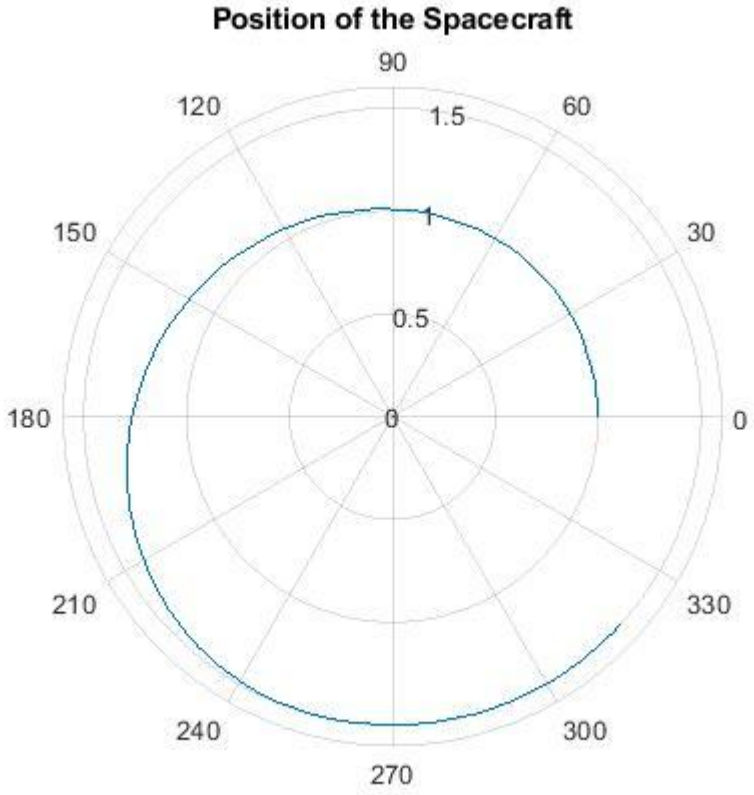
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 16





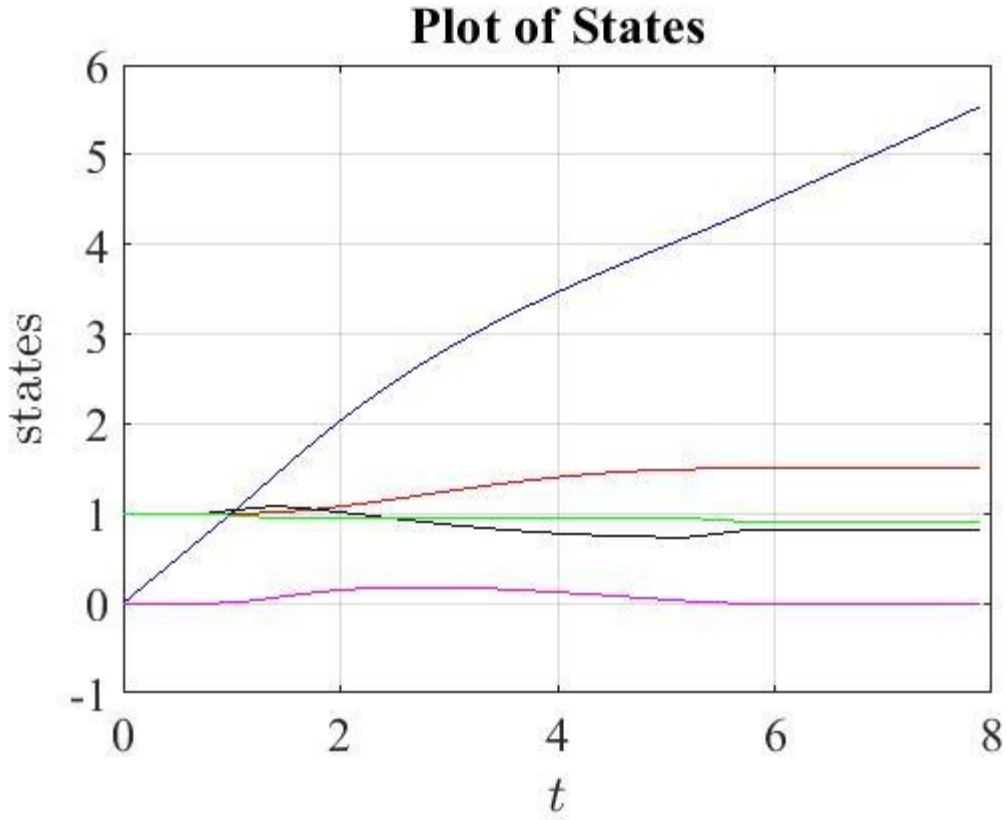


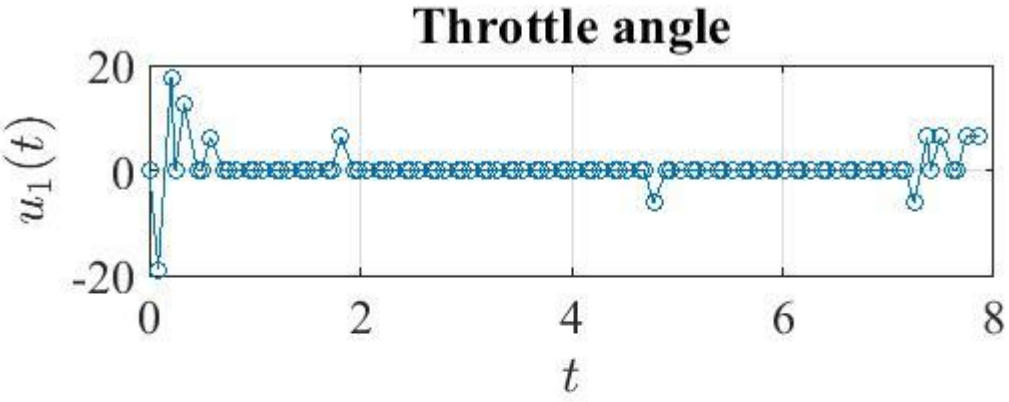


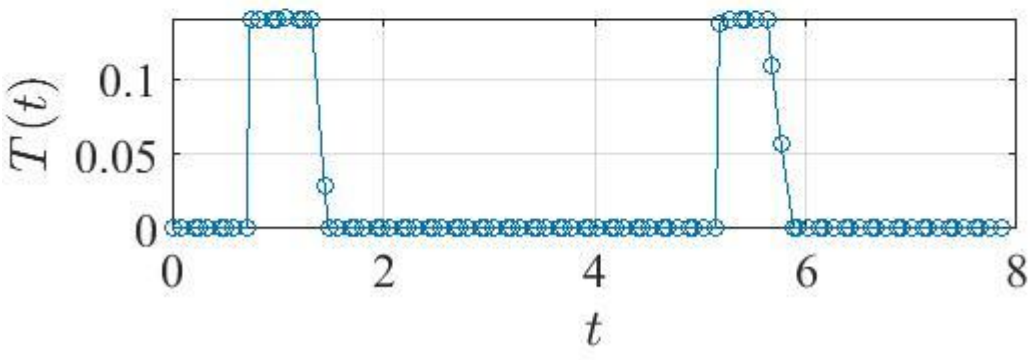


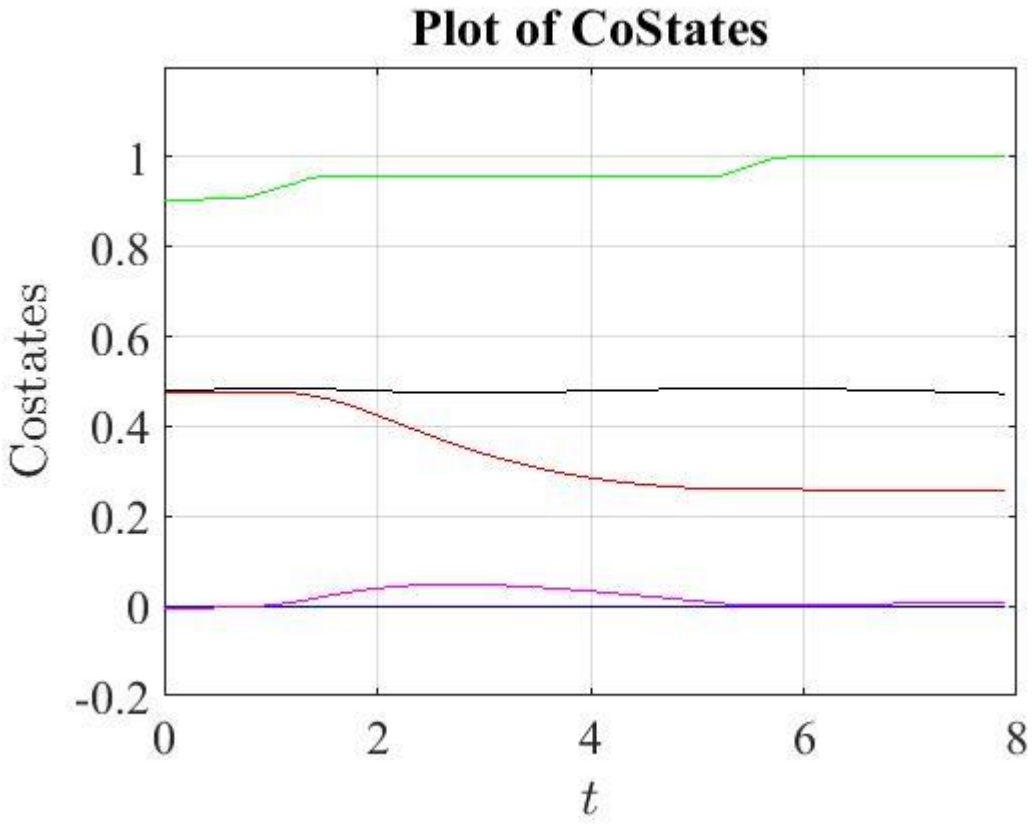
Published with MATLAB® R2021a

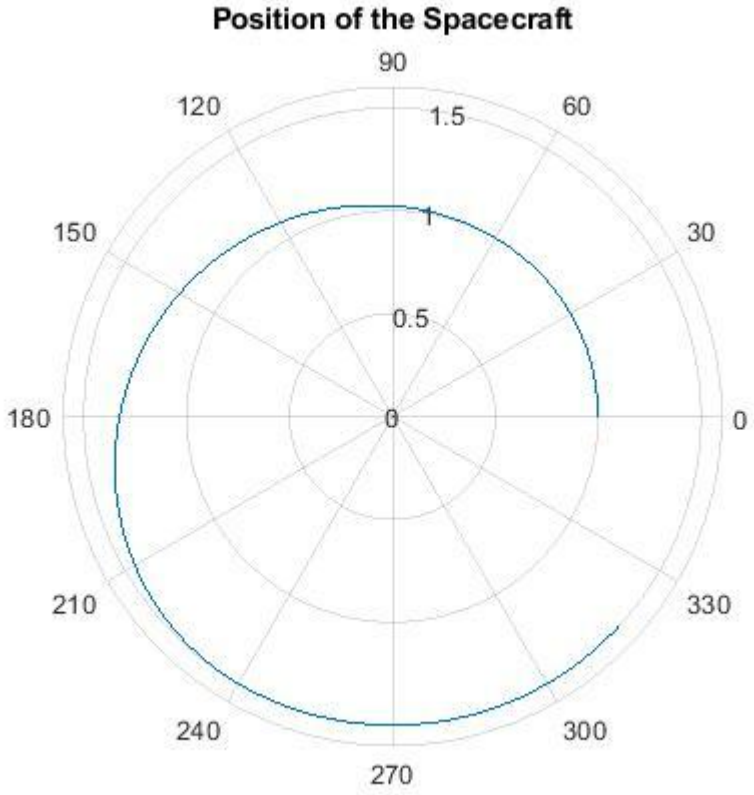
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 32





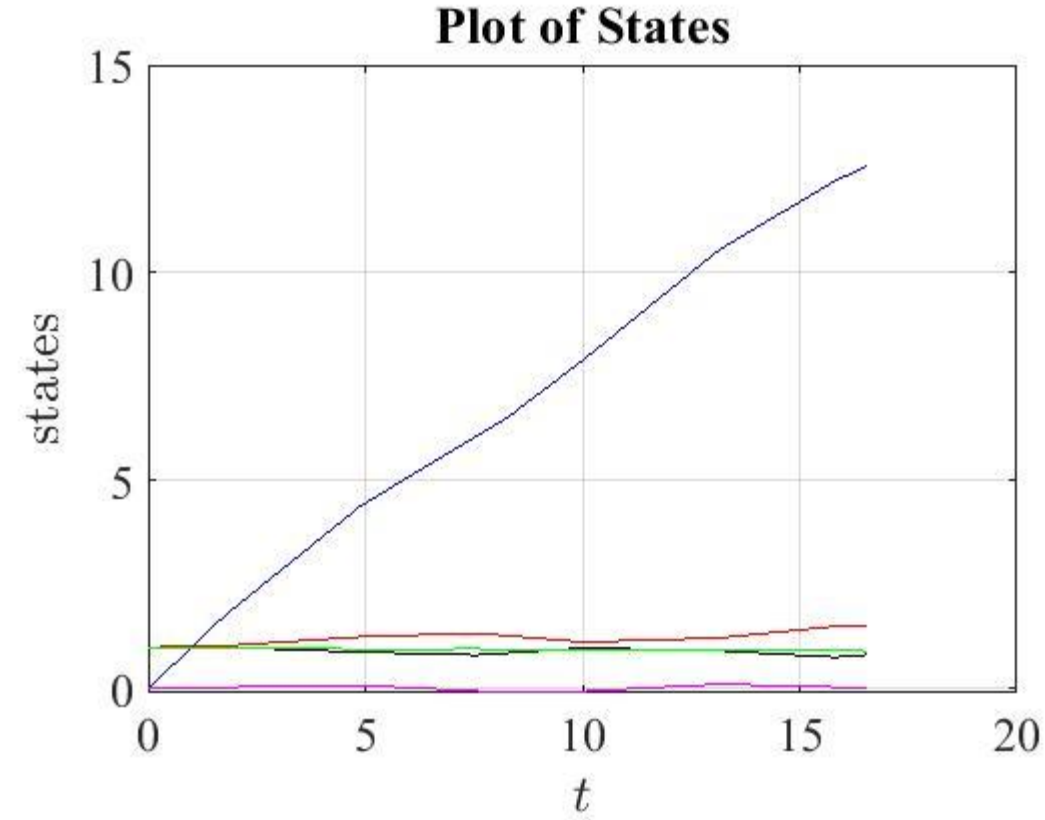


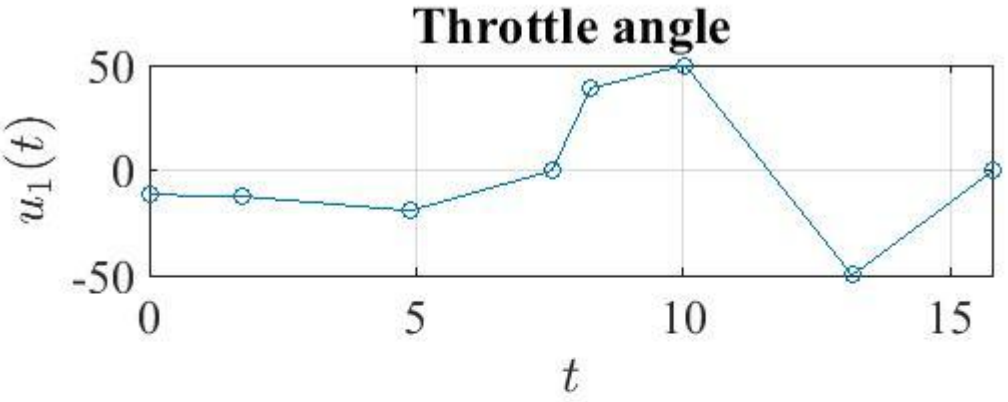


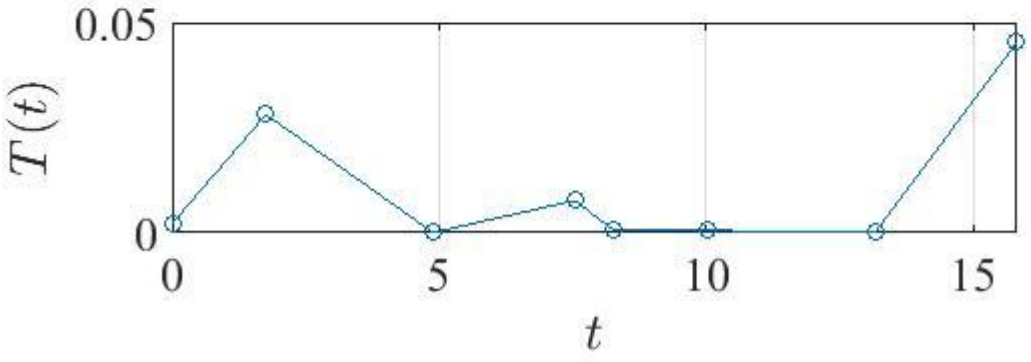


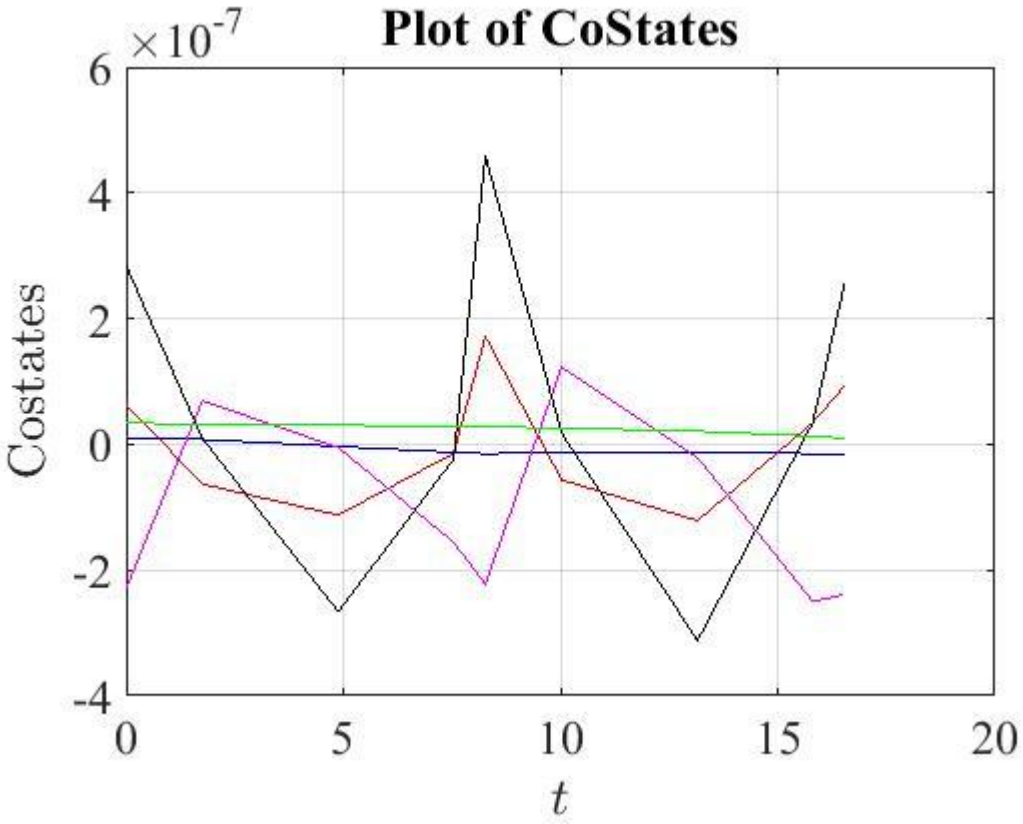
Published with MATLAB® R2021a

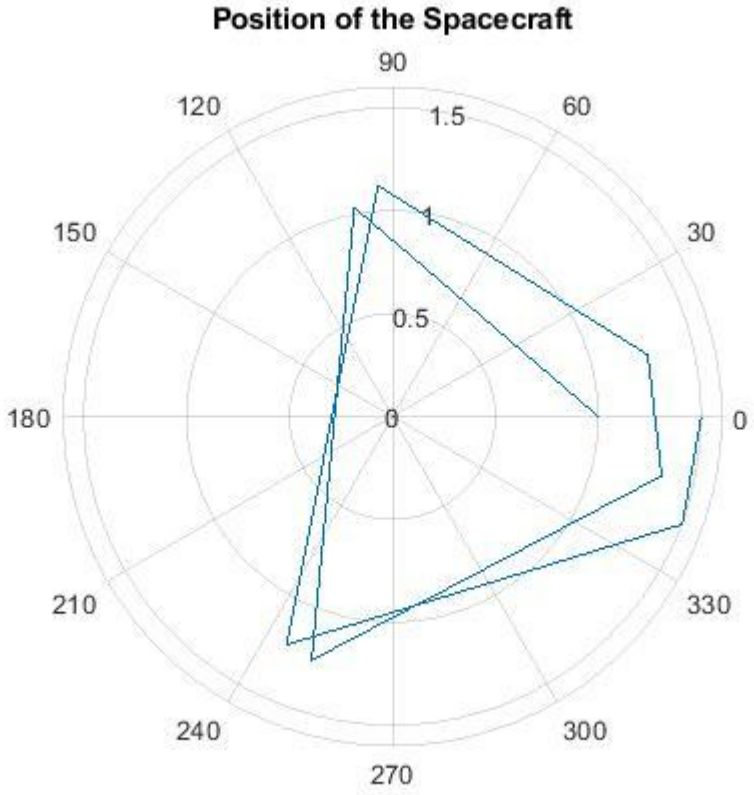
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 2





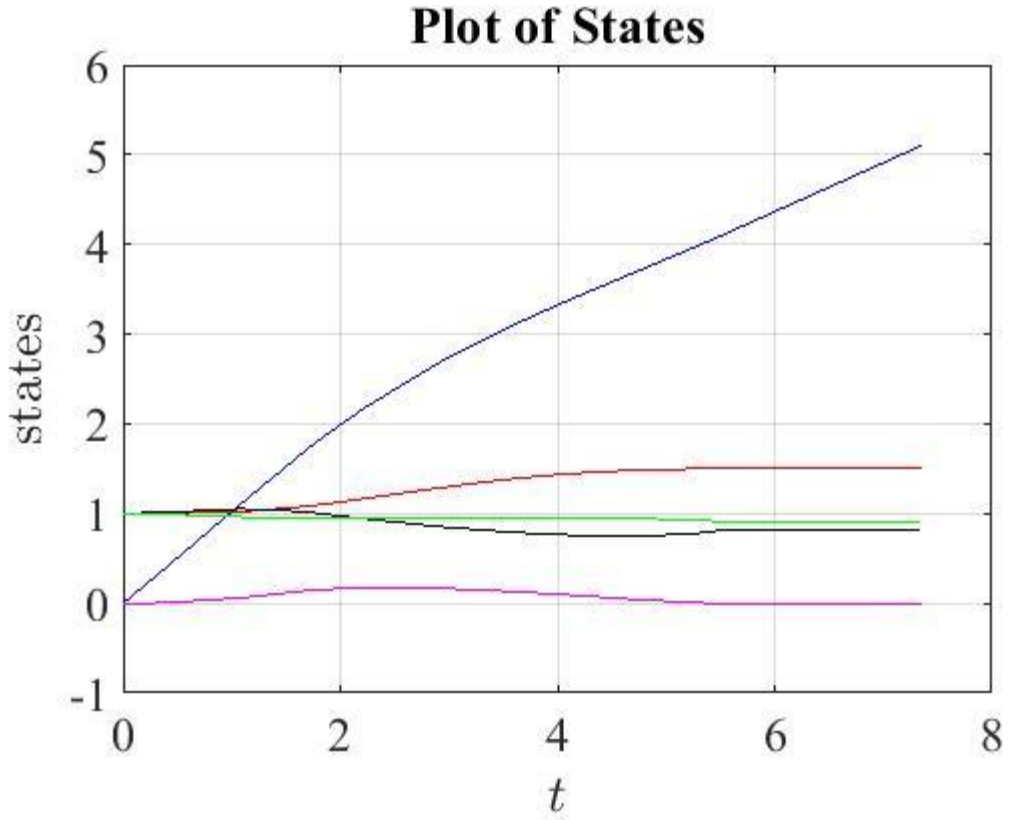


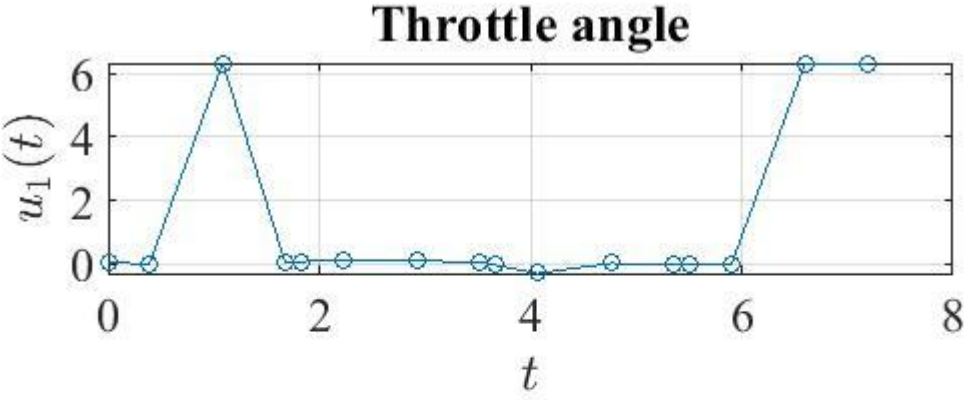


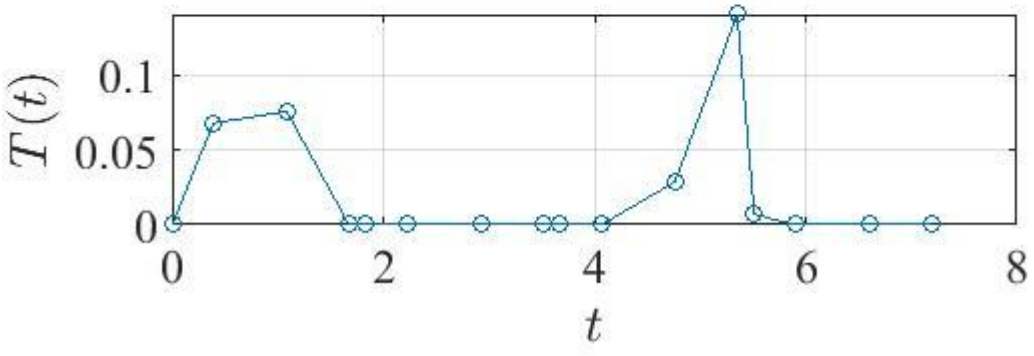


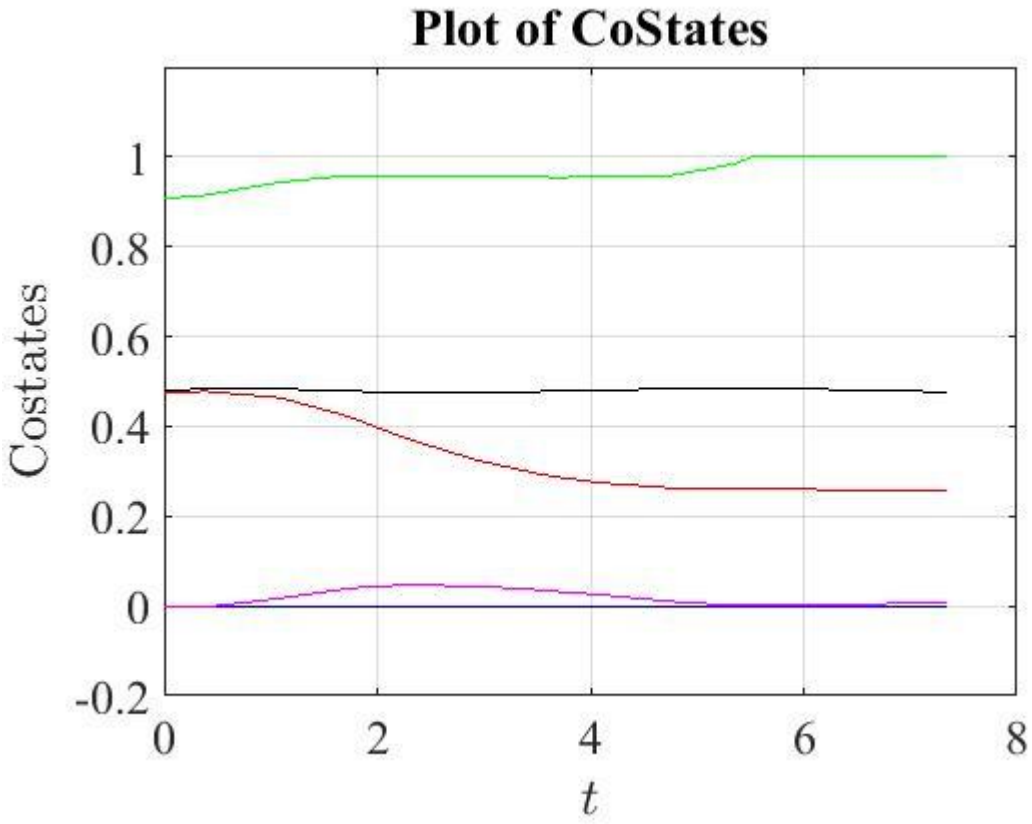
Published with MATLAB® R2021a

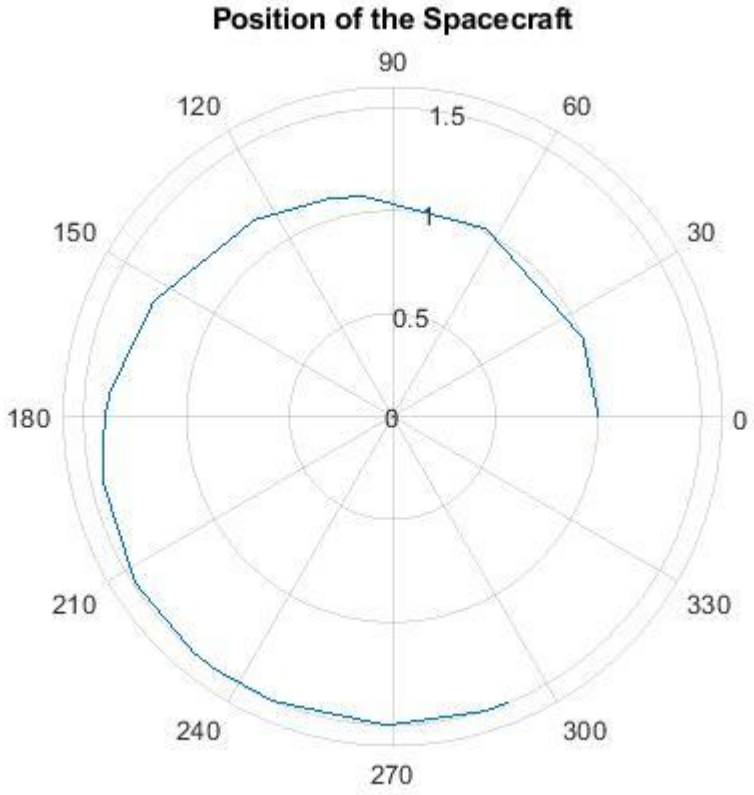
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 4





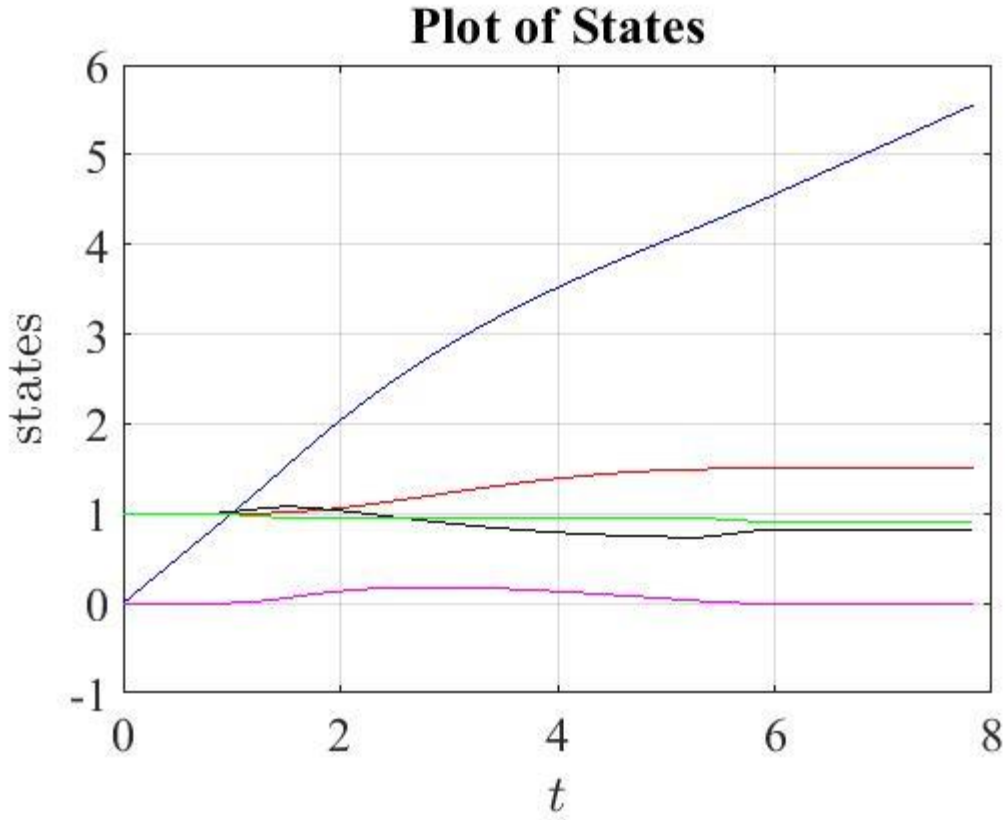


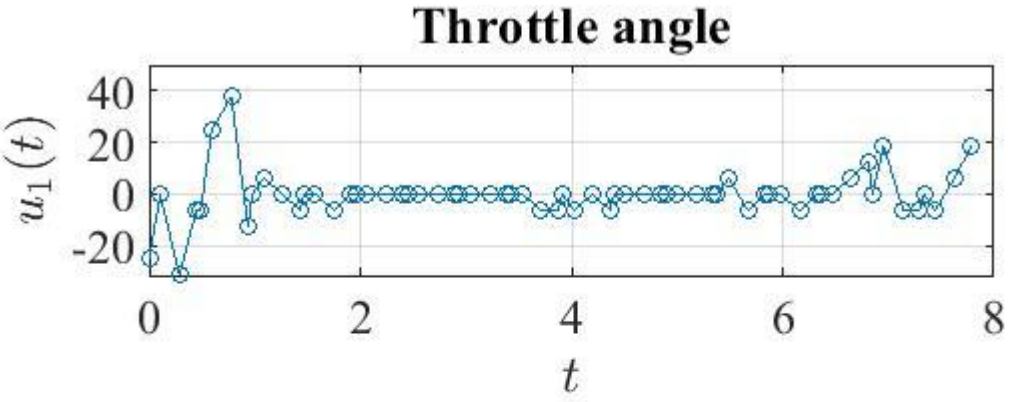


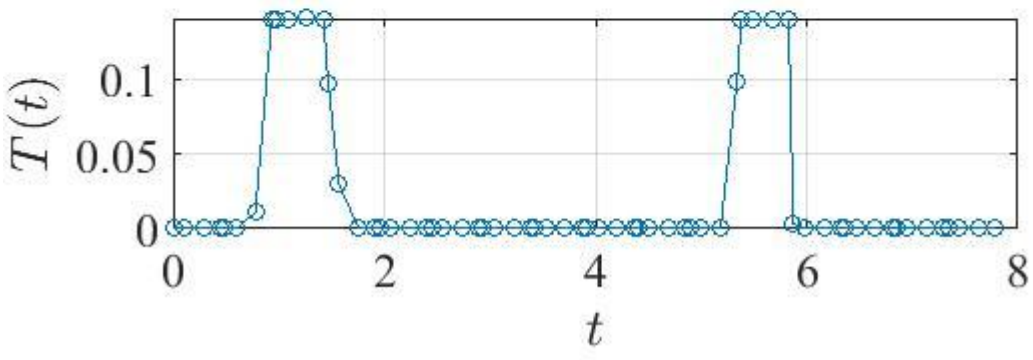


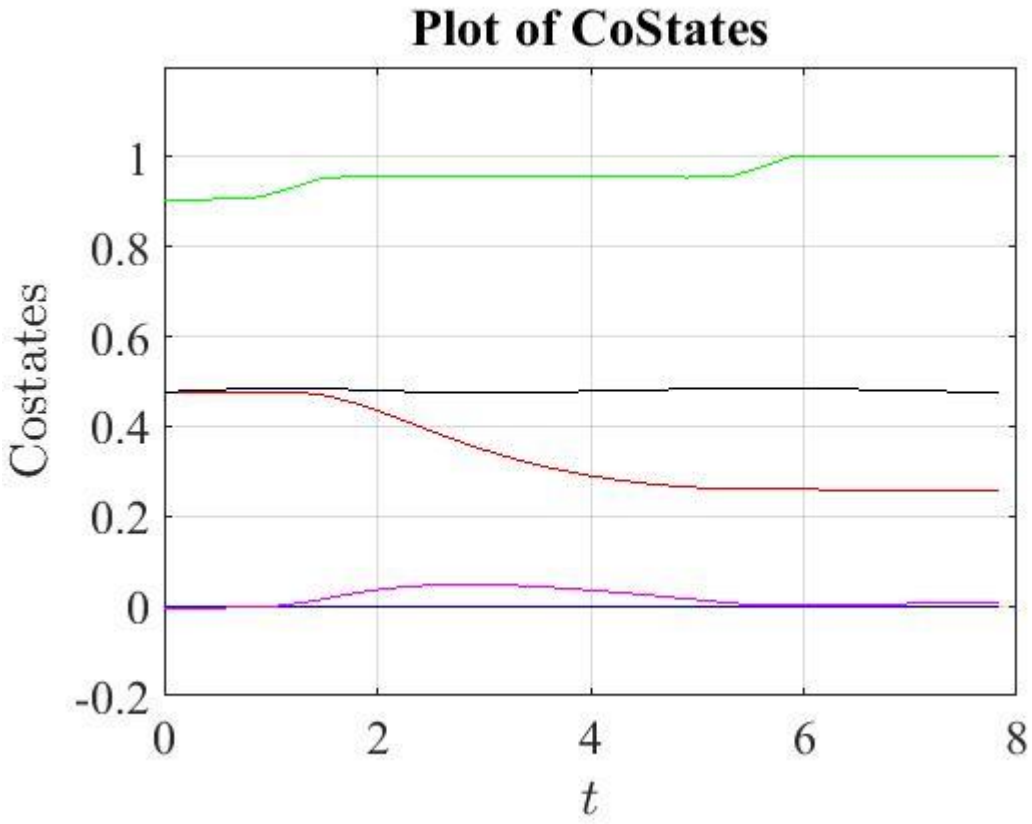
Published with MATLAB® R2021a

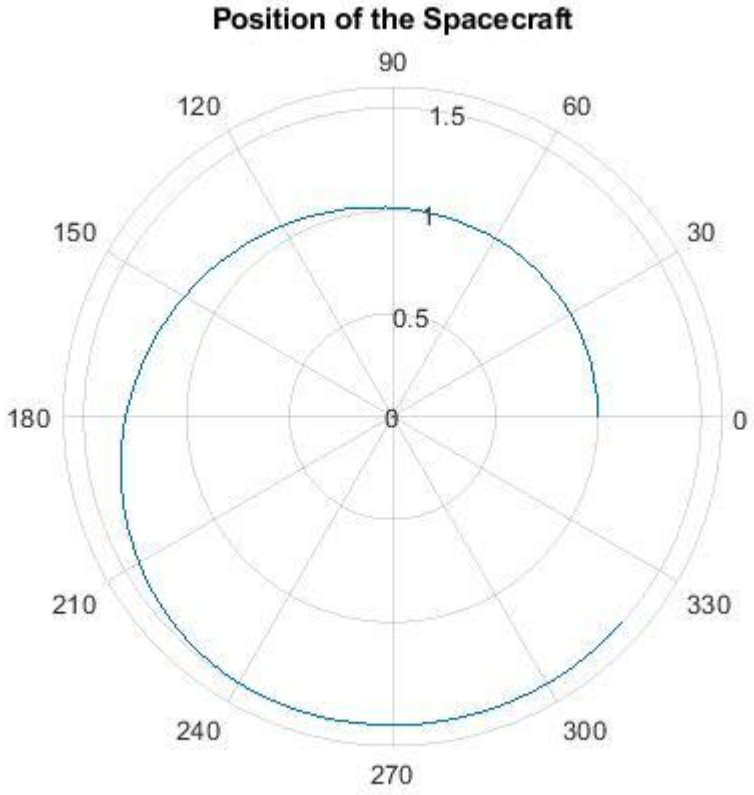
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 16





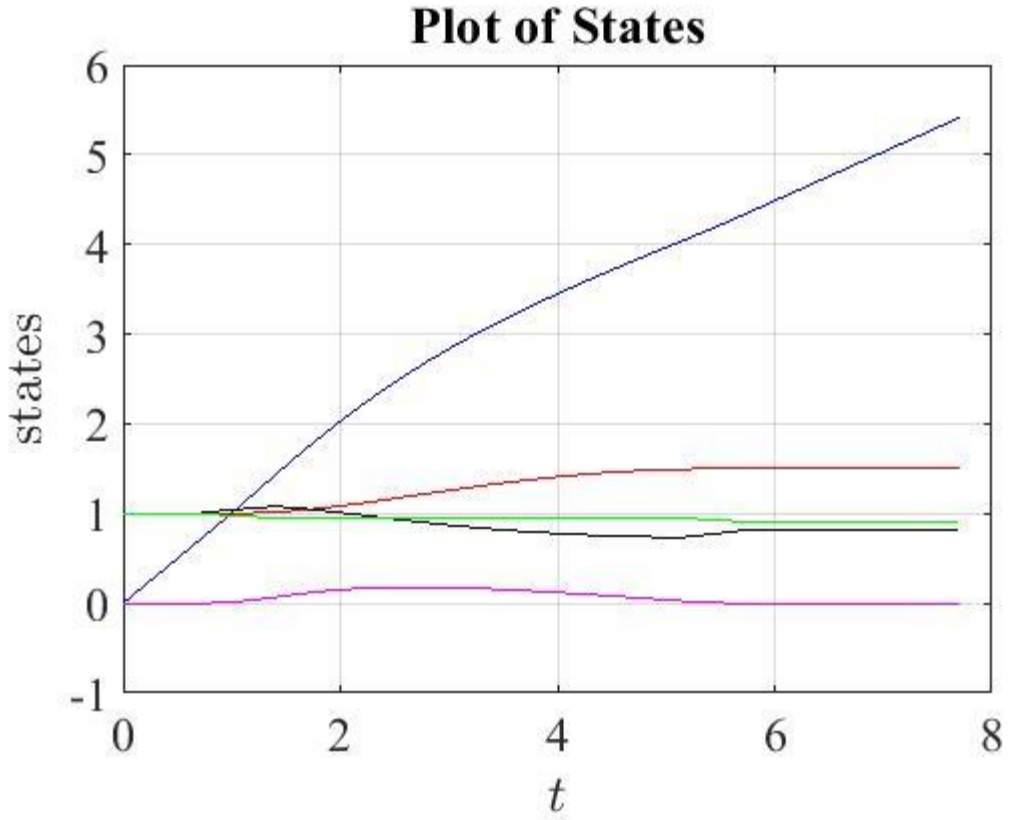


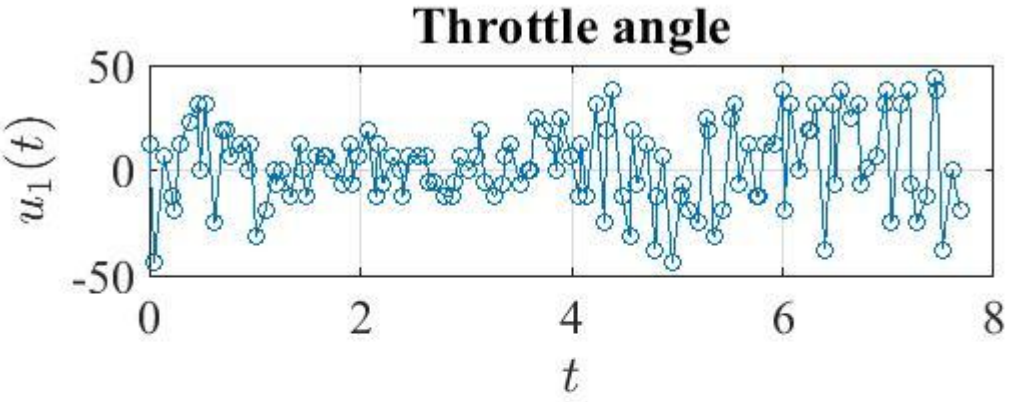


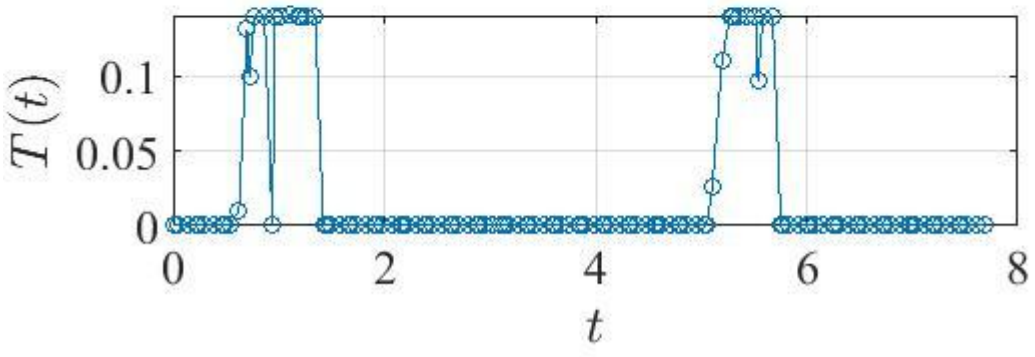


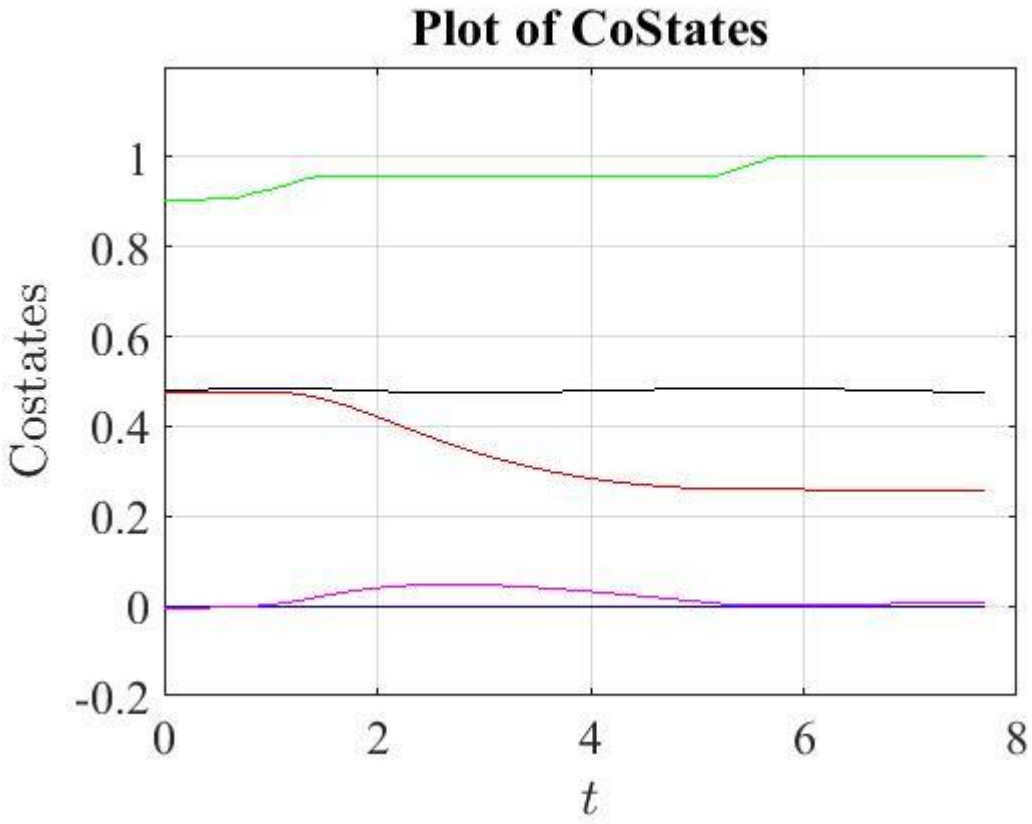
Published with MATLAB® R2021a

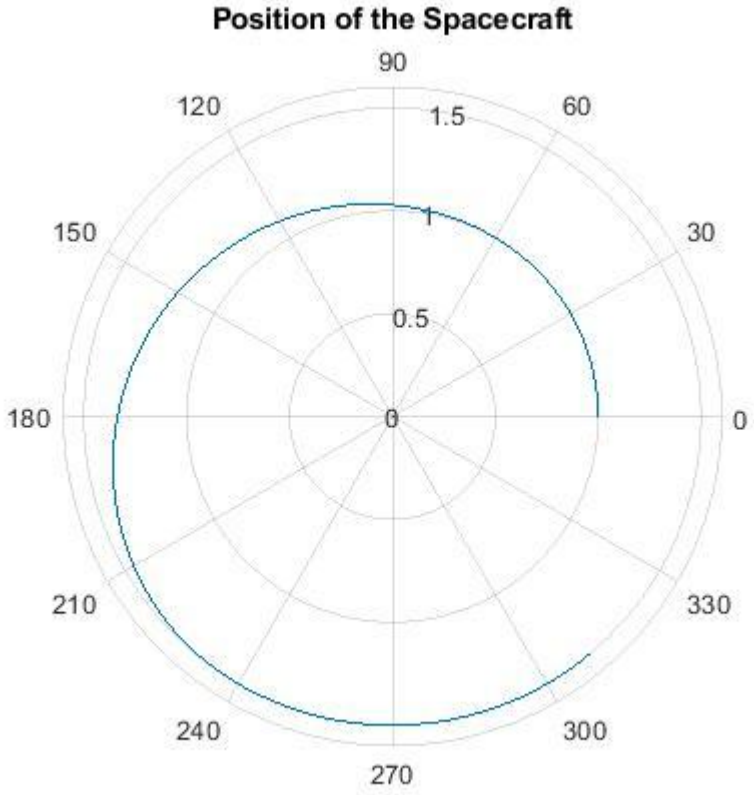
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 32







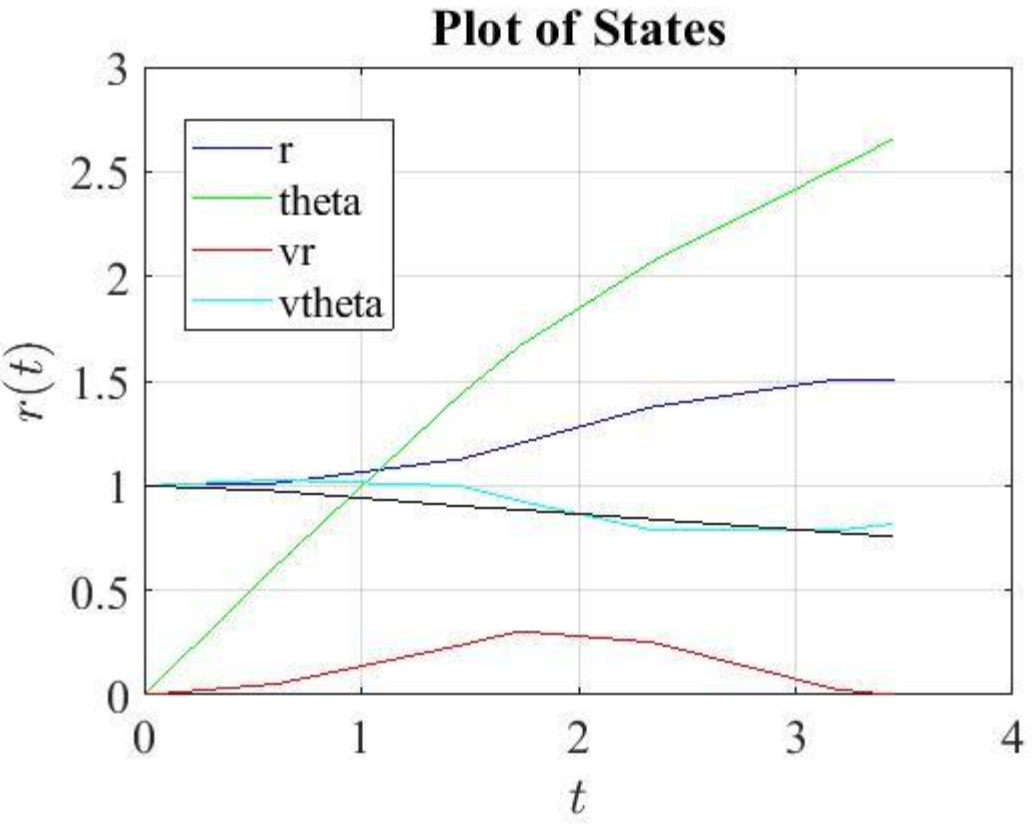


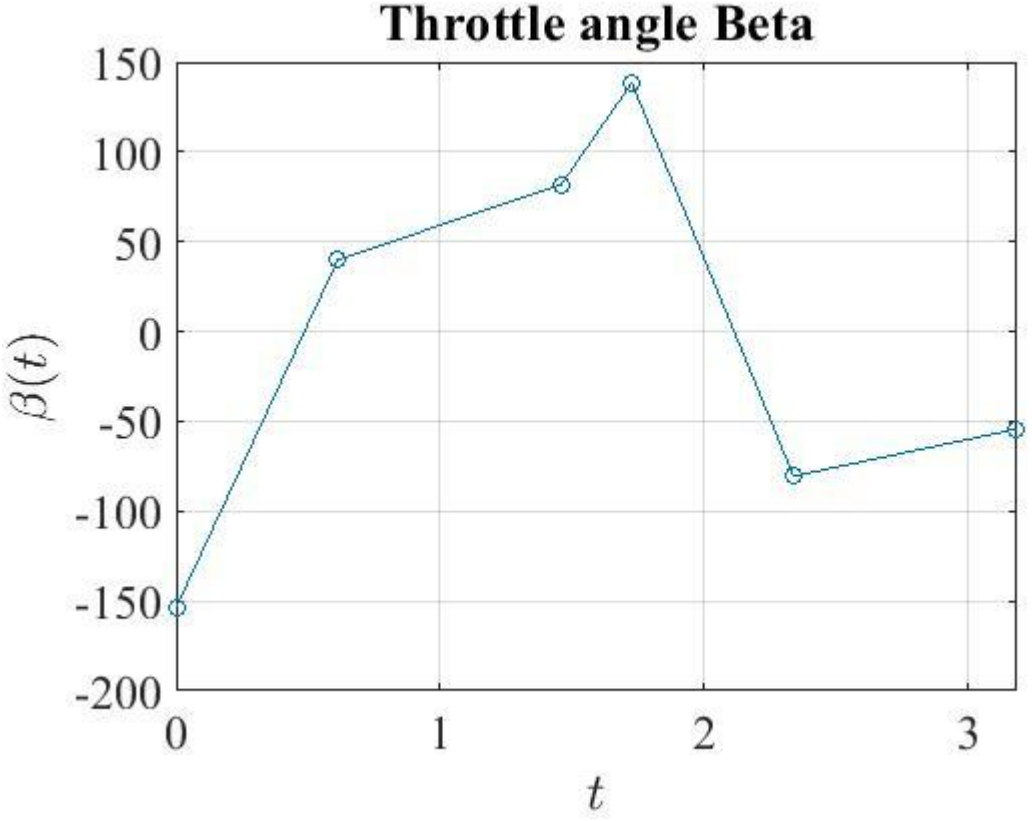


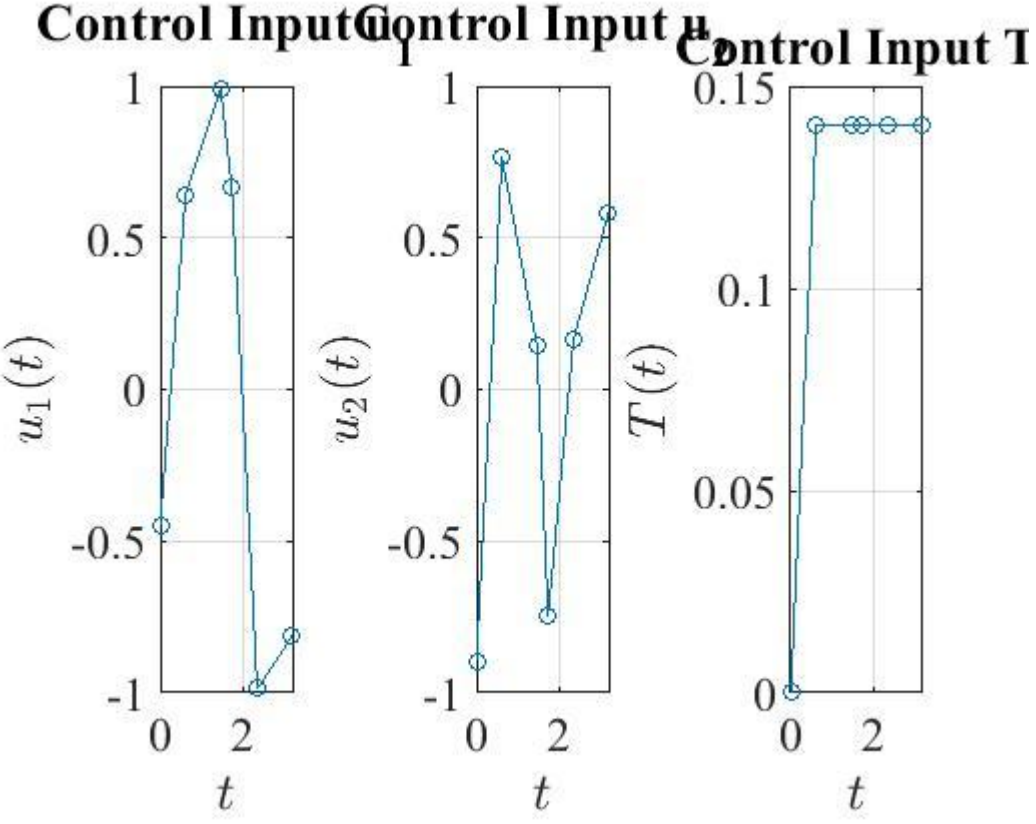
Published with MATLAB® R2021a

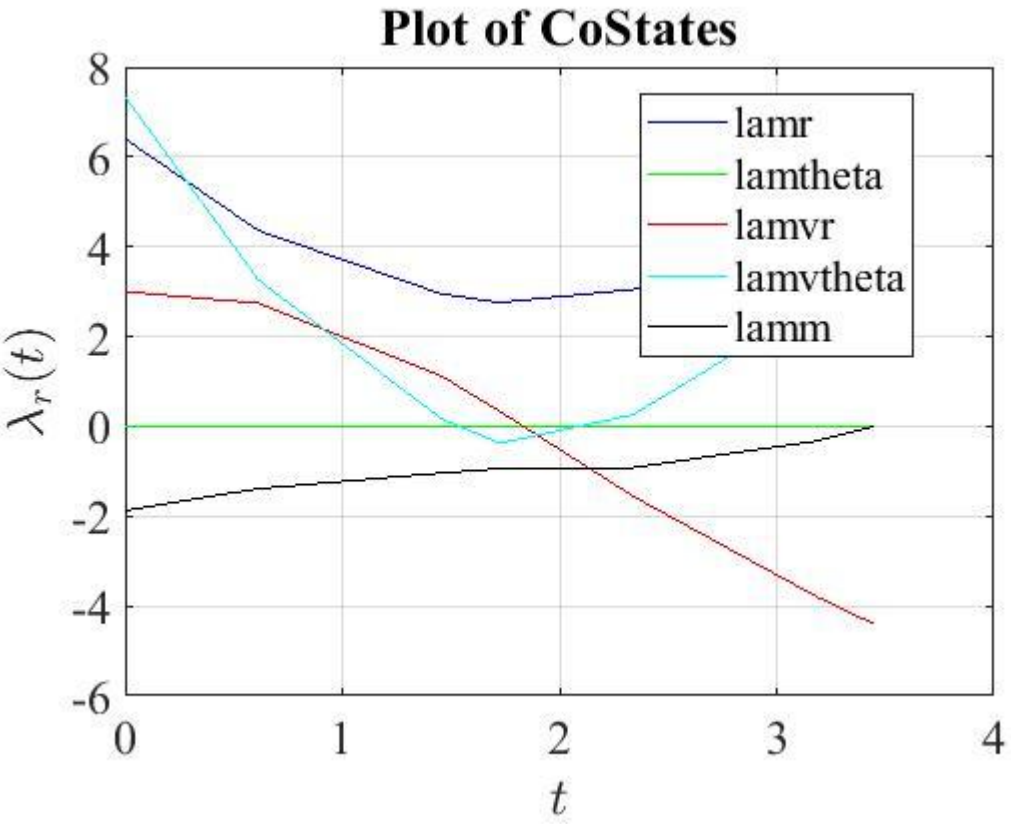
Objective – To minimize final time (M_tf) using U1, U2 and T as control.

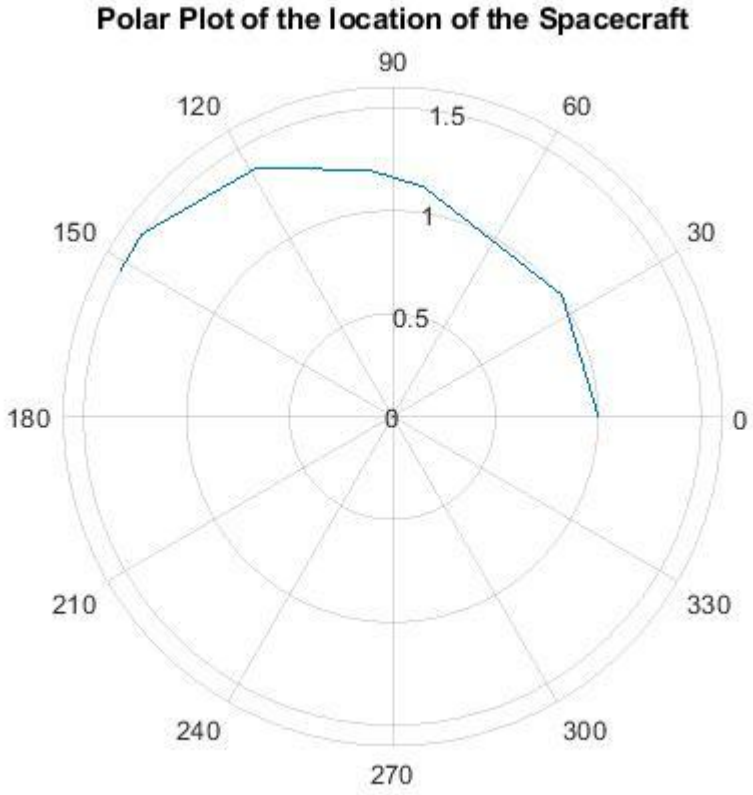
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 2




Throttle angle Beta
150
100
50
0
-50
-100
-150
-200
0
1
2
3
$\beta(t)$
t

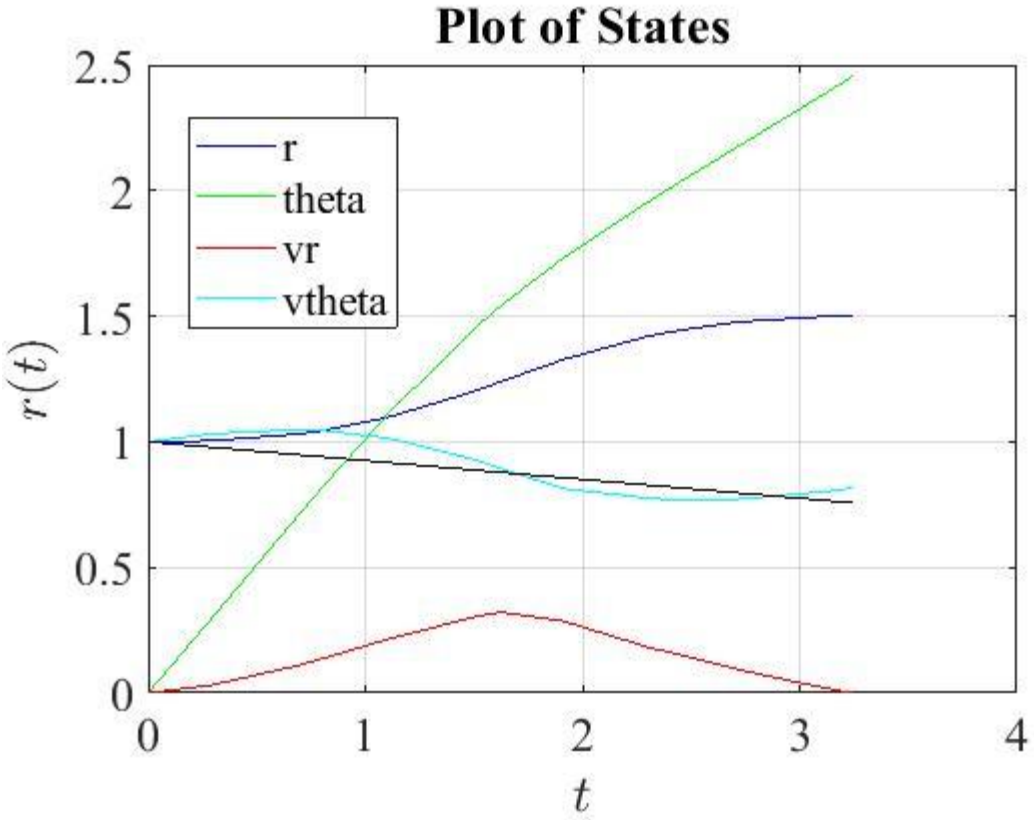

Control Input u1
Control Input u2
Control Input T
1
0.5
0
-0.5
-1
0
2
$u_1(t)$
$u_2(t)$
$T(t)$
0.15
0.1
0.05
0
t

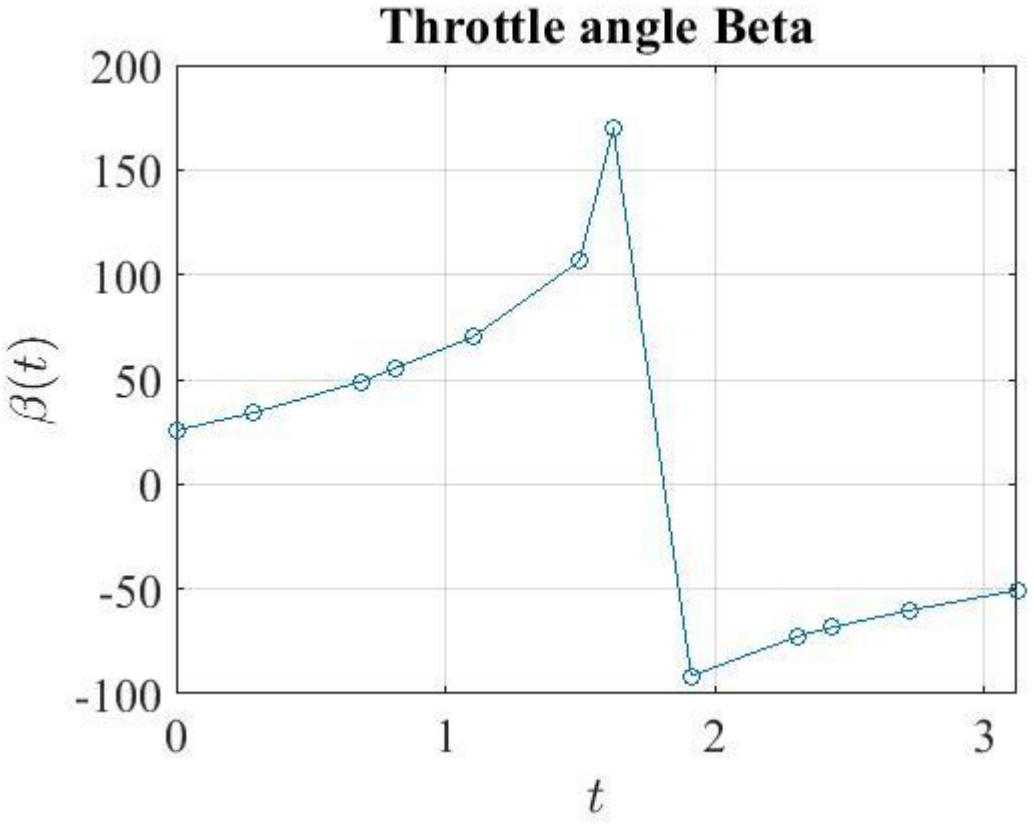


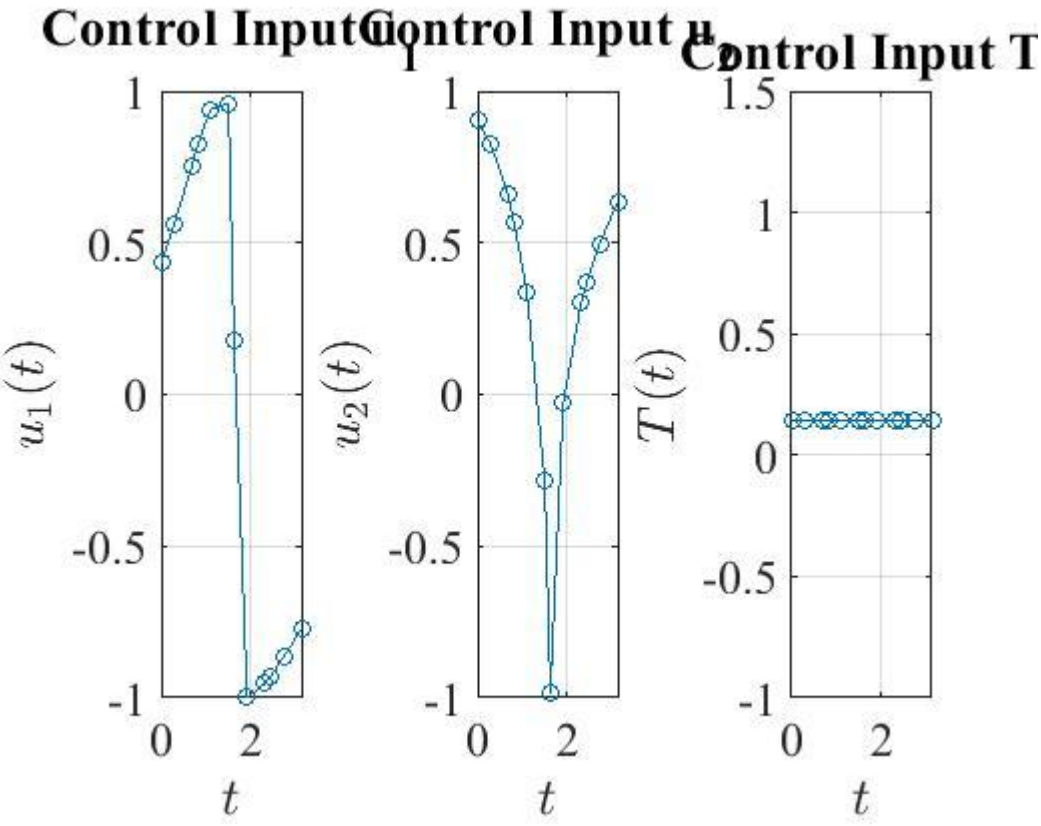


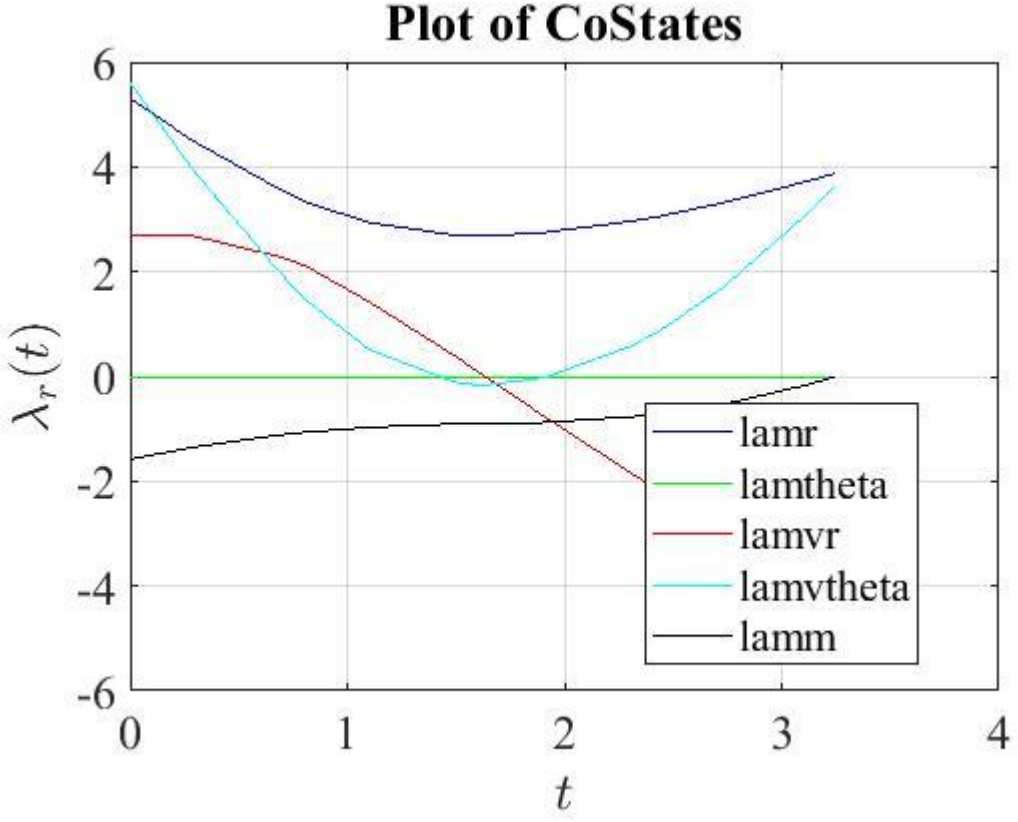
Published with MATLAB® R2021a

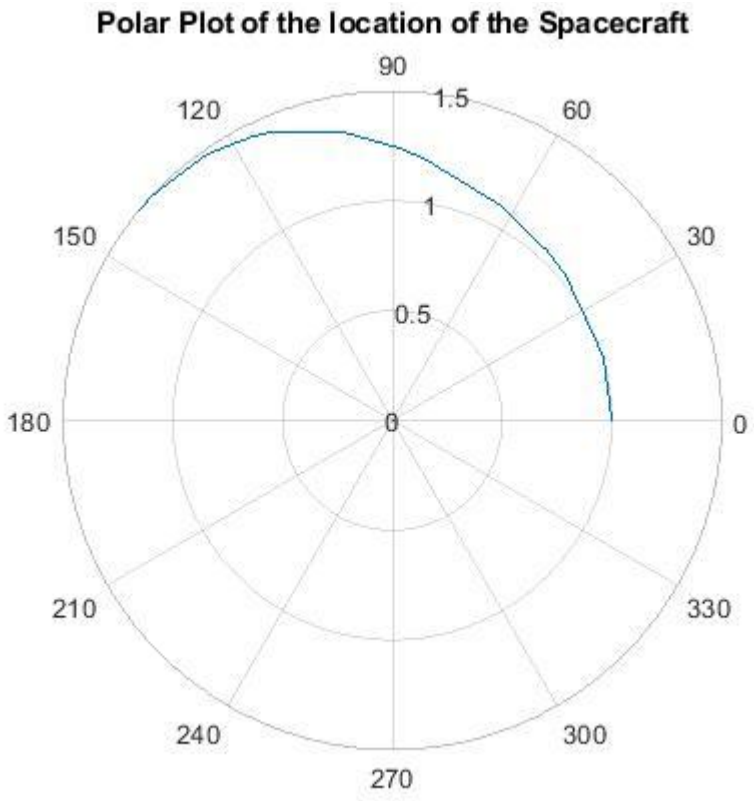
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 4





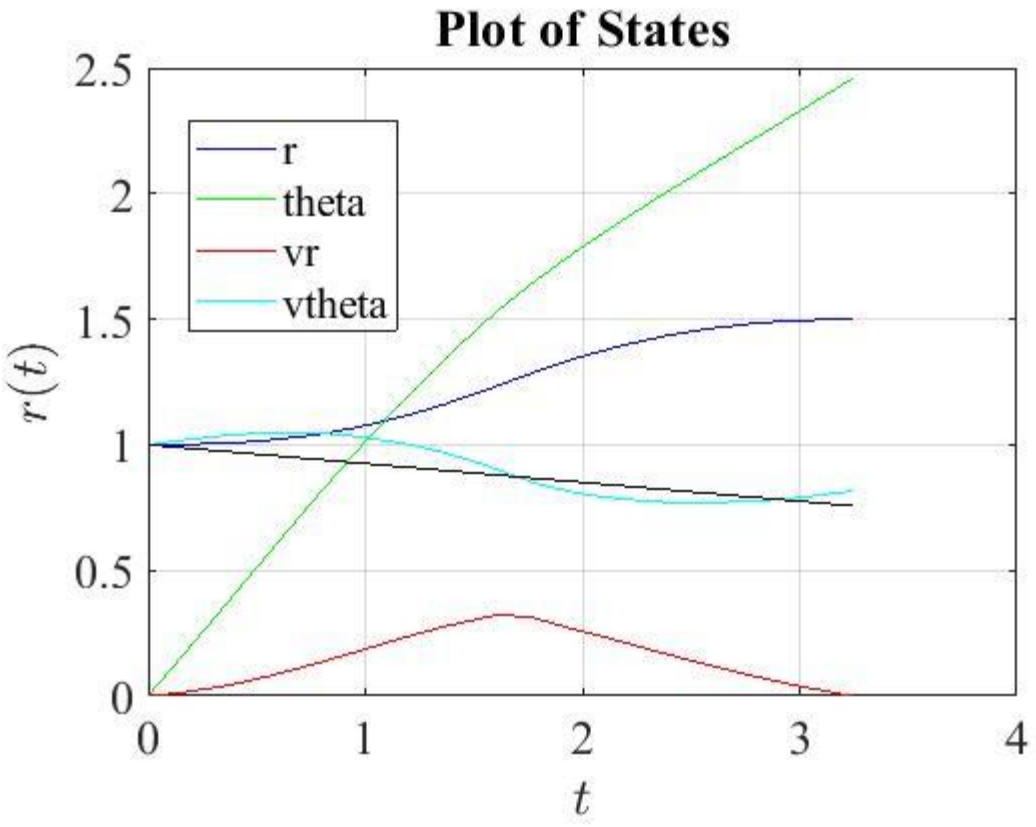

Control Input u1
Control Input u2
Control Input T
1
0.5
0
-0.5
-1
1.5
0
2
t
u1(t)
u2(t)
T(t)

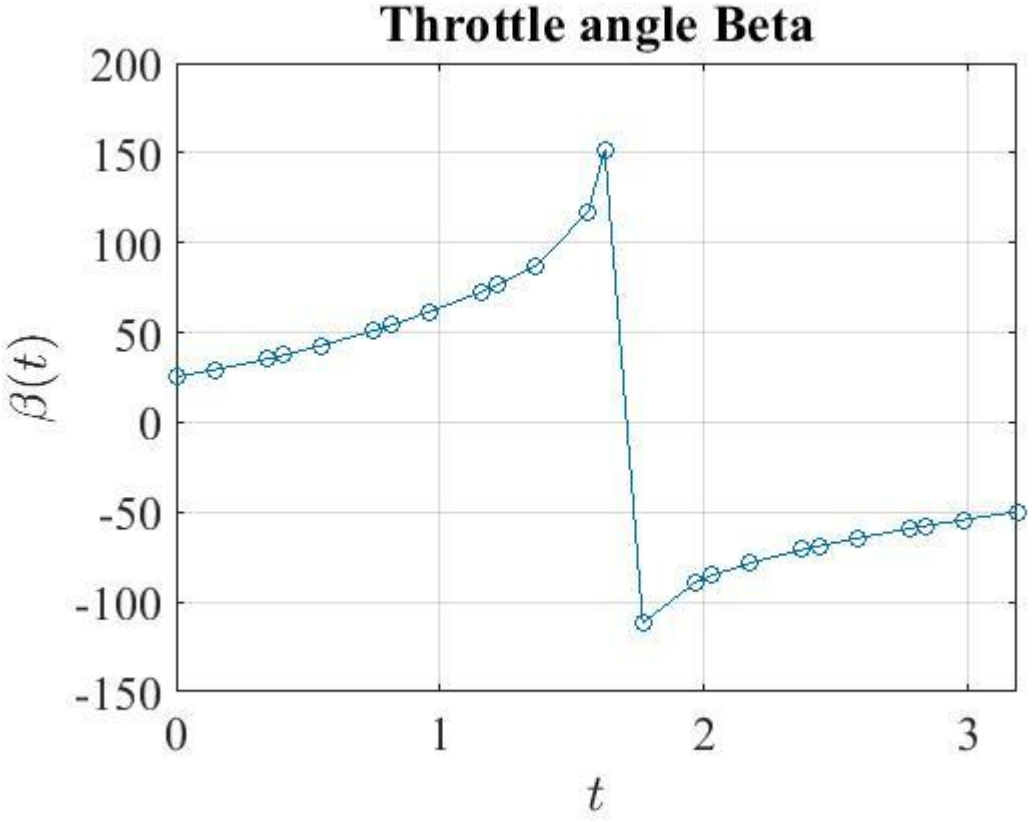

Plot of CoStates
6
4
2
0
-2
-4
-6
1
3
4
t
λr(t)
lamr
lamtheta
lamvr
lamvtheta
lamm

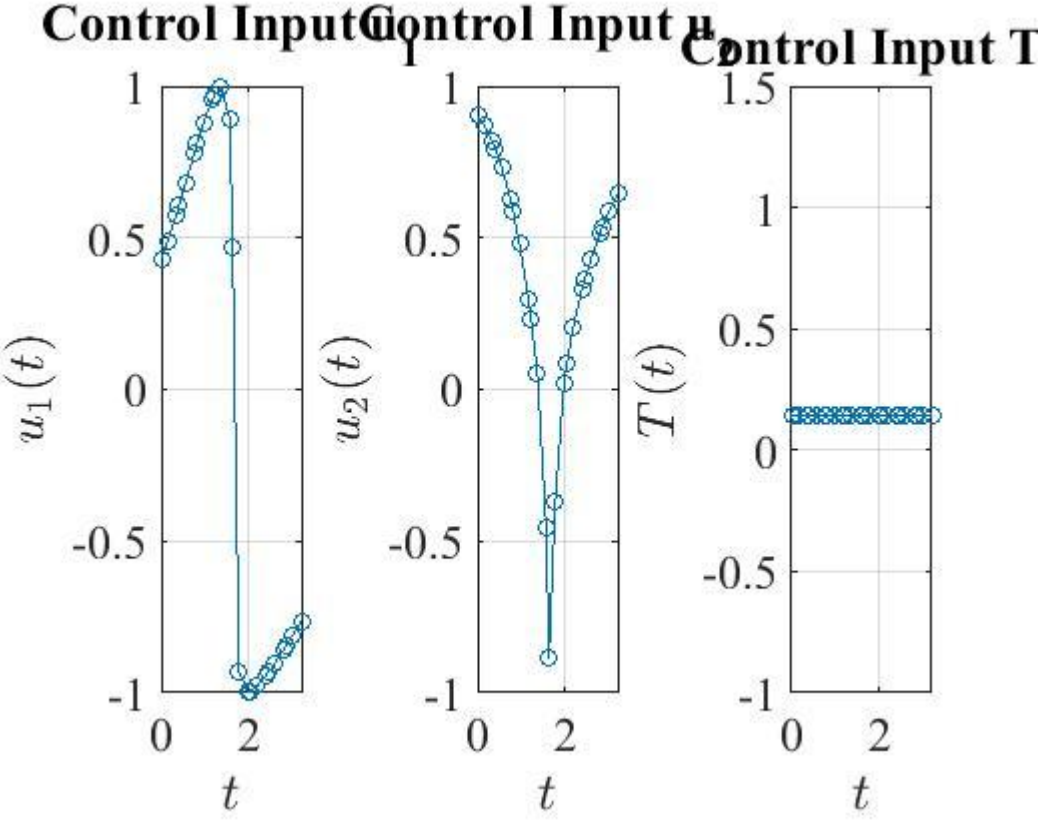


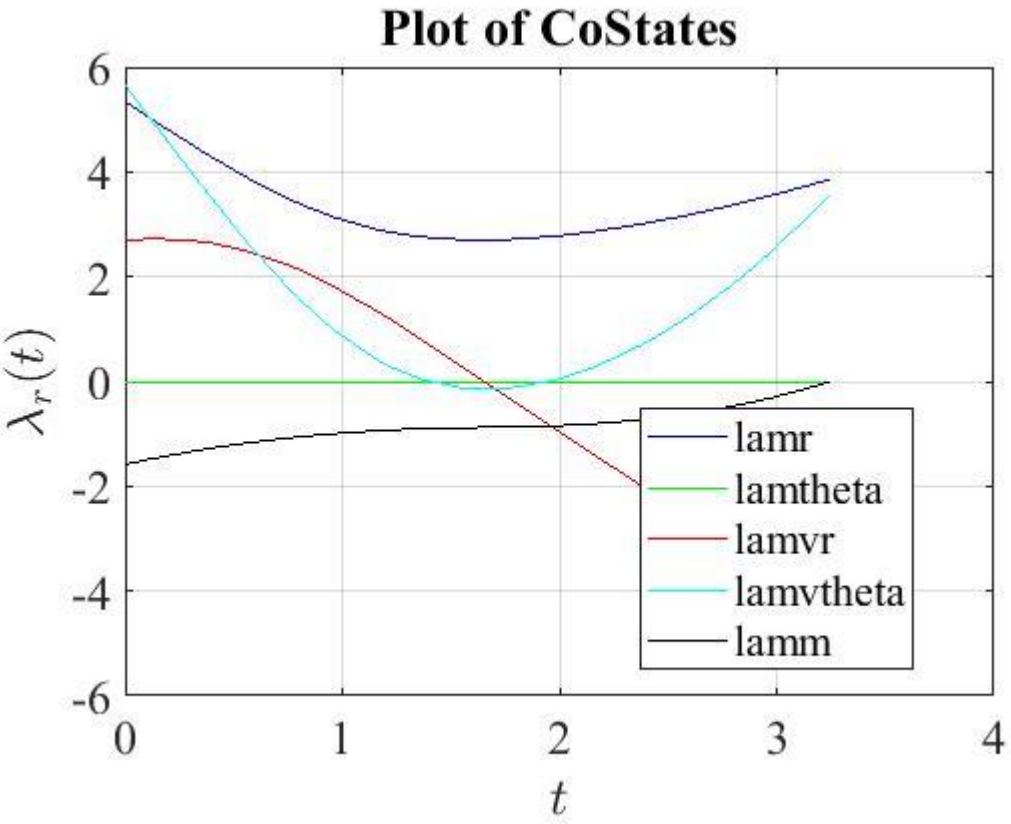
Published with MATLAB® R2021a

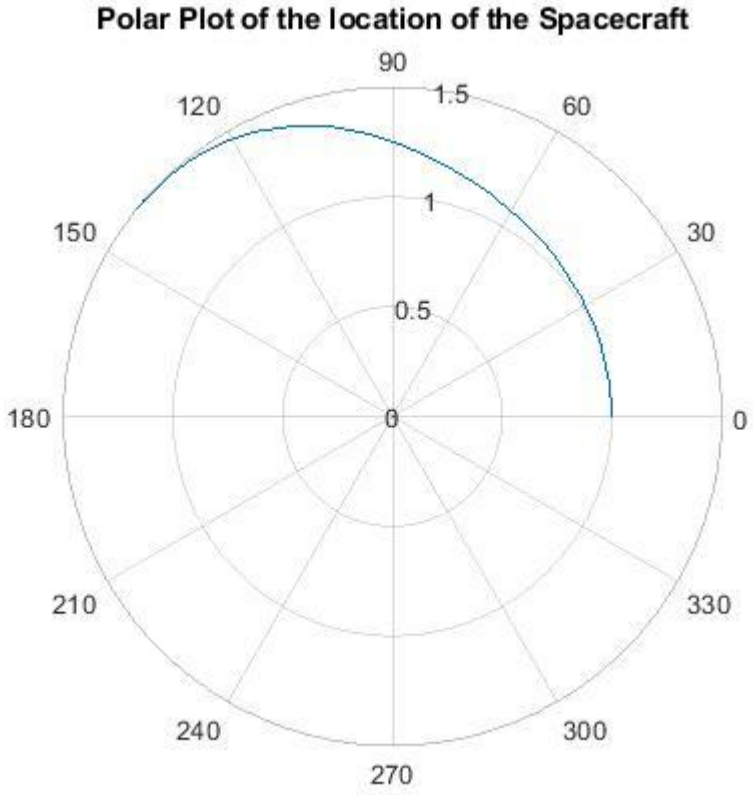
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 8




Throttle angle Beta
200
150
100
50
0
-50
-100
-150
0
1
2
3
$\beta(t)$
t

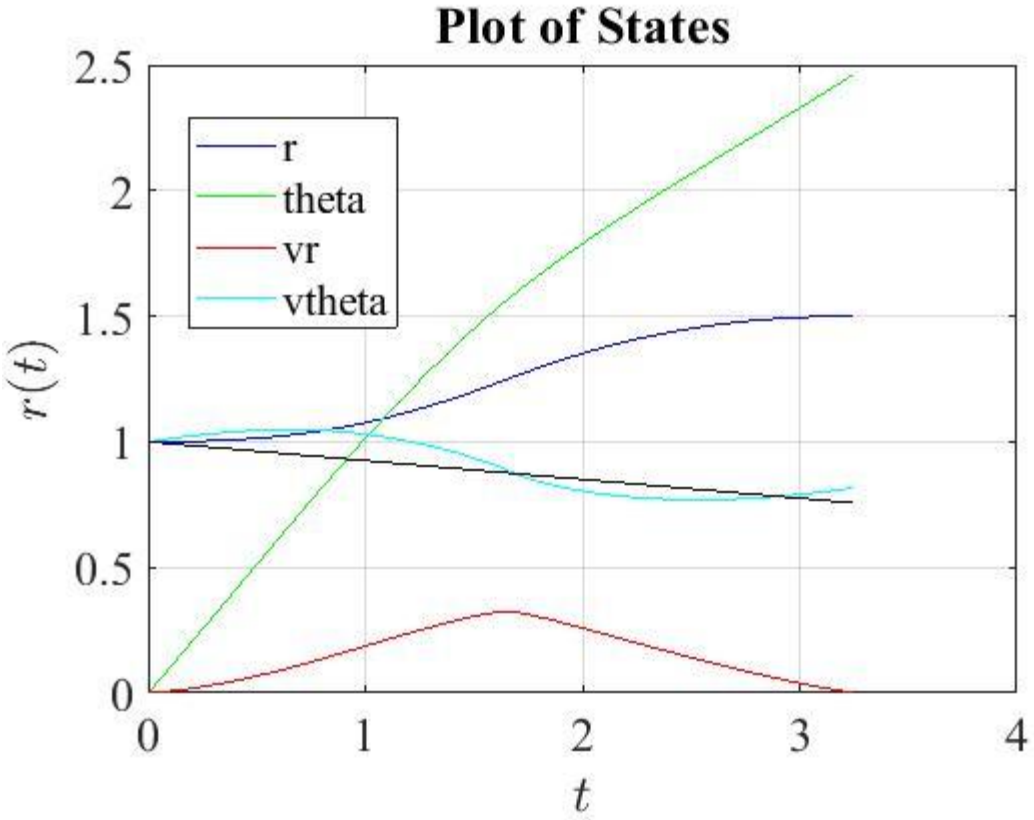

Control Input u1
Control Input u2
Control Input T
1
0.5
0
-0.5
-1
1.5
0
2
$u_1(t)$
$u_2(t)$
$T(t)$
t

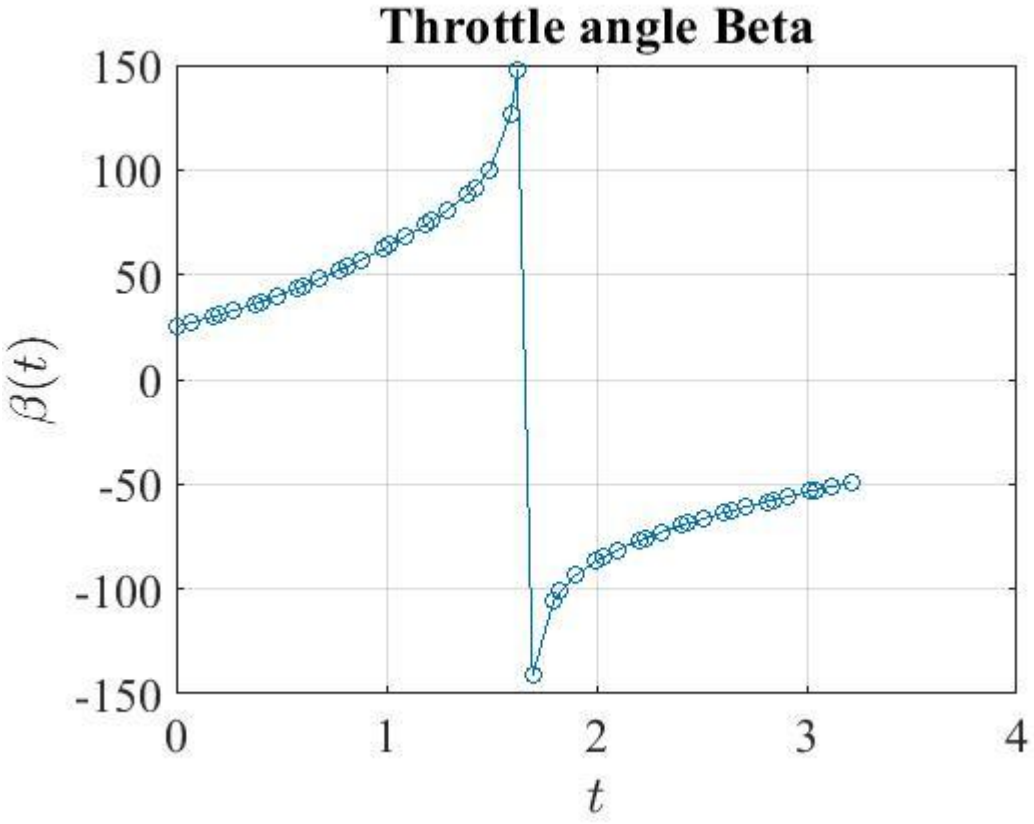


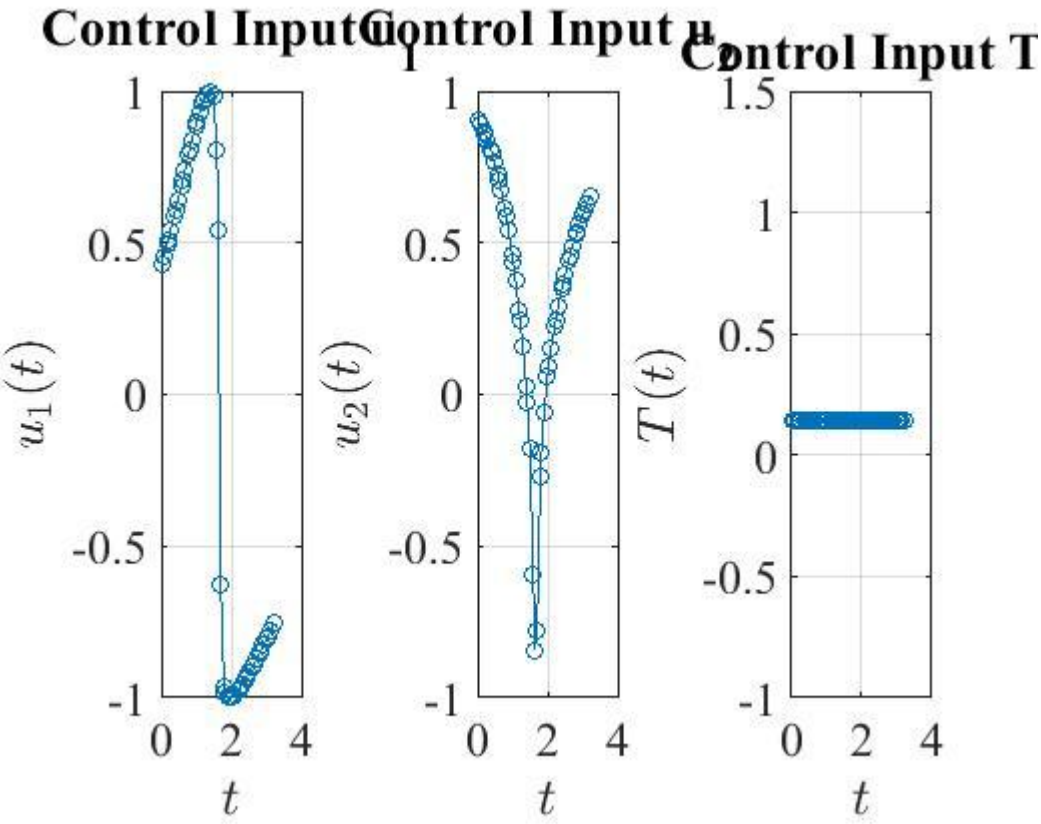


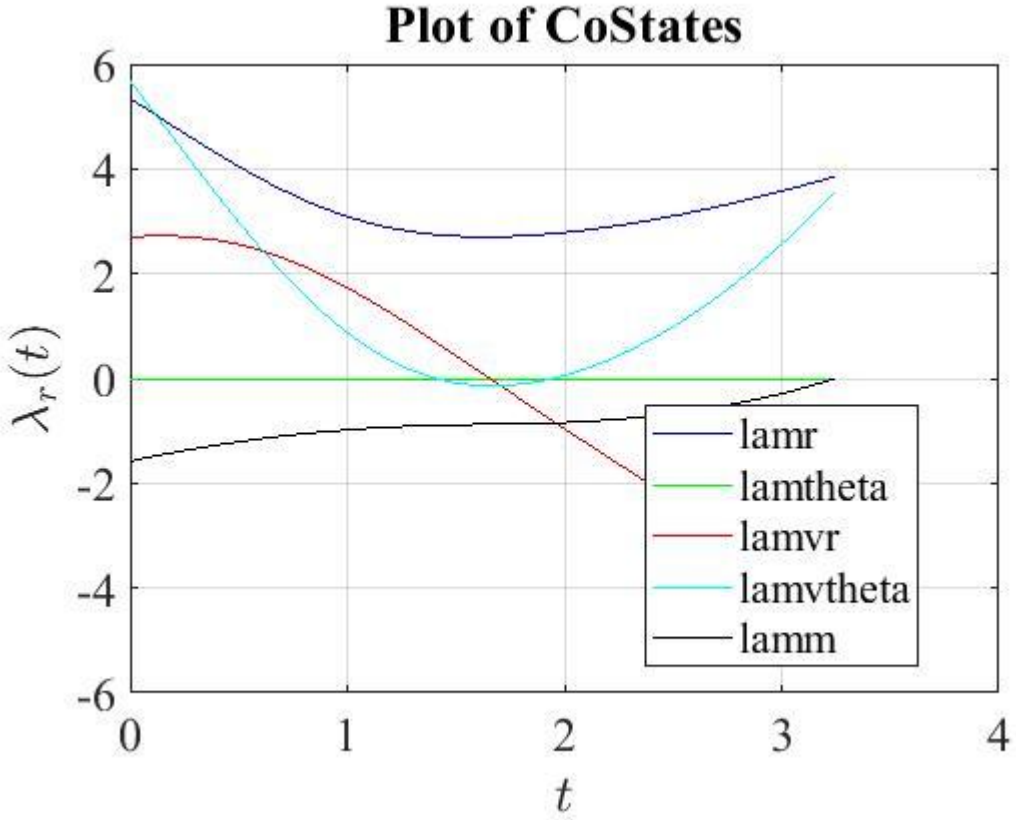
Published with MATLAB® R2021a

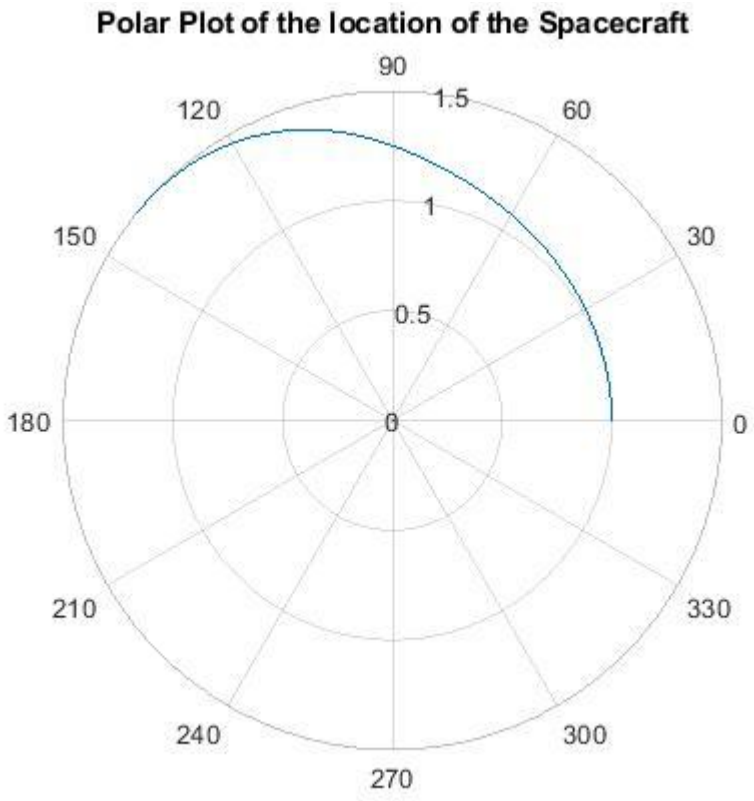
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 16





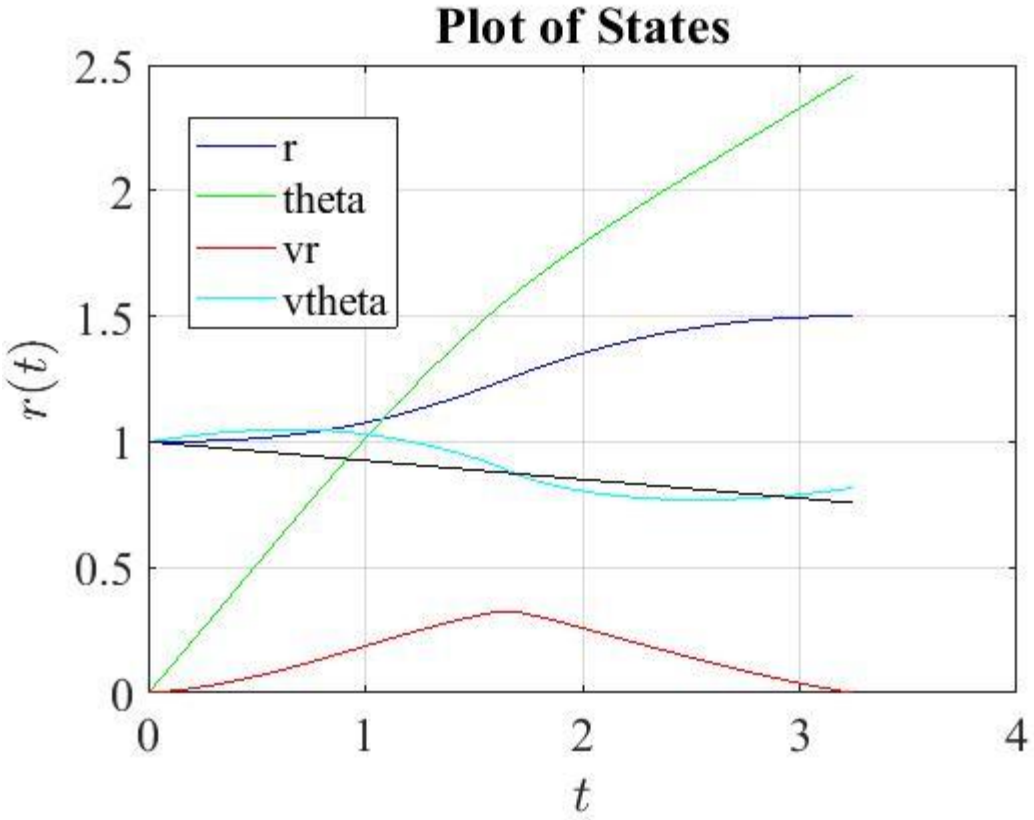

Control Input u1
Control Input u2
Control Input T
u1(t)
u2(t)
T(t)
t
t
t

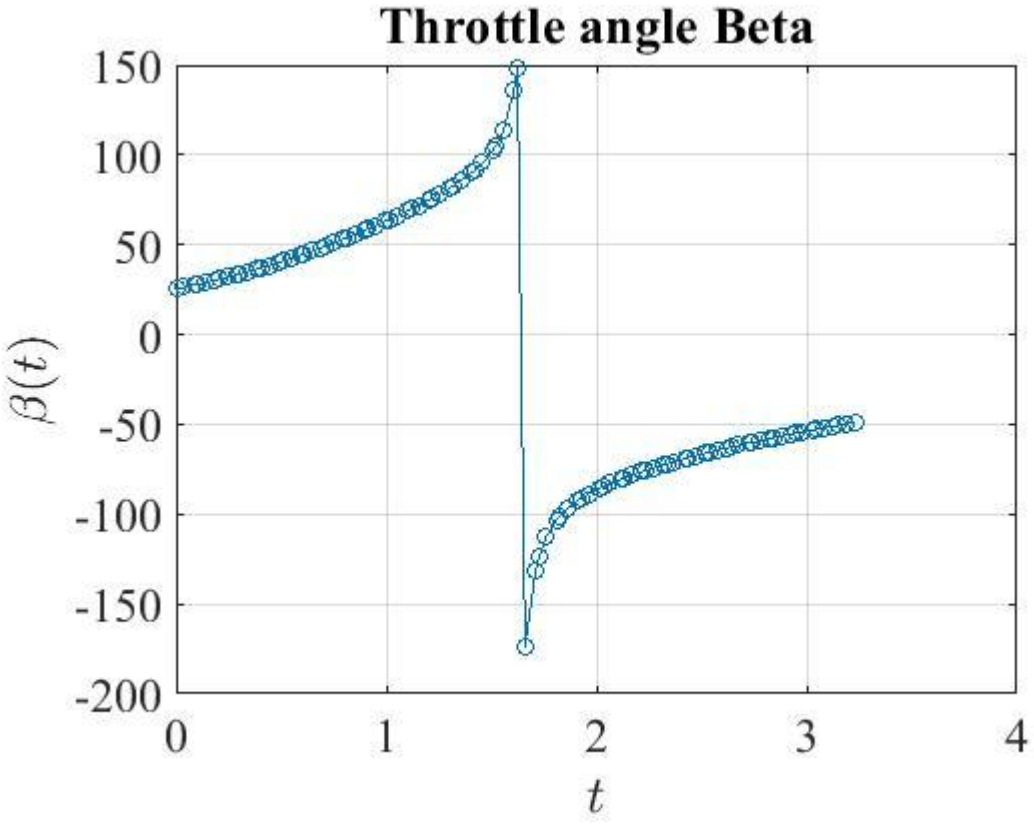

Plot of CoStates
λr(t)
t
lamr
lamtheta
lamvr
lamvtheta
lamm

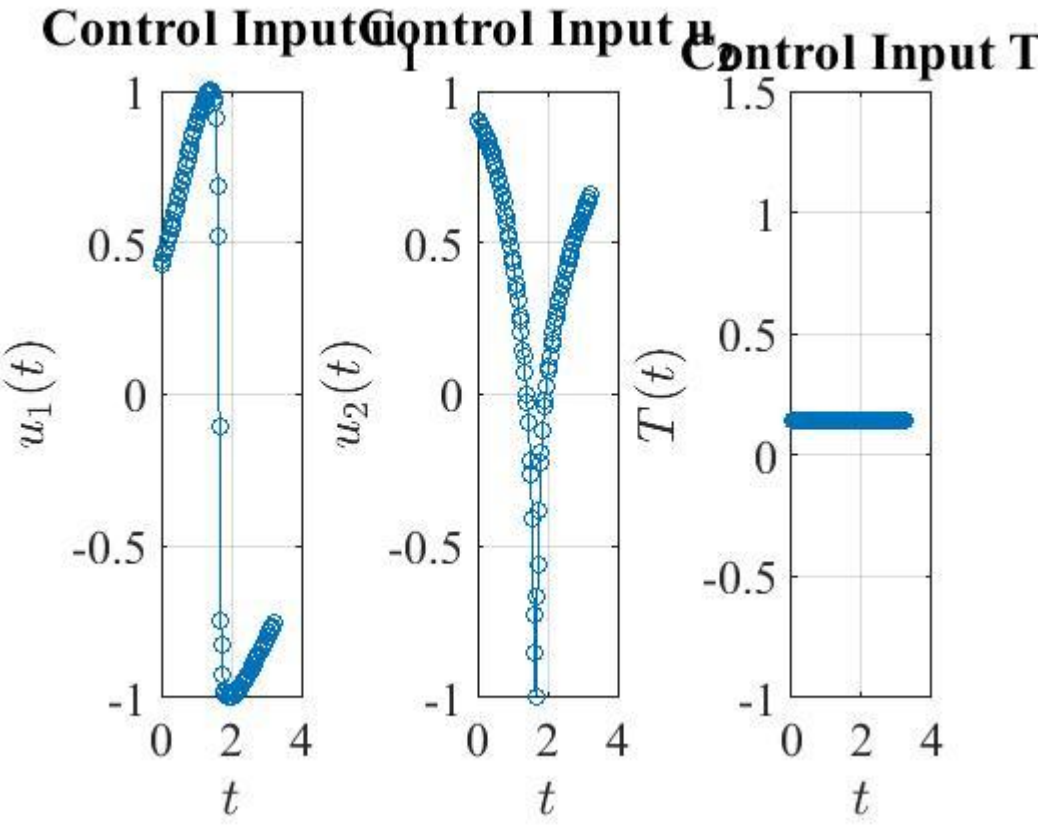


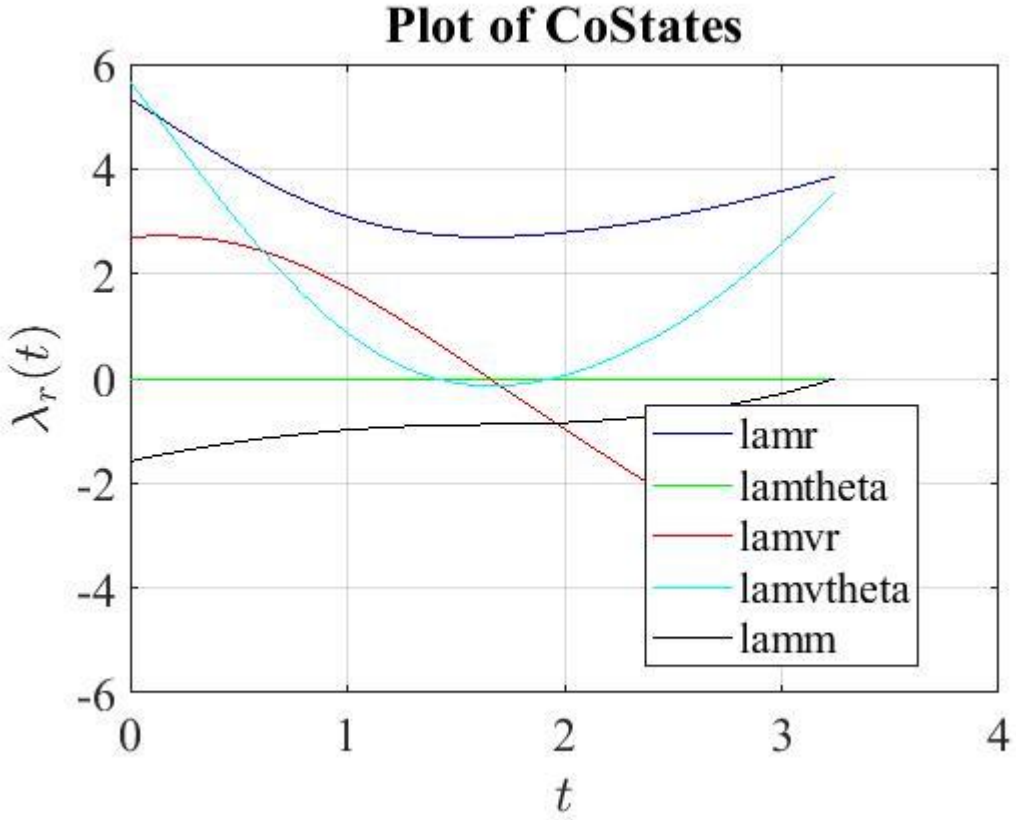
Published with MATLAB® R2021a

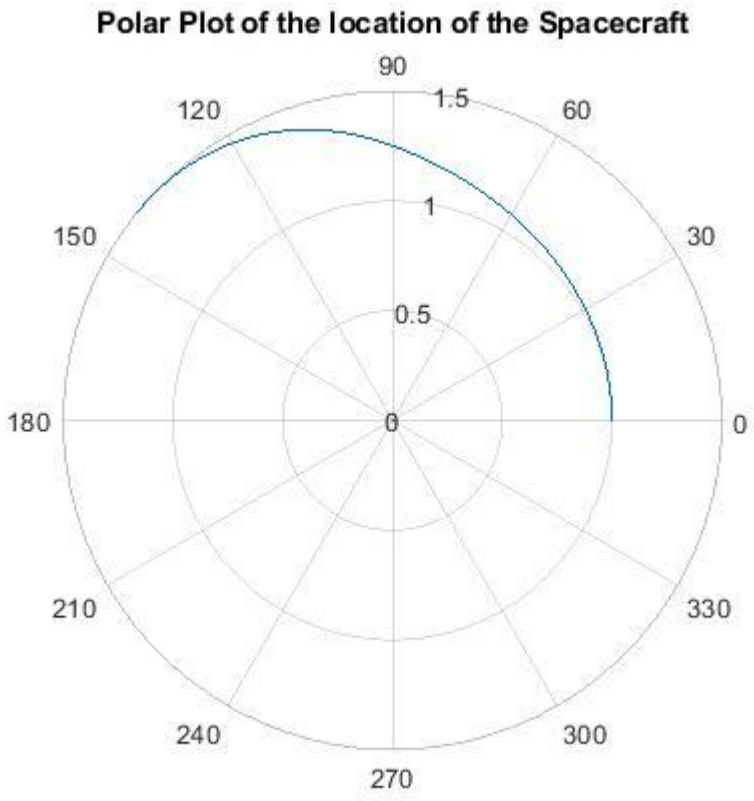
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 32





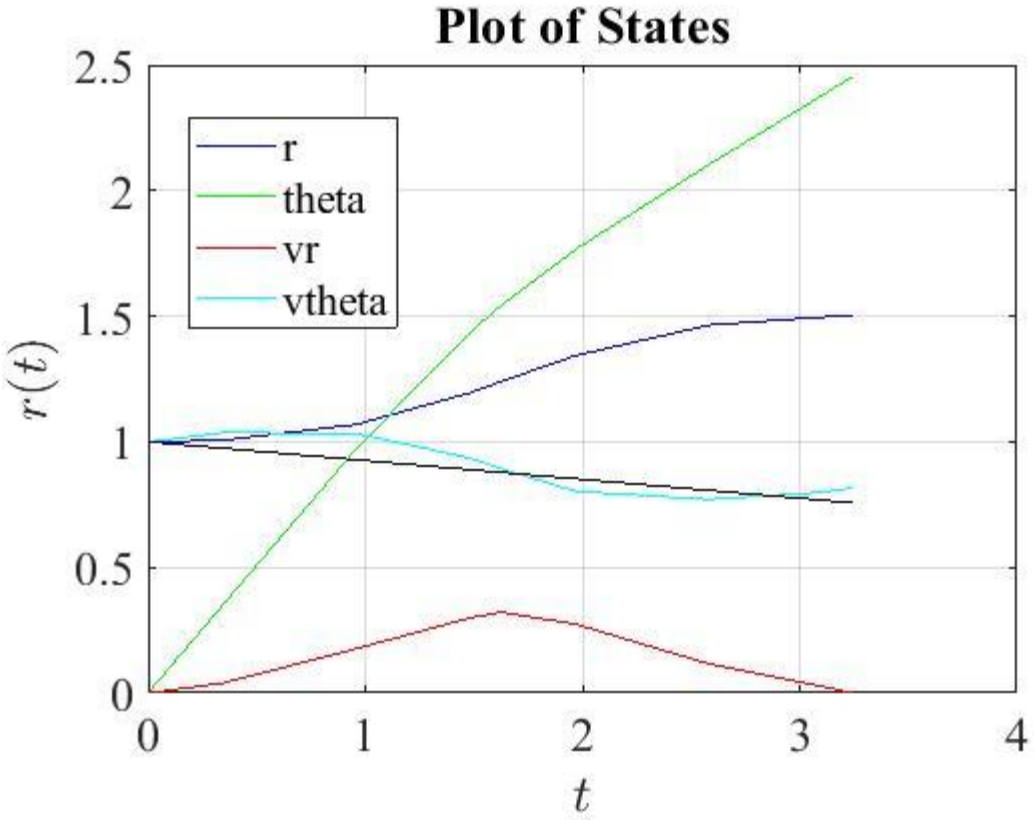

Control Input u1
Control Input u2
Control Input T
u1(t)
u2(t)
T(t)
t
t
t

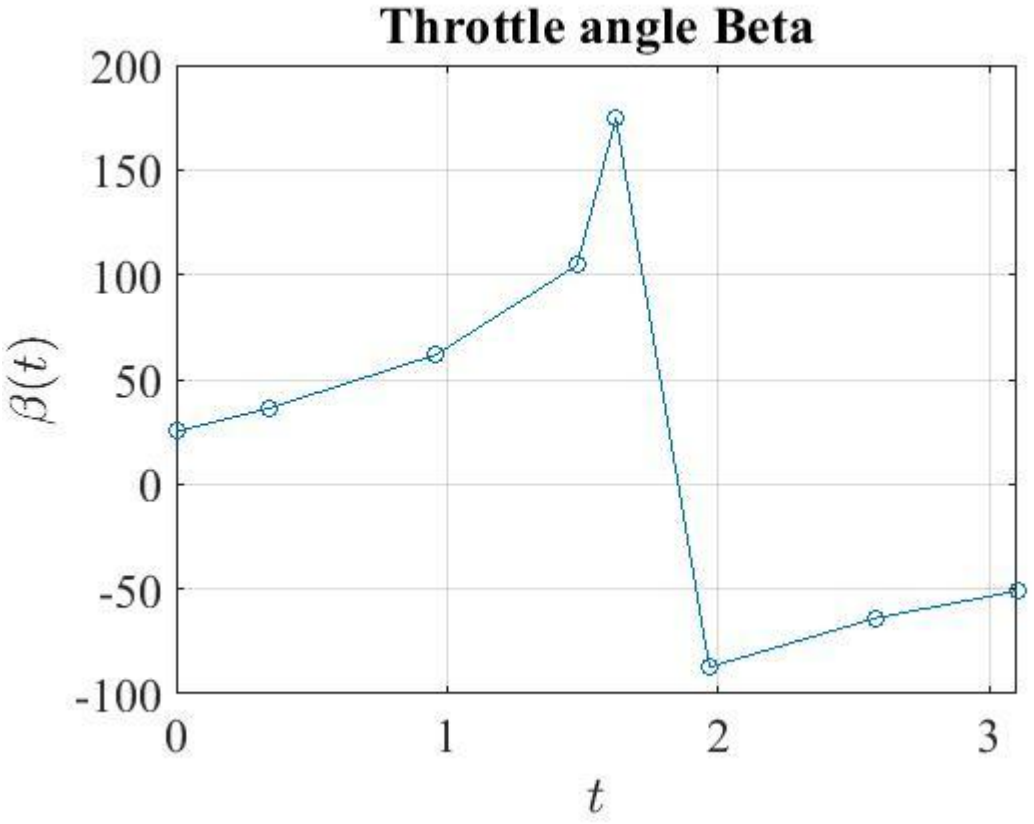

Plot of CoStates
λr(t)
t
lamr
lamtheta
lamvr
lamvtheta
lamm

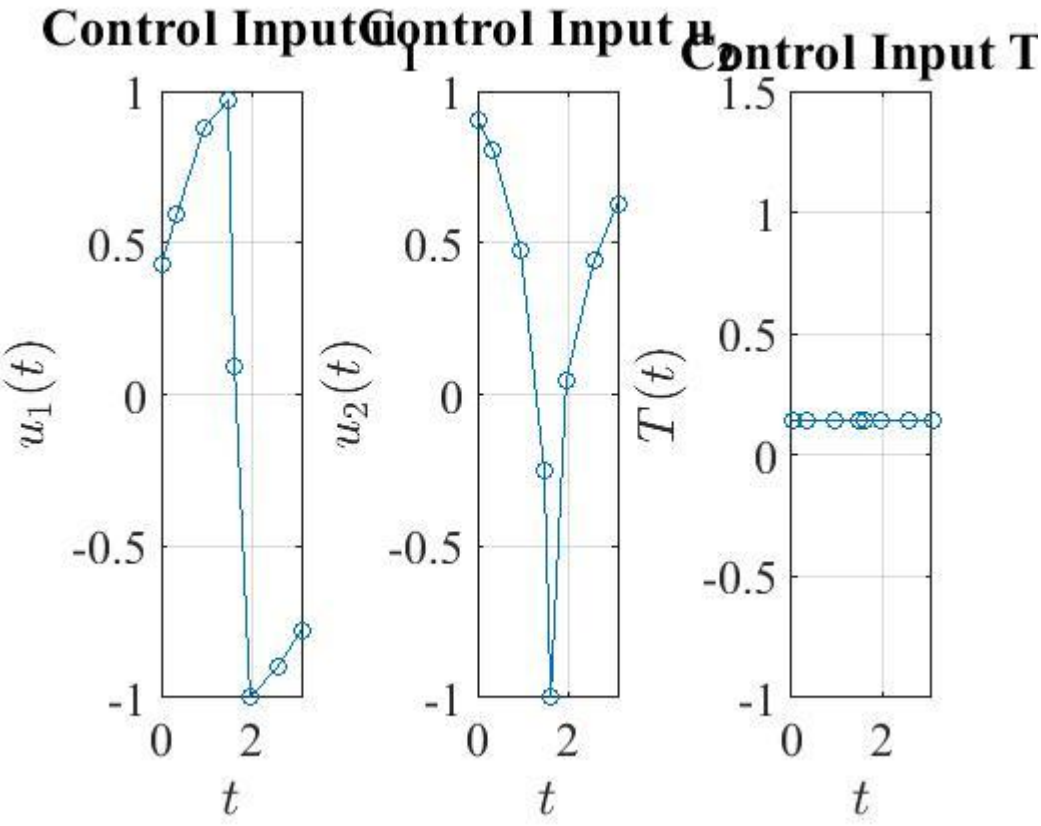


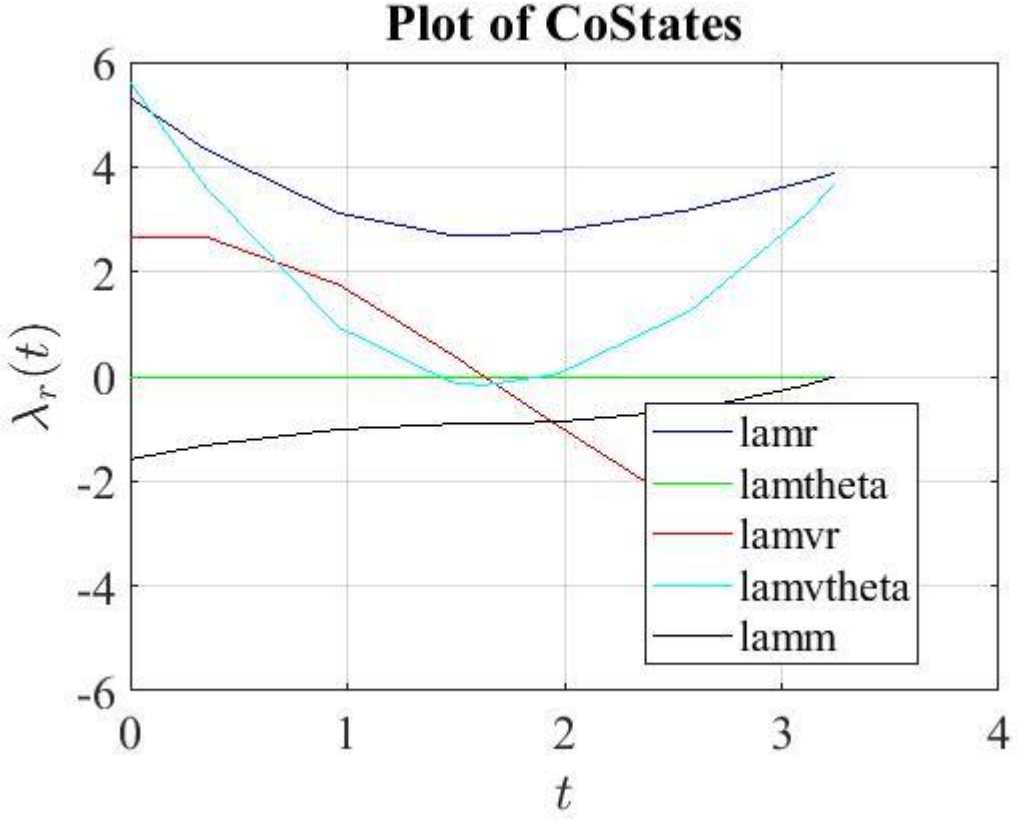
Published with MATLAB® R2021a

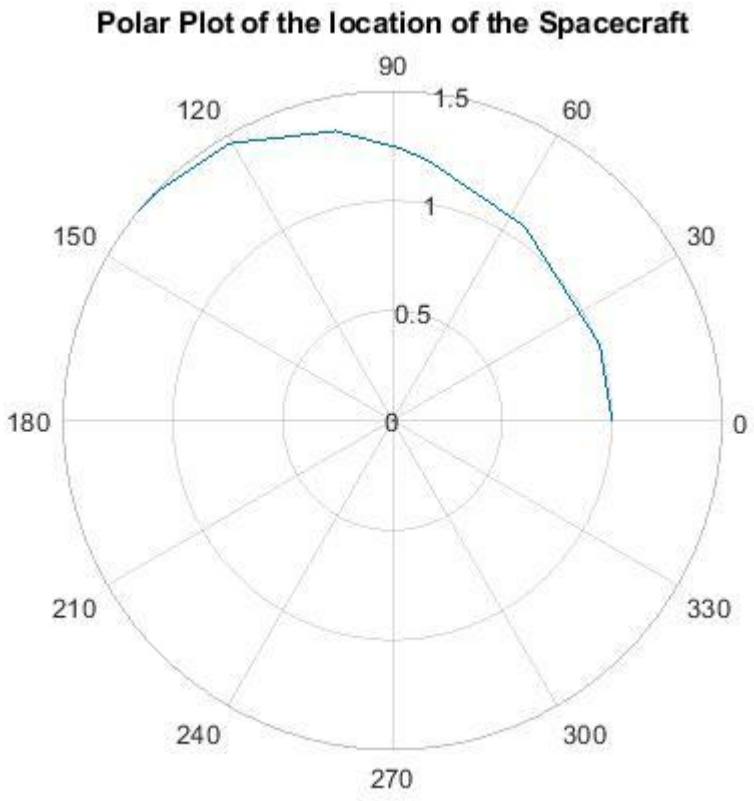
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 2





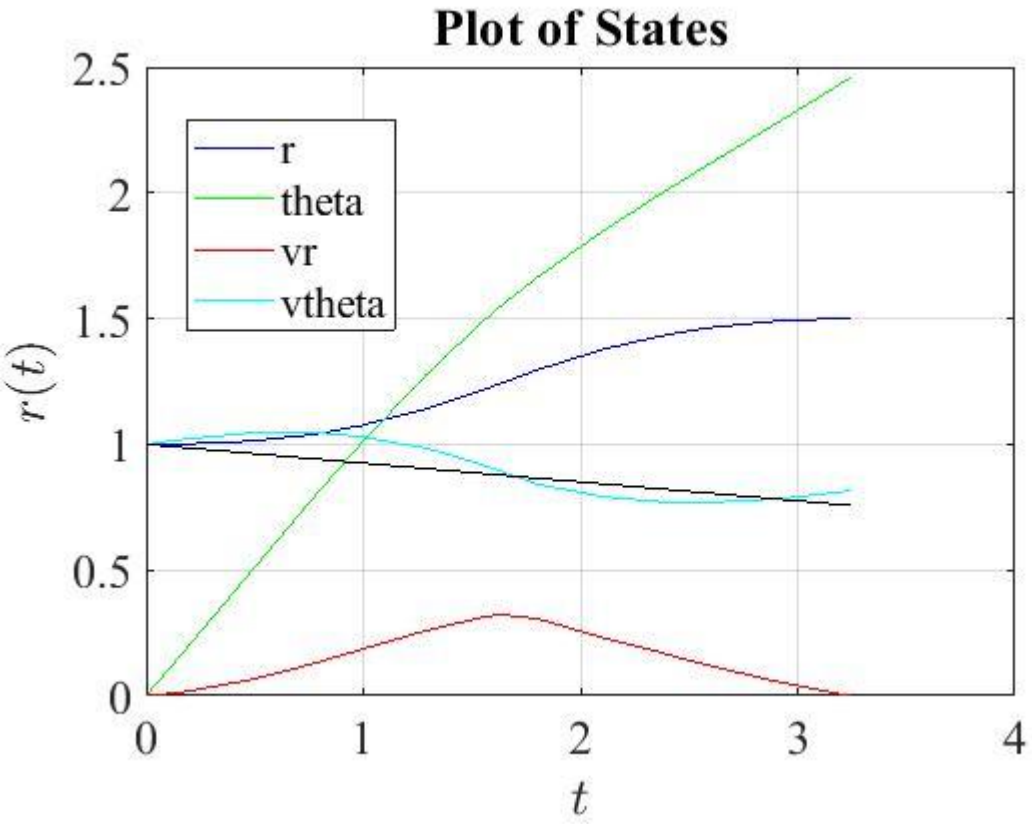

Control Input u1
Control Input u2
Control Input T
1
0.5
0
-0.5
-1
1.5
0
2
t
u1(t)
u2(t)
T(t)

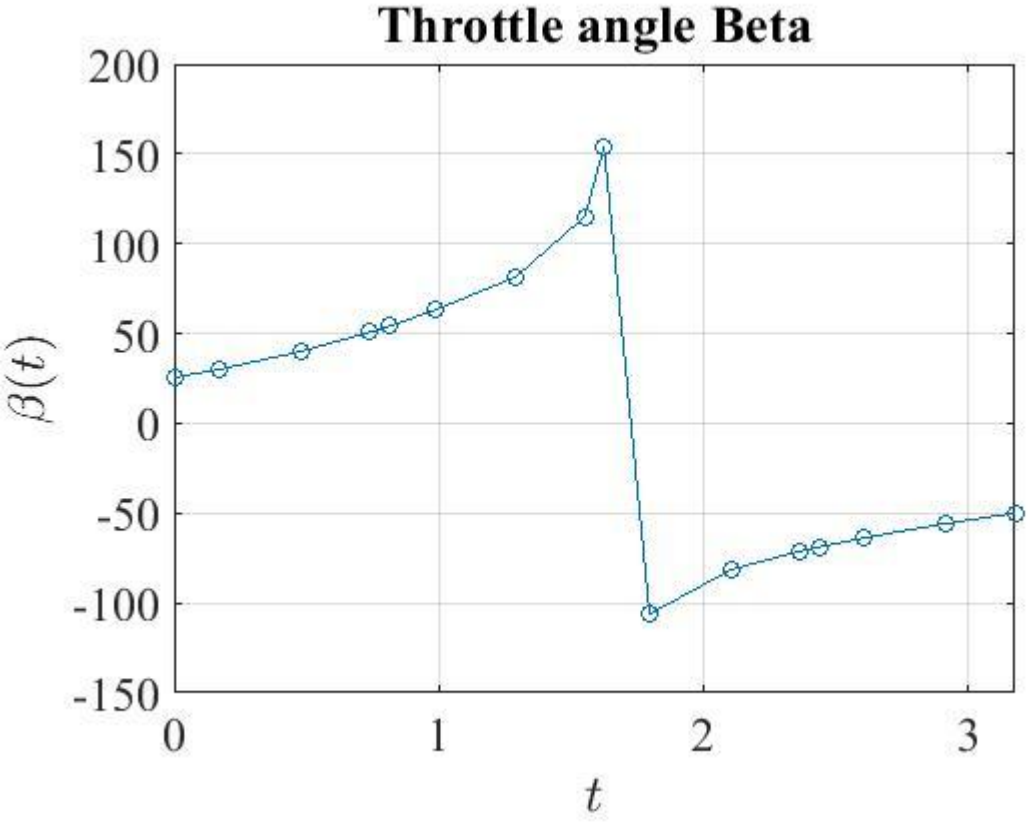

Plot of CoStates
6
4
2
0
-2
-4
-6
0
1
2
3
4
t
λr(t)
lamr
lamtheta
lamvr
lamvtheta
lamm

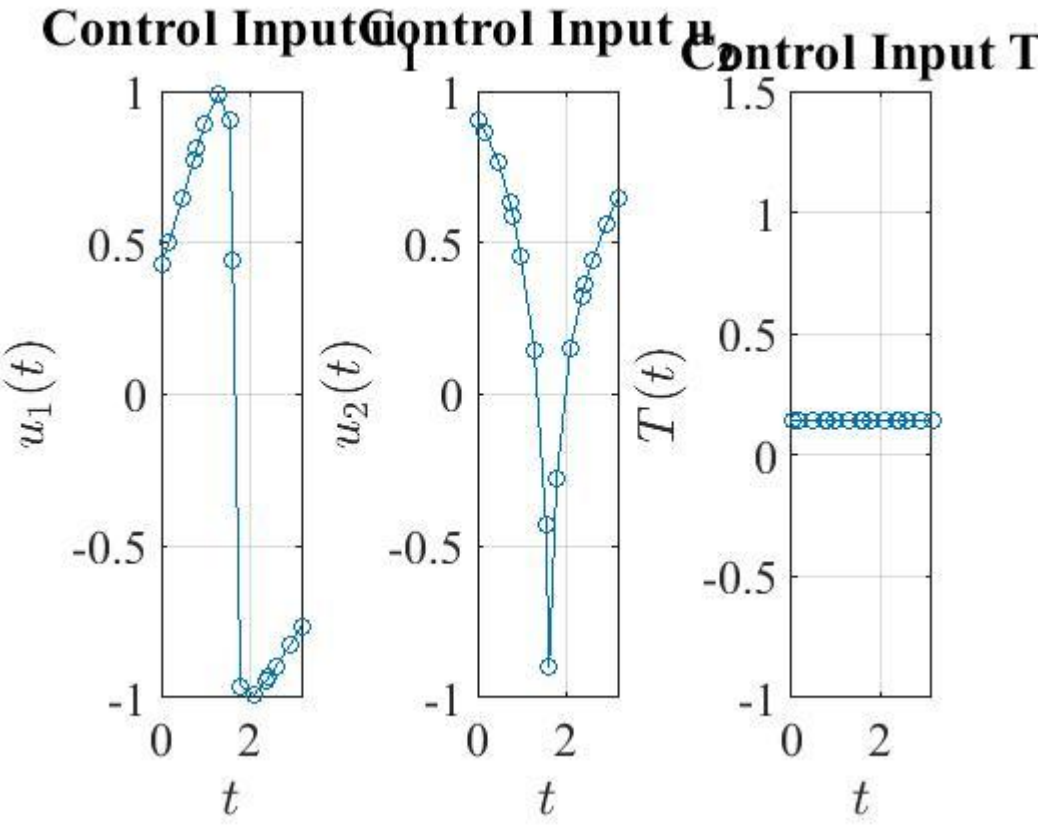


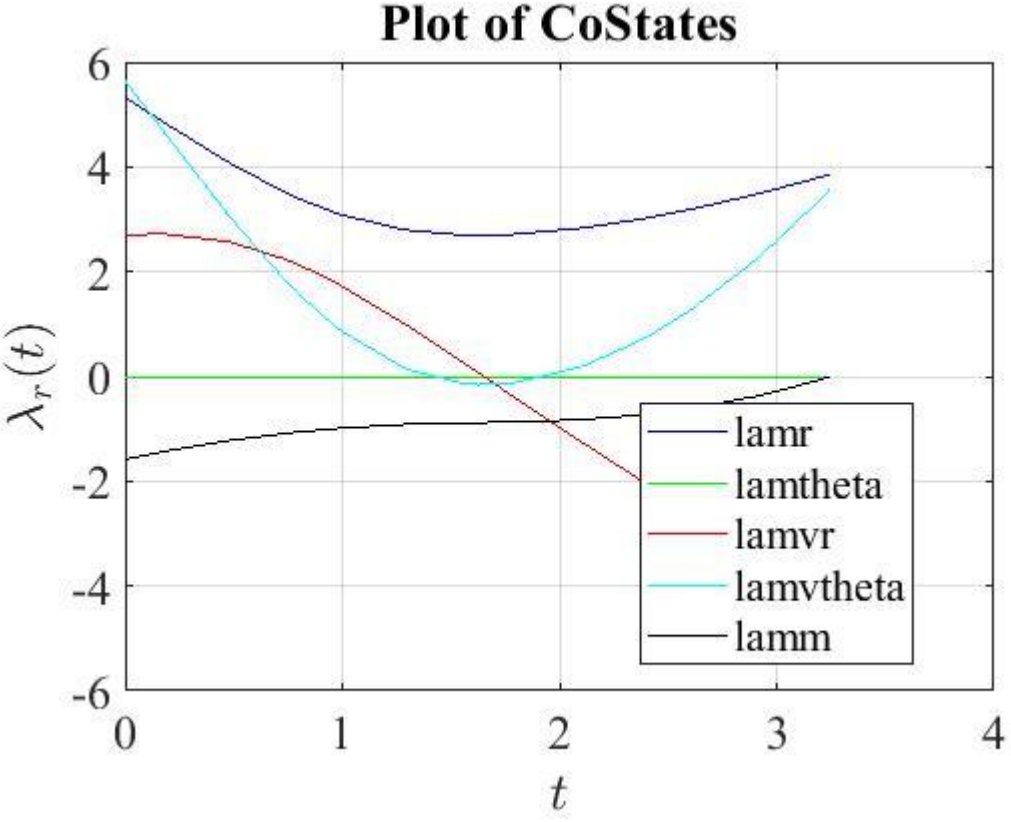
Published with MATLAB® R2021a

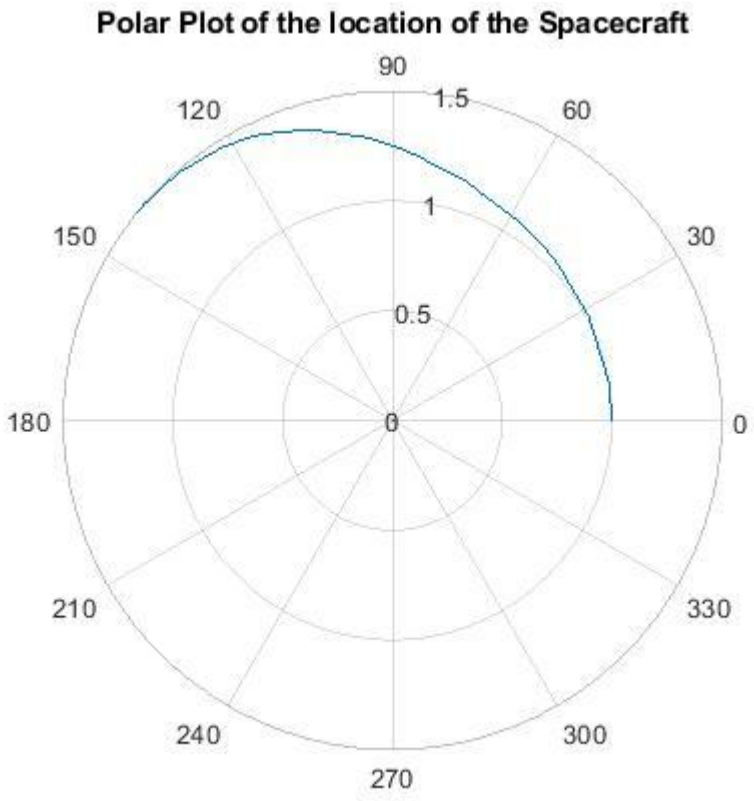
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 4


Plot of States
r
theta
vr
vtheta
r(t)
t
2.5
2
1.5
1
0.5
0
0
1
2
3
4


Throttle angle Beta
β(t)
t
200
150
100
50
0
-50
-100
-150
0
1
2
3

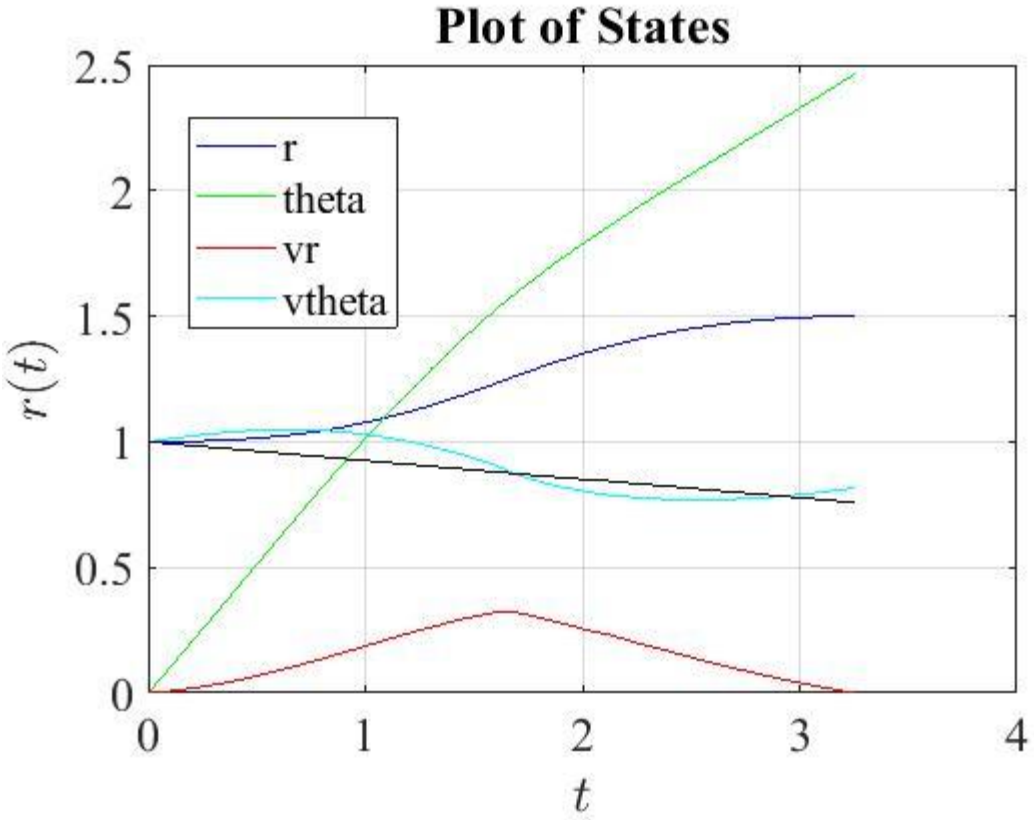

Control Input u1
Control Input u2
Control Input T
u1(t)
u2(t)
T(t)
t
1
0.5
0
-0.5
-1
1.5
0
2

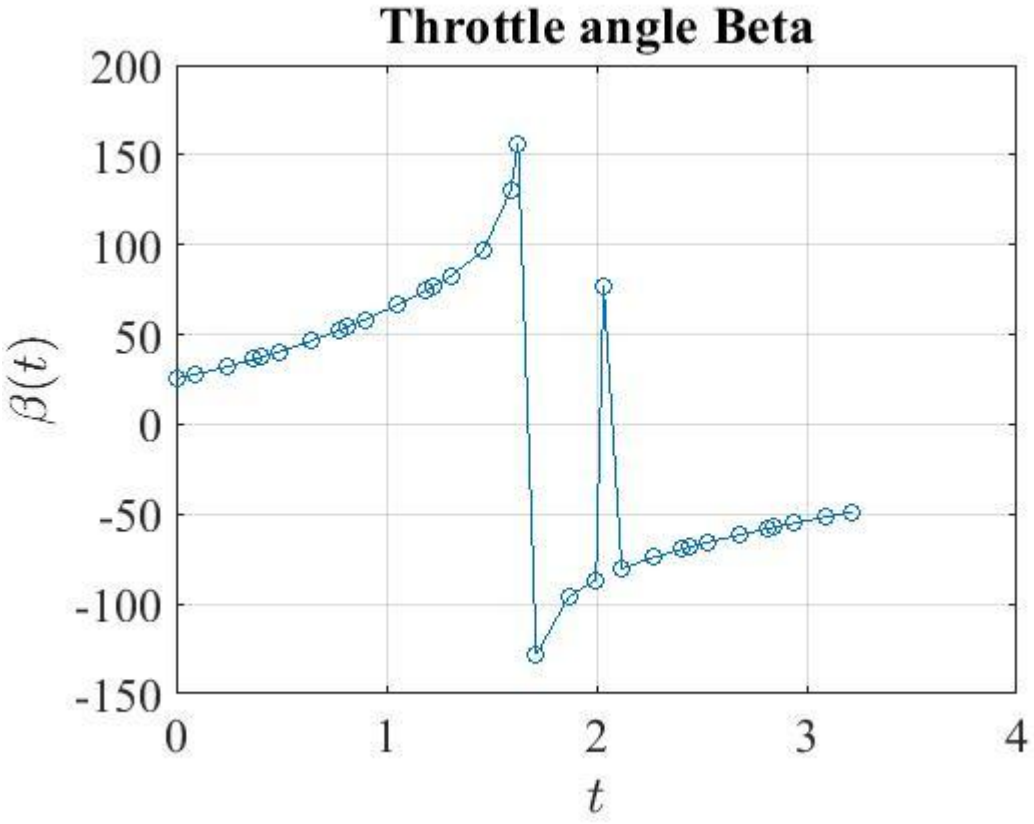

Plot of CoStates
λr(t)
t
lamr
lamtheta
lamvr
lamvtheta
lamm
6
4
2
0
-2
-4
-6
0
1
2
3
4

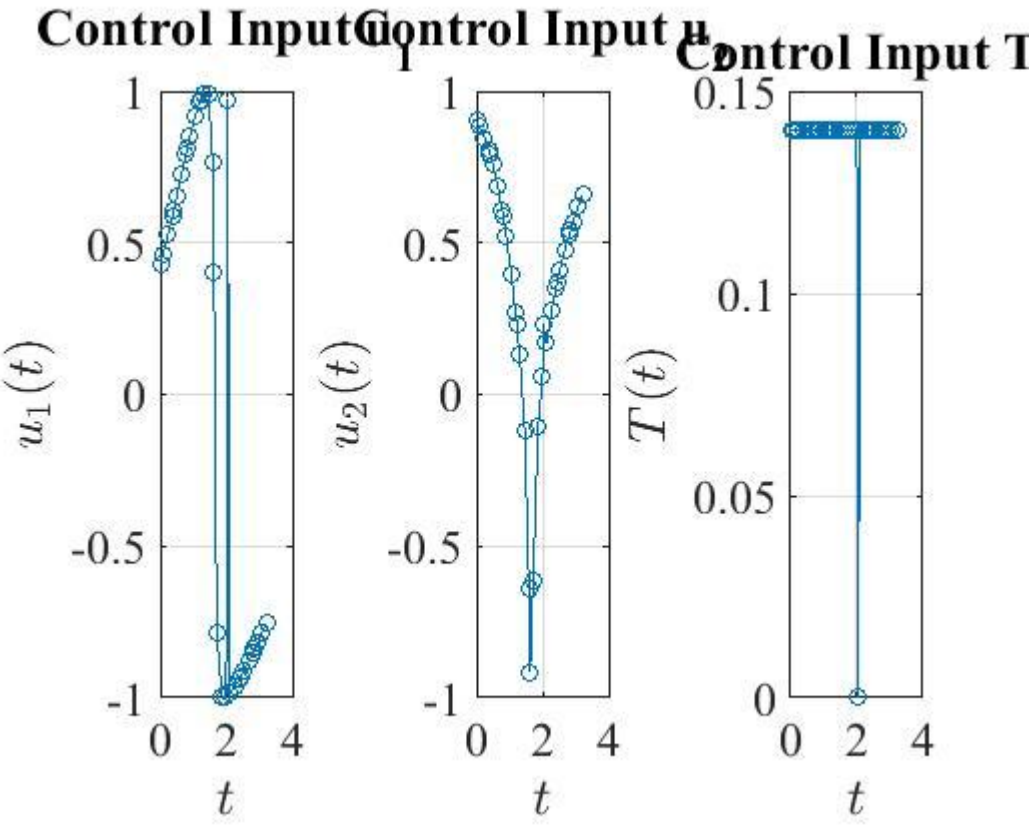


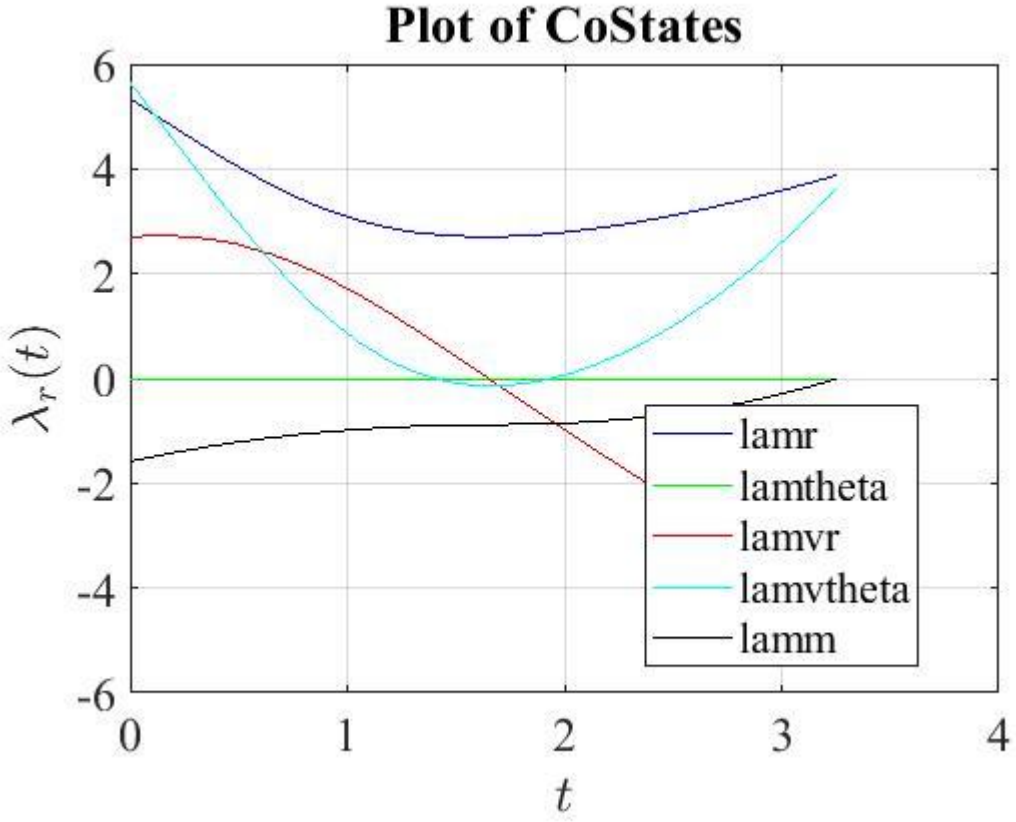
Published with MATLAB® R2021a

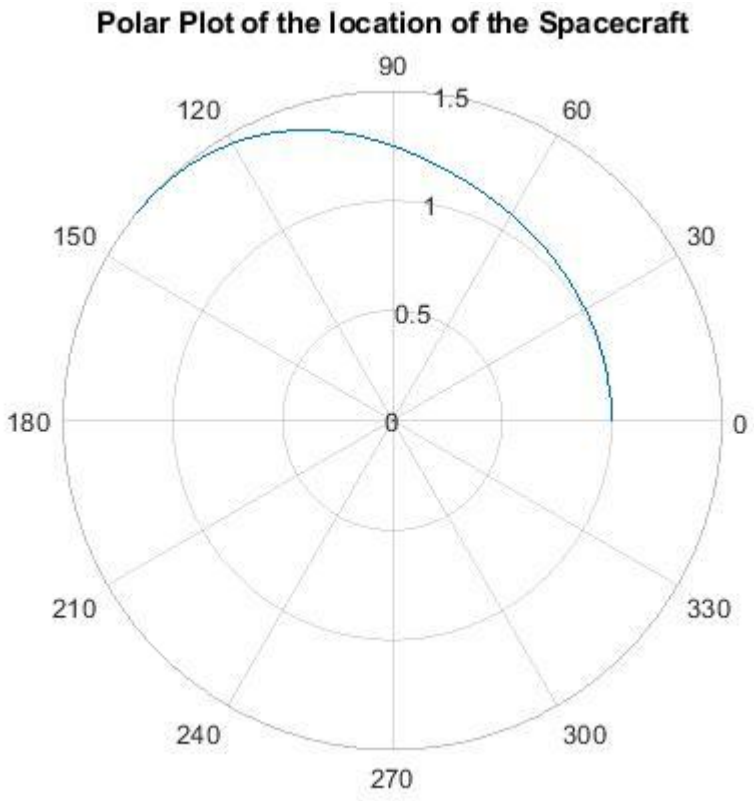
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 8





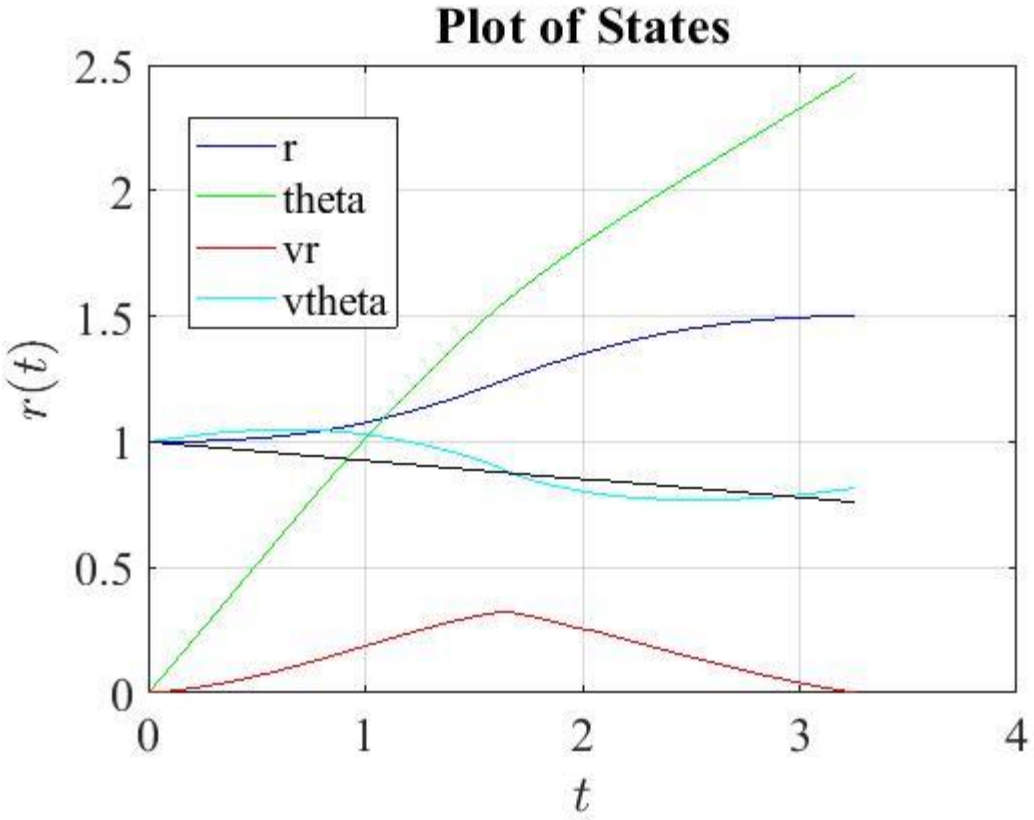

Control Input u1
Control Input u2
Control Input T
u1(t)
u2(t)
T(t)
t
t
t

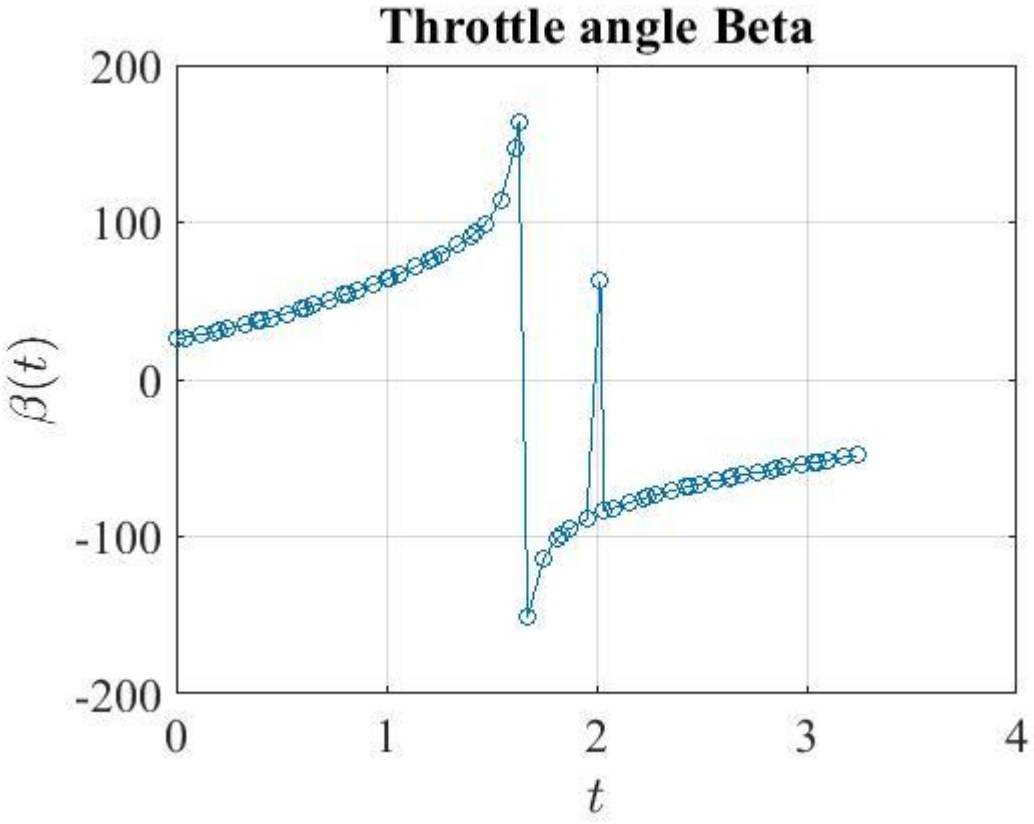

Plot of CoStates
λr(t)
t
lamr
lamtheta
lamvr
lamvtheta
lamm

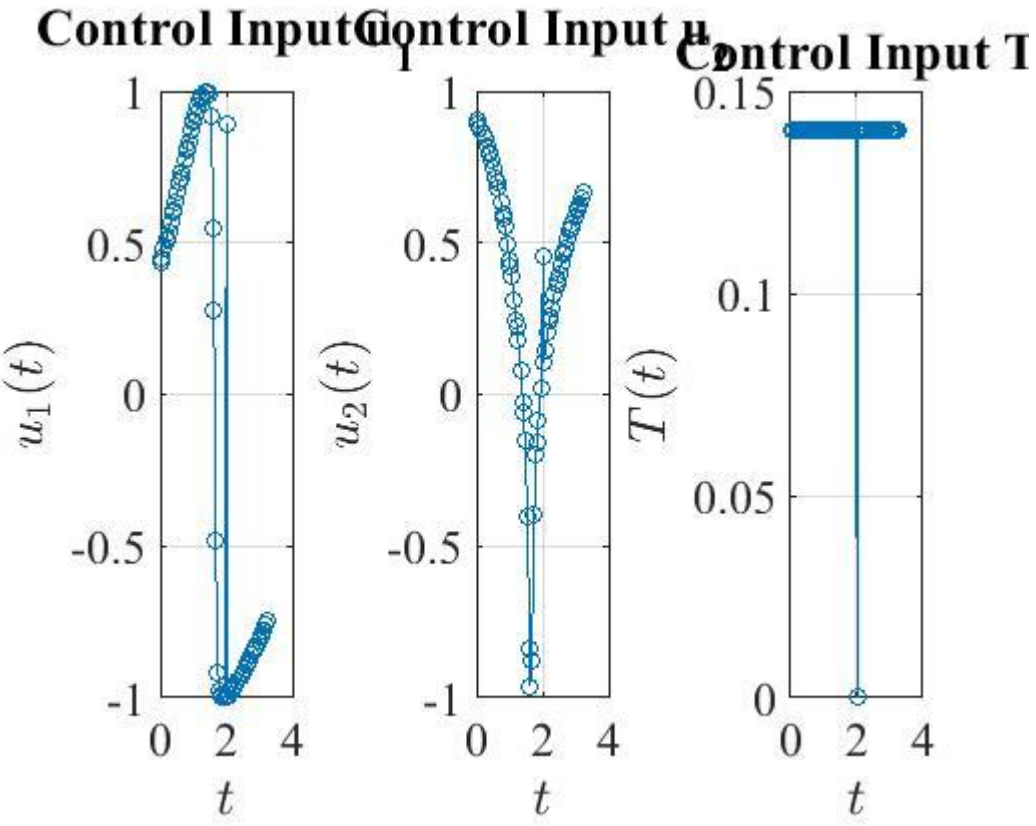


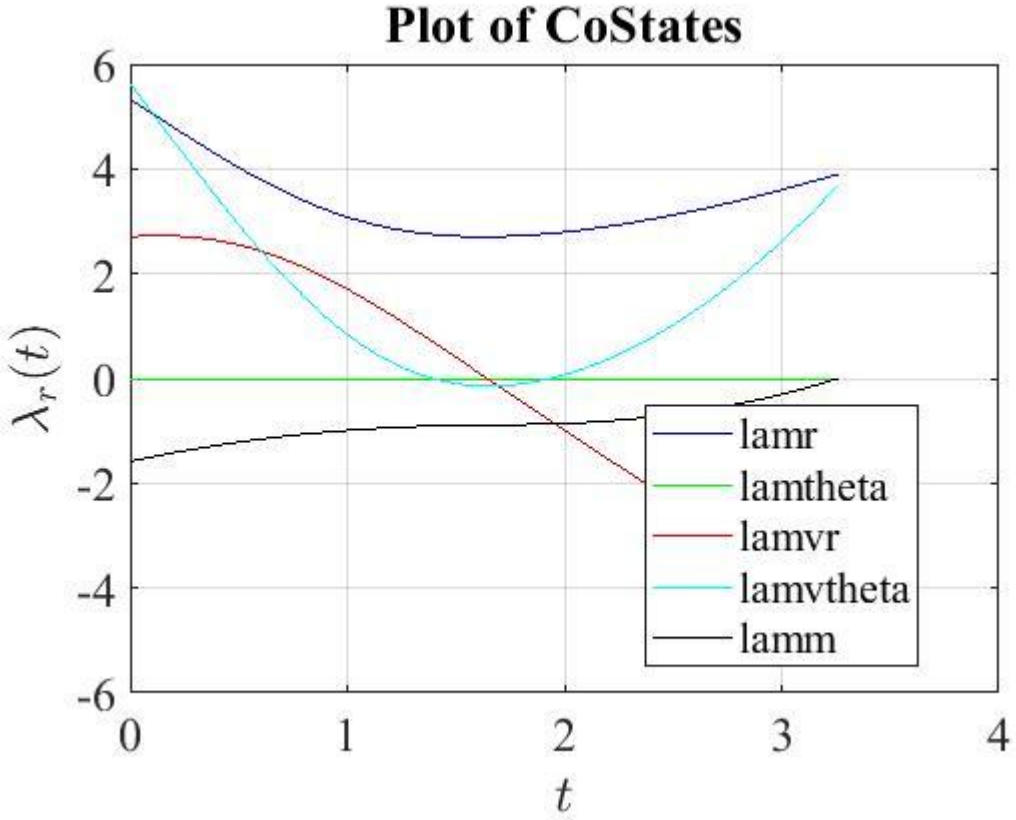
Published with MATLAB® R2021a

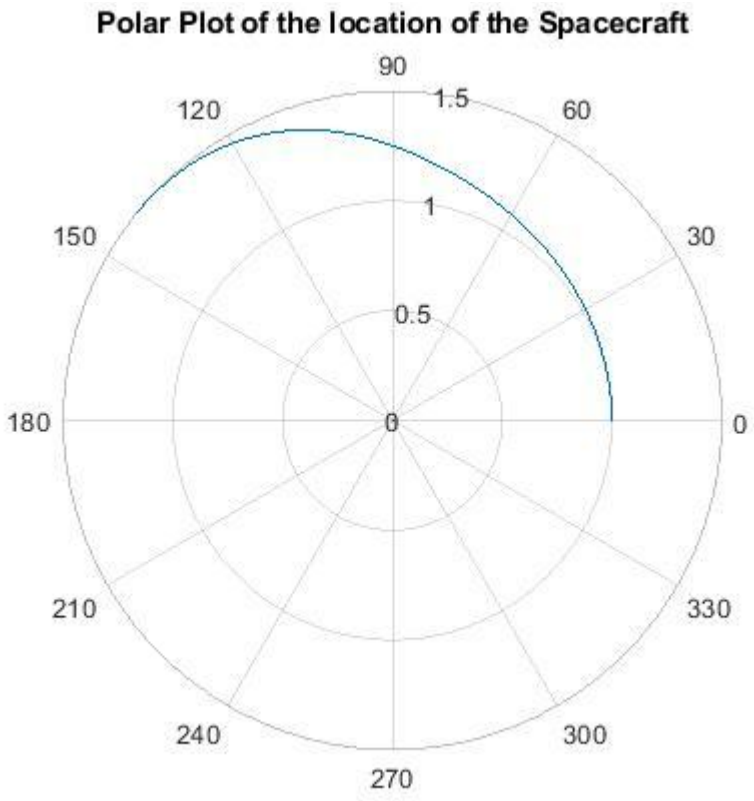
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 16





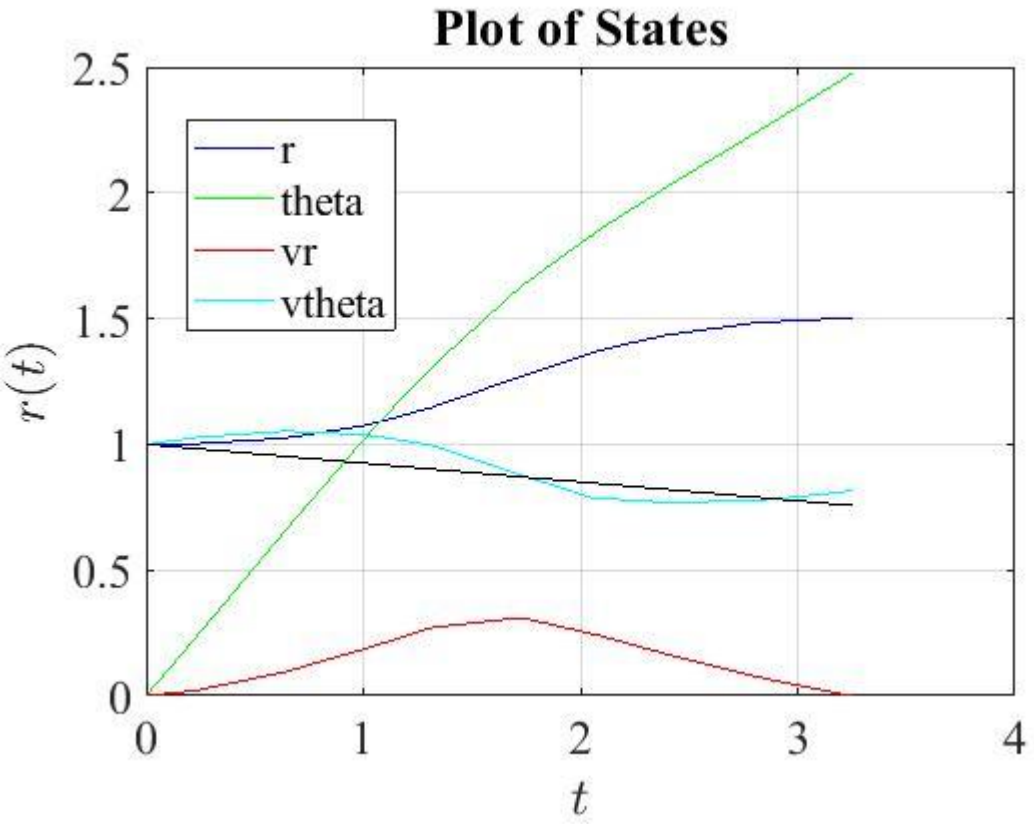

Control Input u1
Control Input u2
Control Input T
u1(t)
u2(t)
T(t)
t
t
t

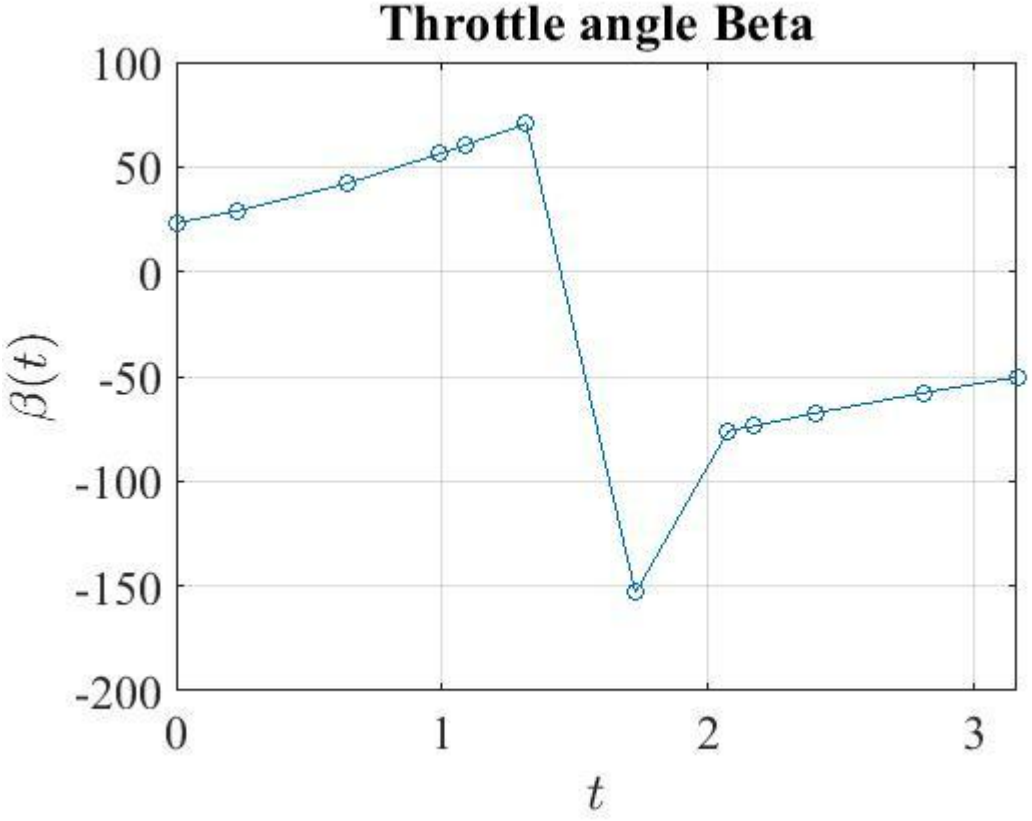

Plot of CoStates
λr(t)
t
lamr
lamtheta
lamvr
lamvtheta
lamm

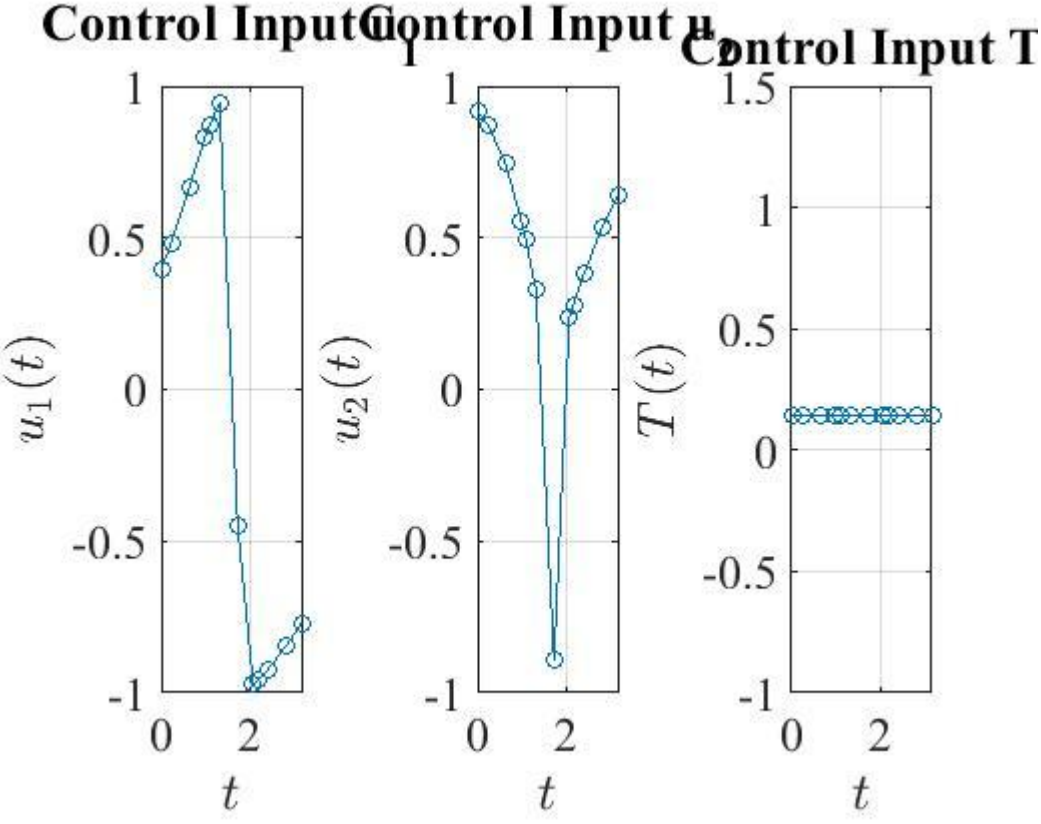


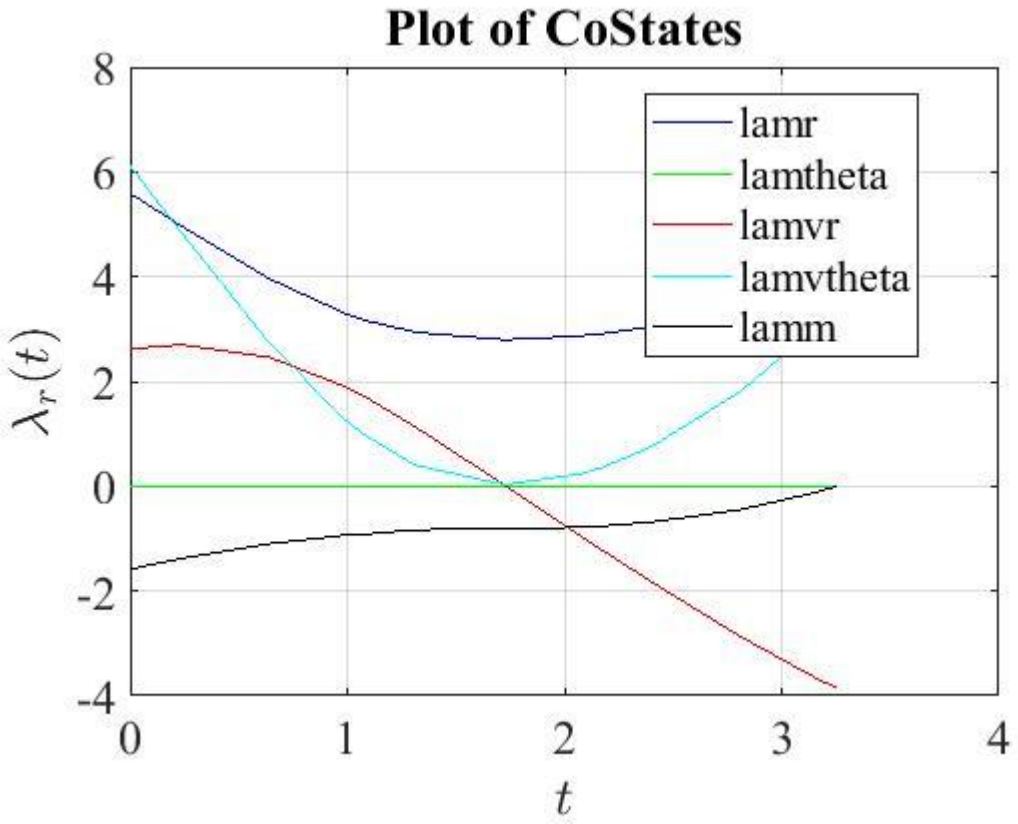
Published with MATLAB® R2021a

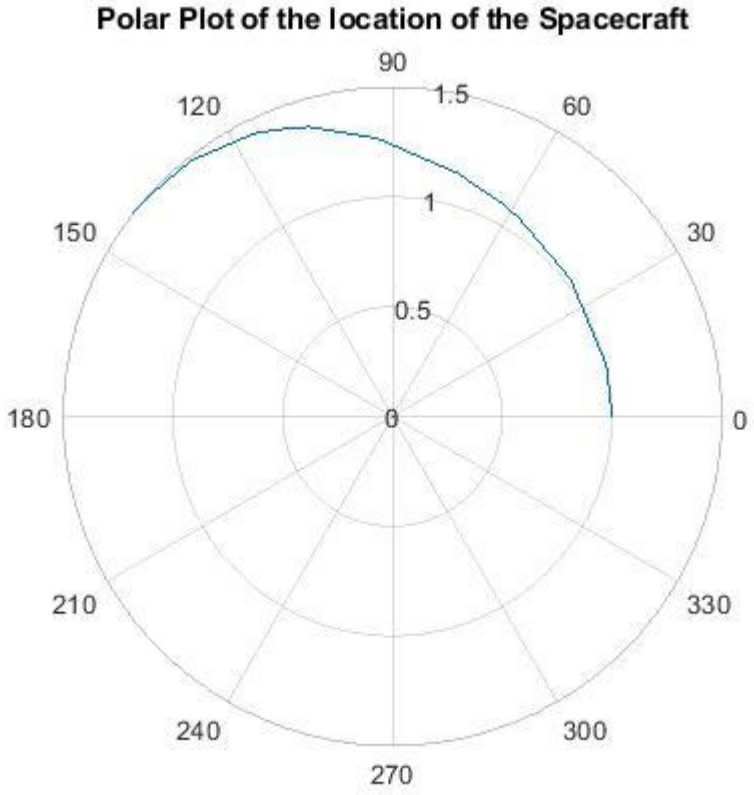
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 16




Throttle angle Beta
100
50
0
-50
-100
-150
-200
$\beta(t)$
0
1
2
3
t


Control Input u1
Control Input u2
Control Input T
1
0.5
0
-0.5
-1
$u_1(t)$
$u_2(t)$
$T(t)$
1.5
0
2
t

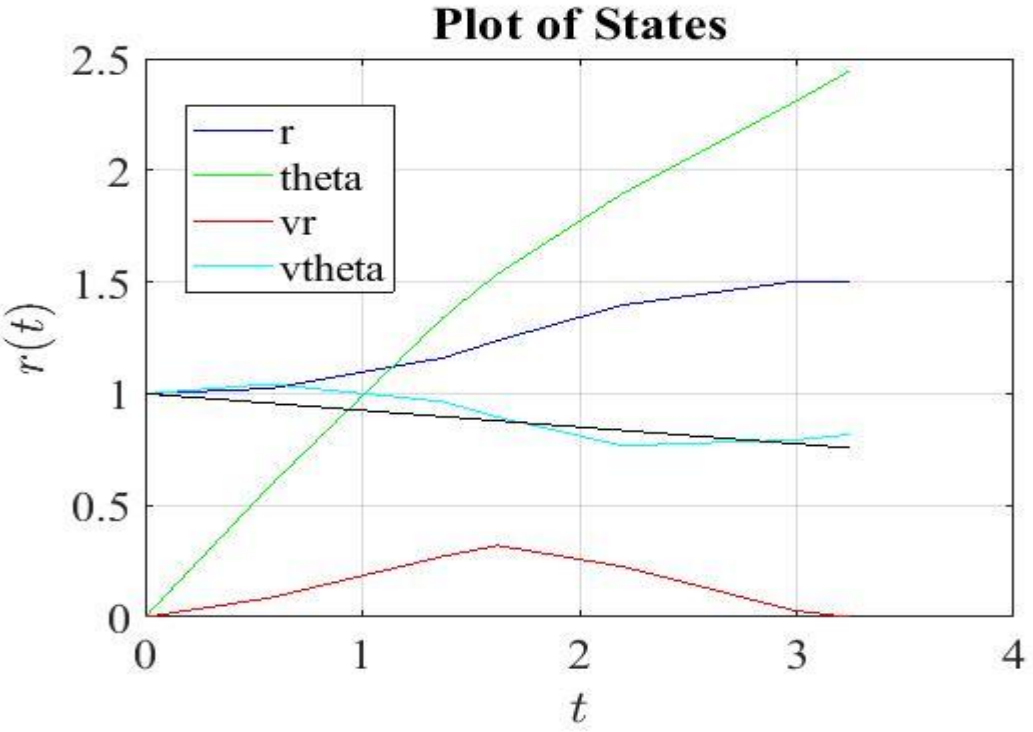


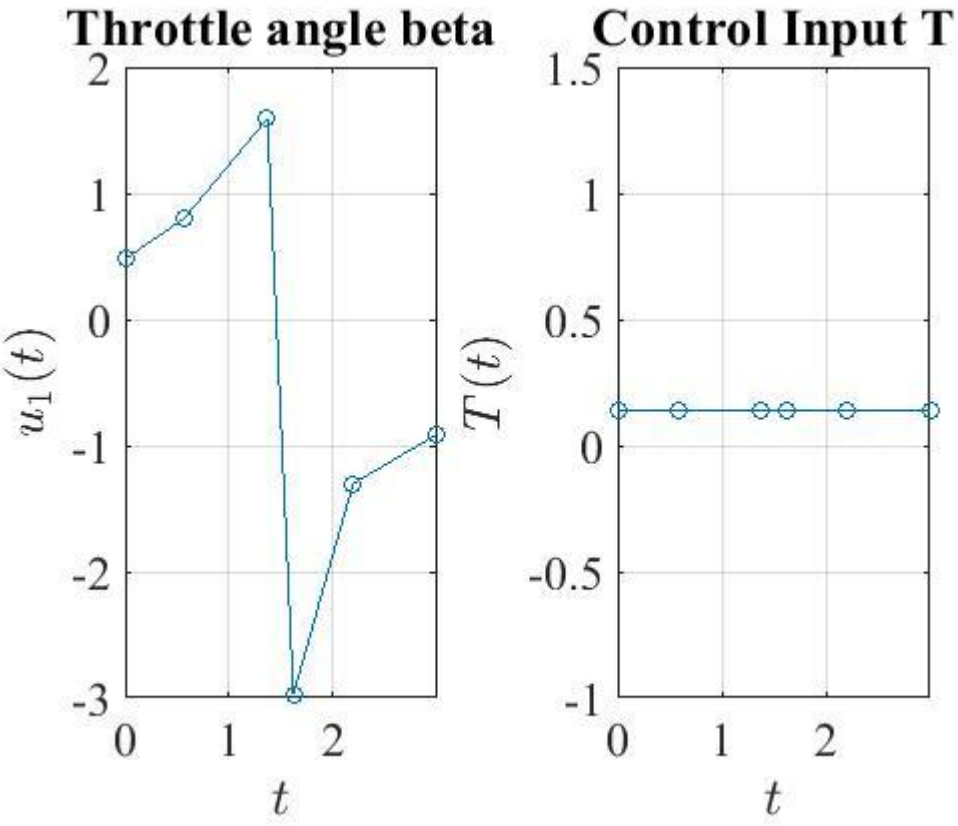


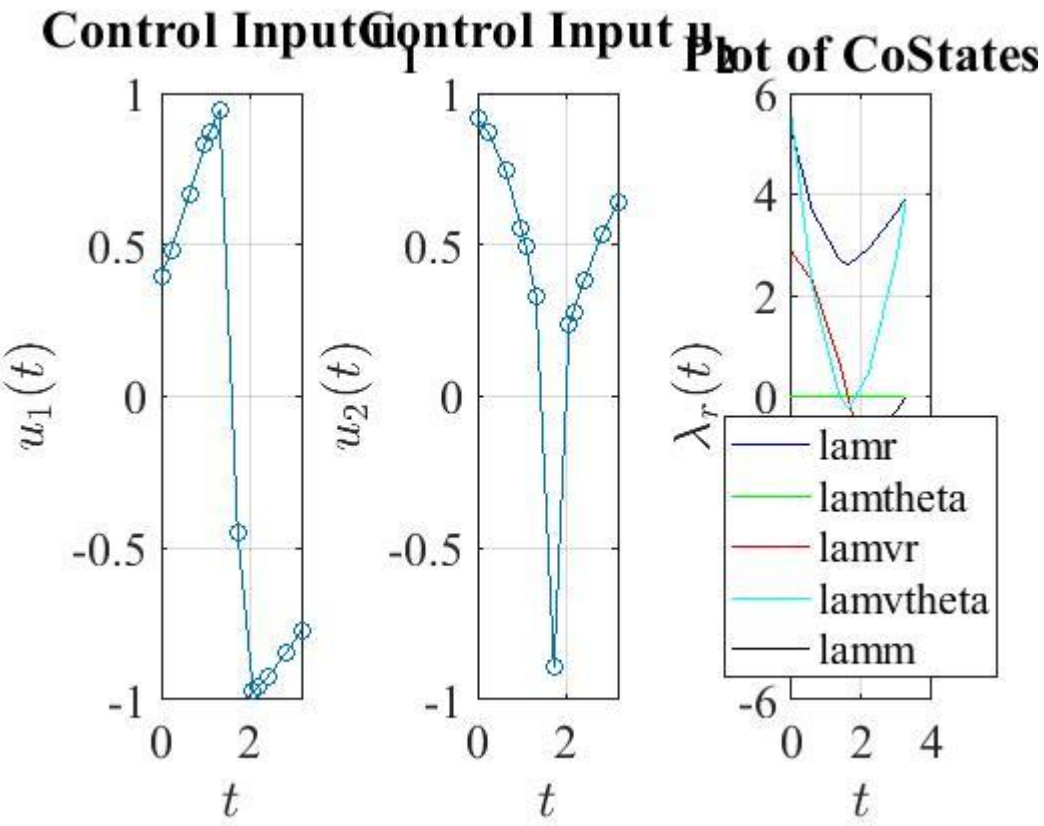
Published with MATLAB® R2021a

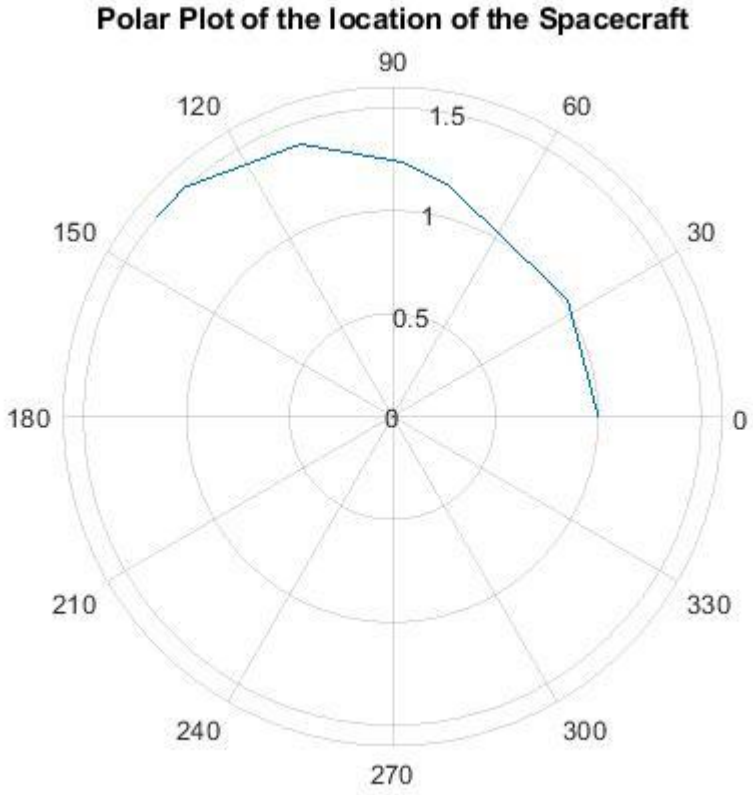
Objective – To minimize final time (M_tf) using beta and T as control.

For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 2

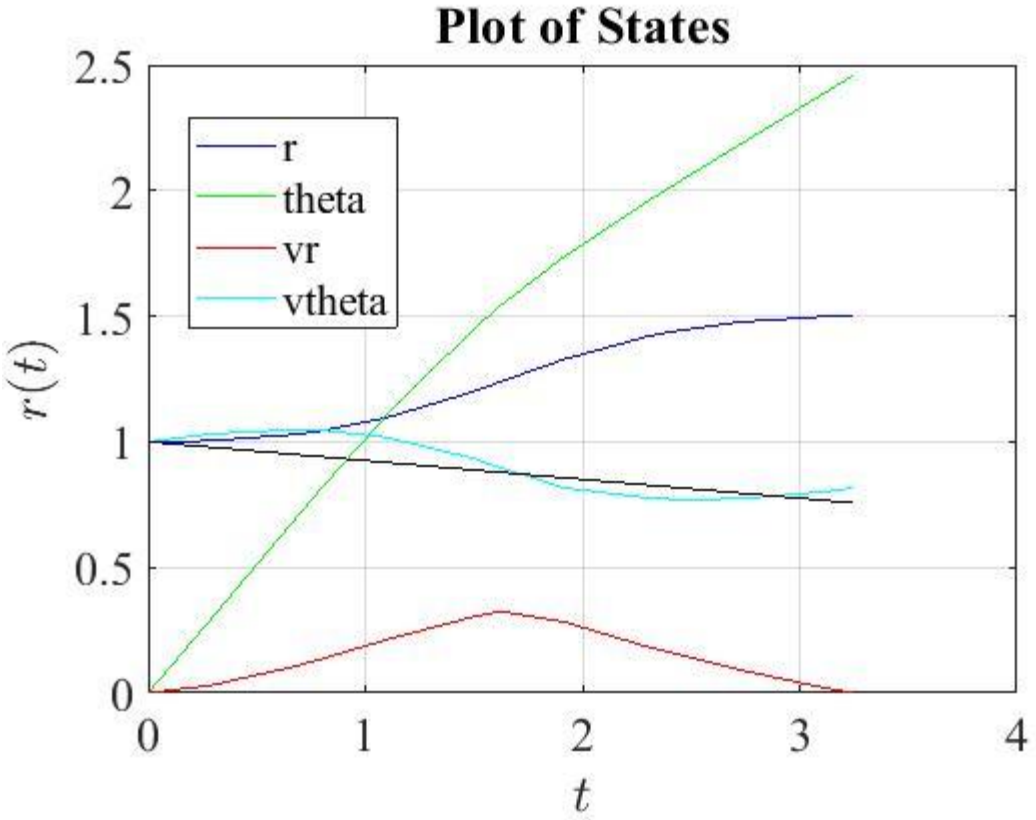


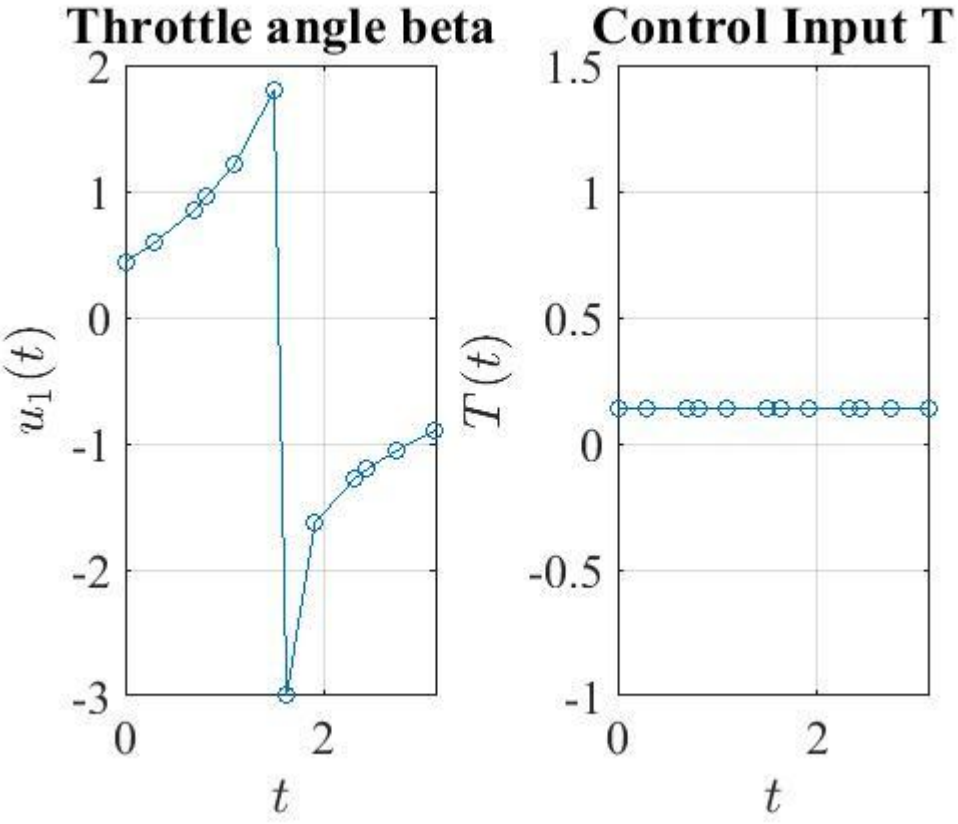


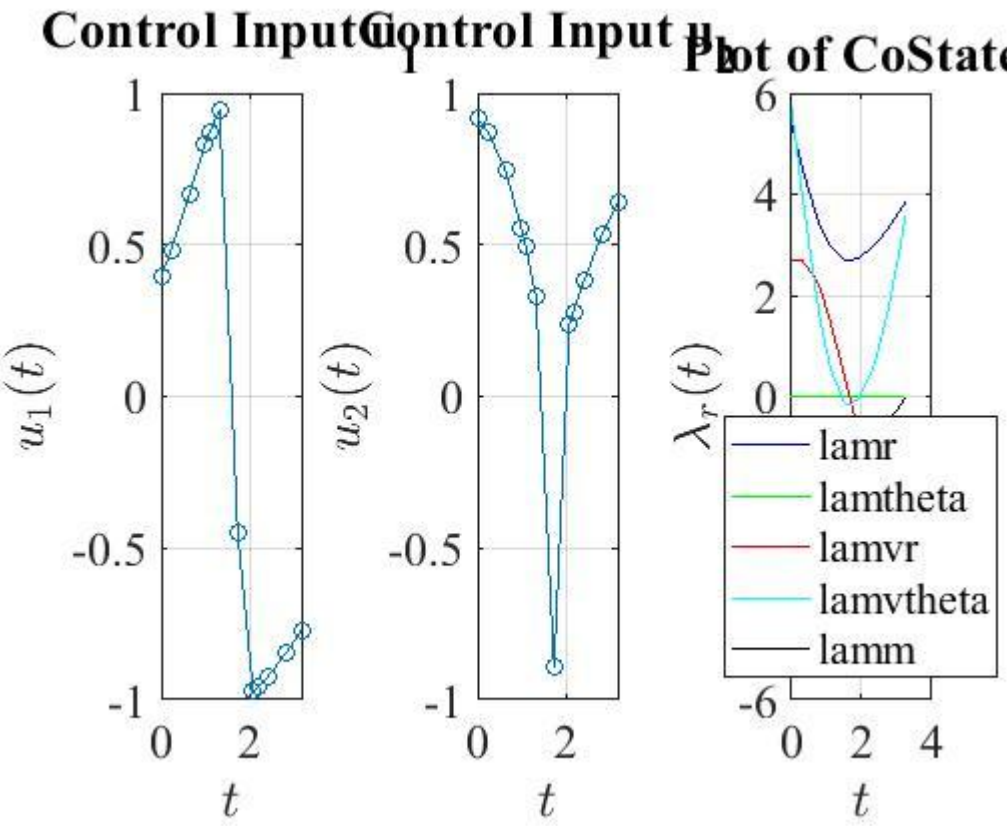


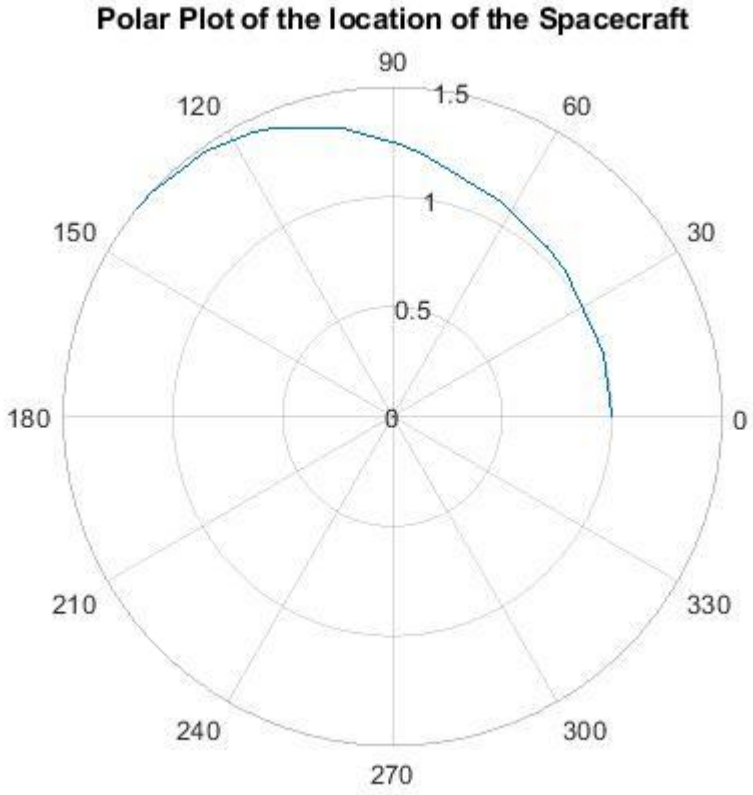


For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 4



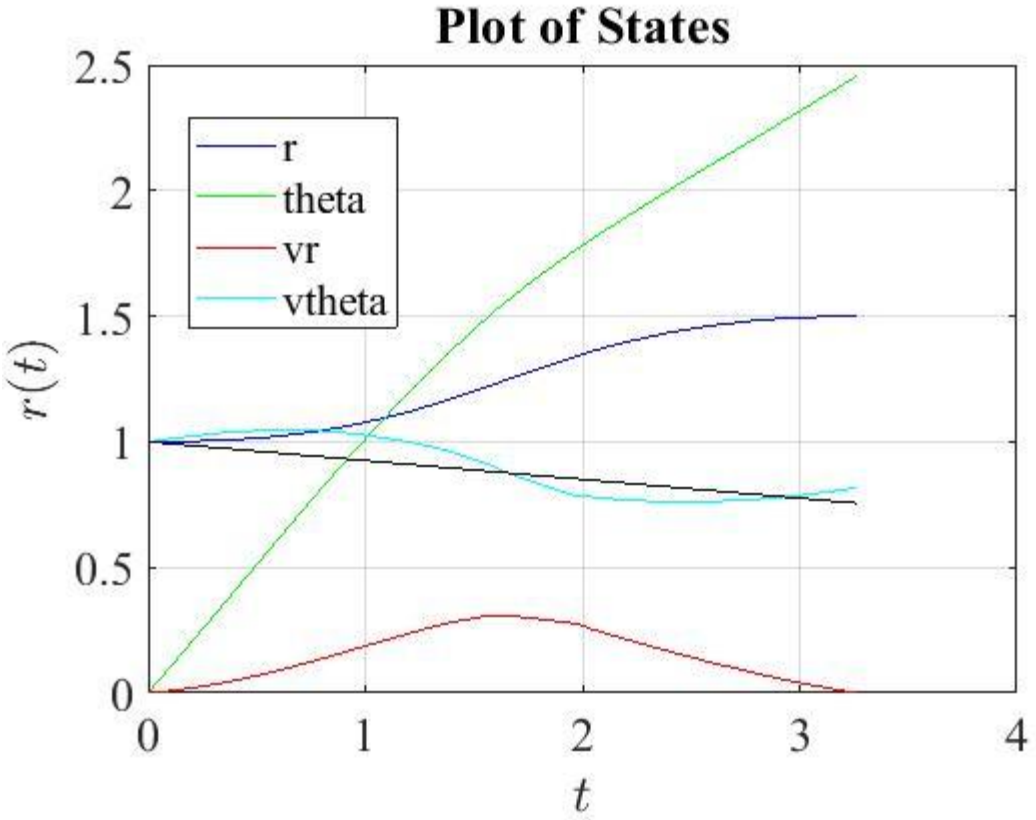


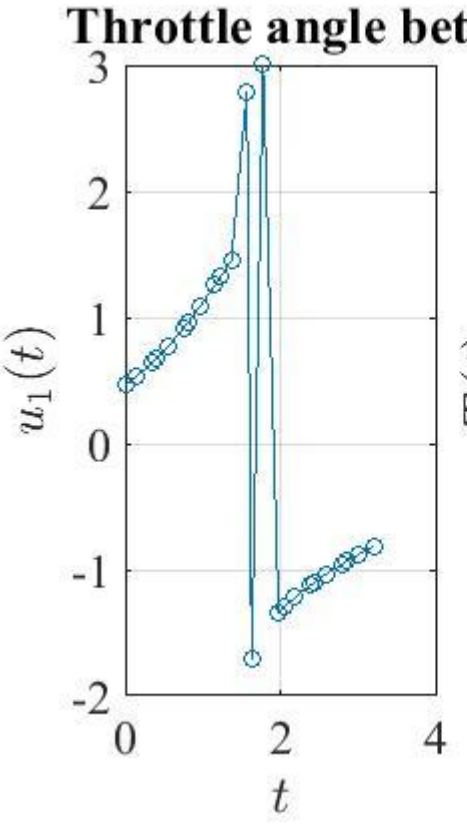


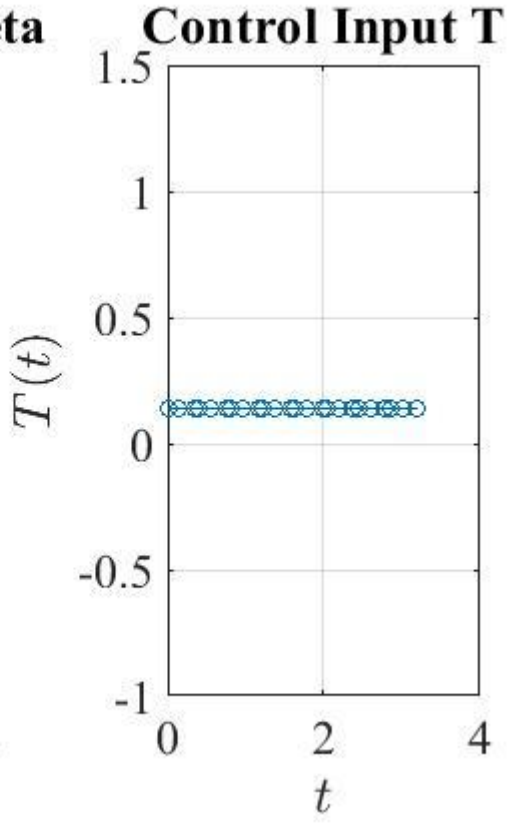


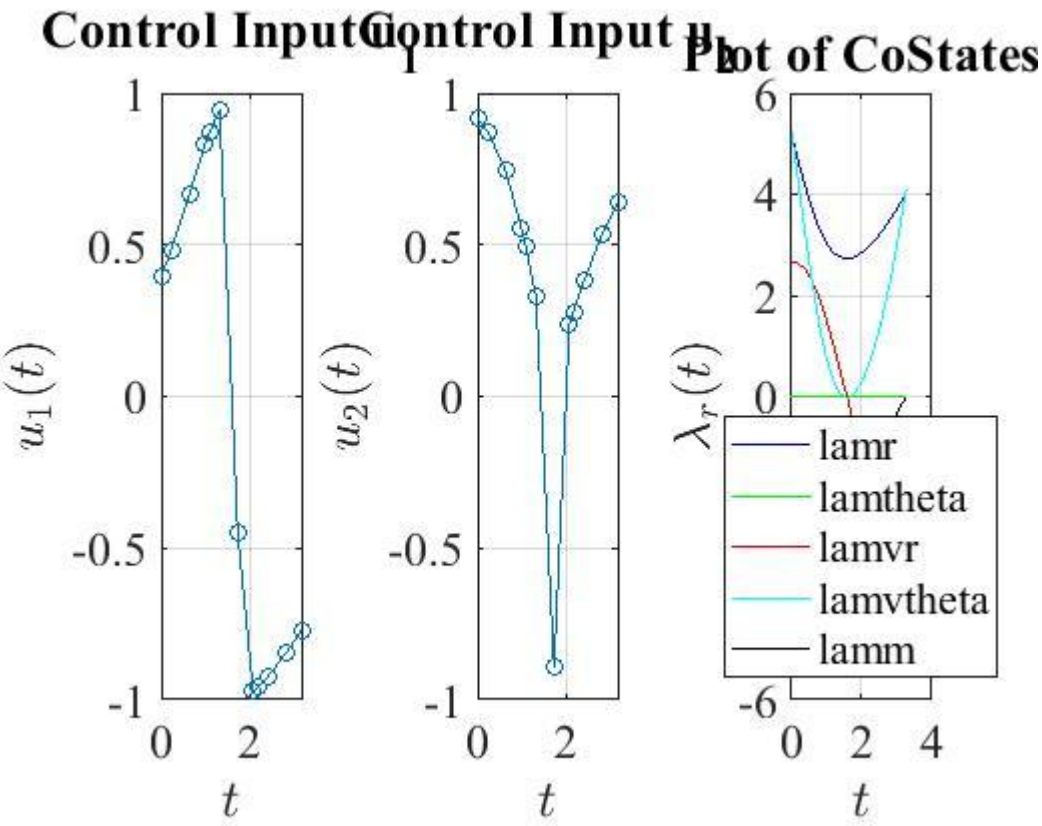
Published with MATLAB® R2021a

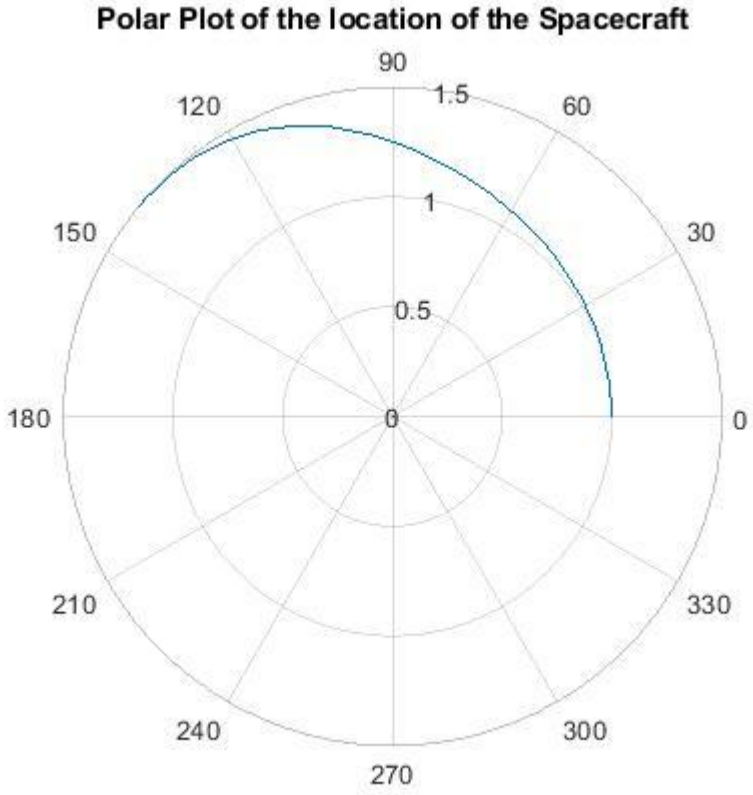
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 8





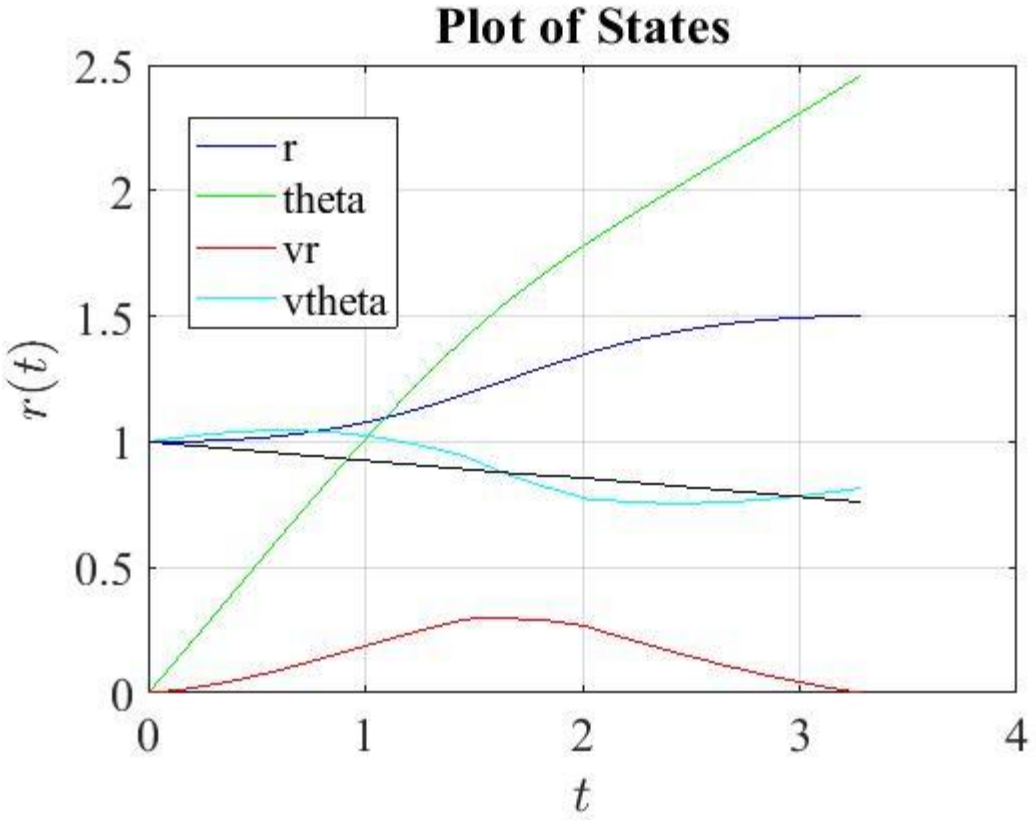


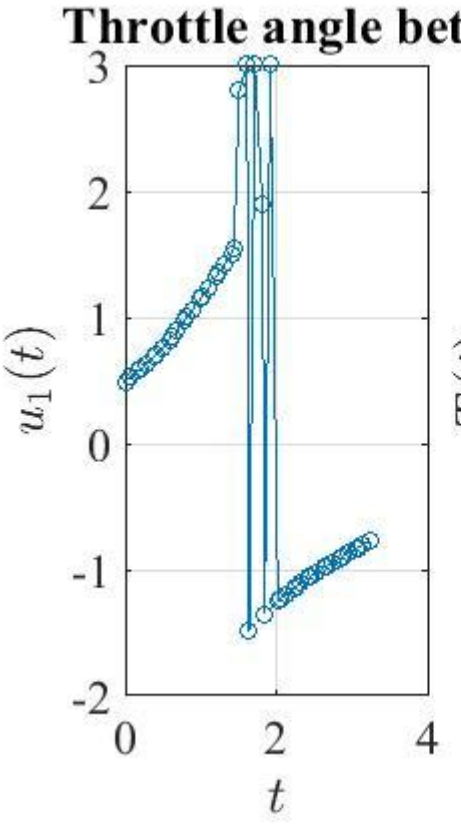


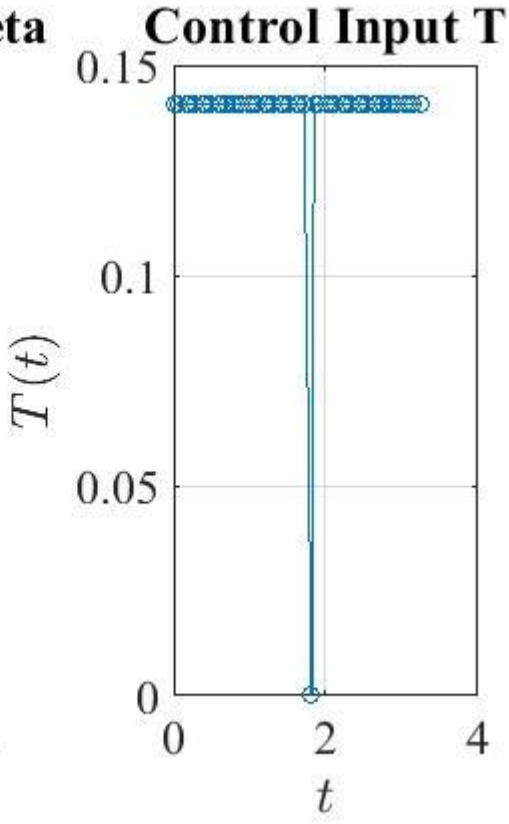


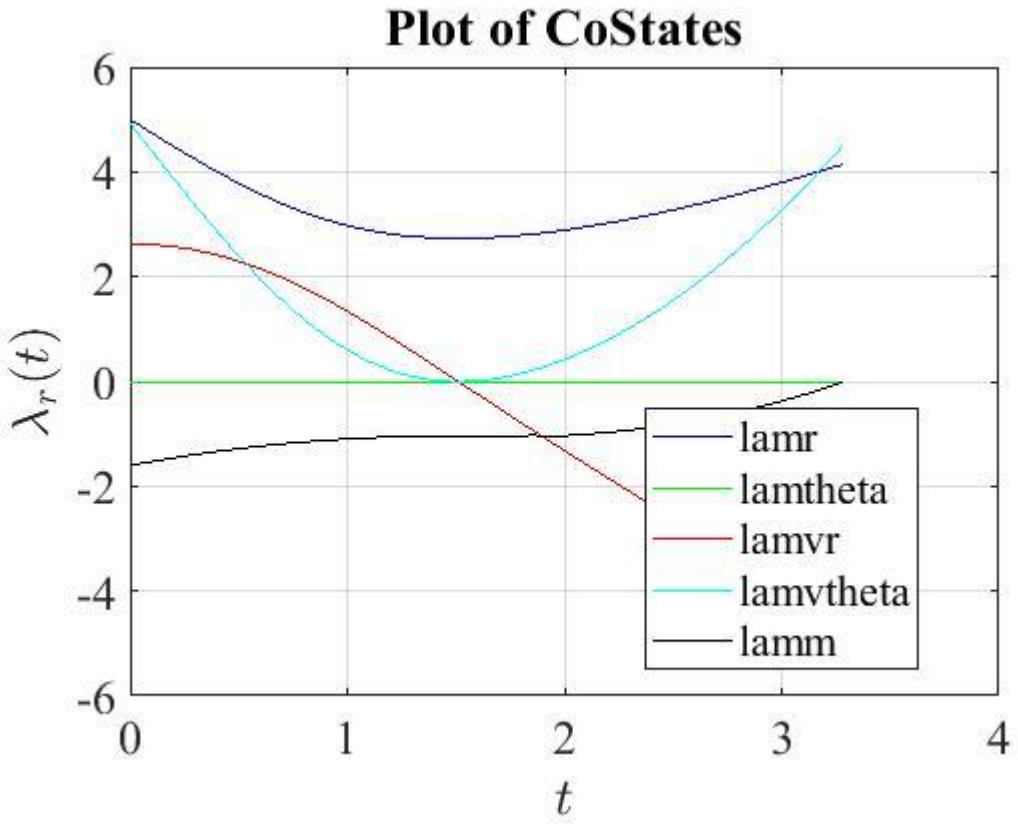
Published with MATLAB® R2021a

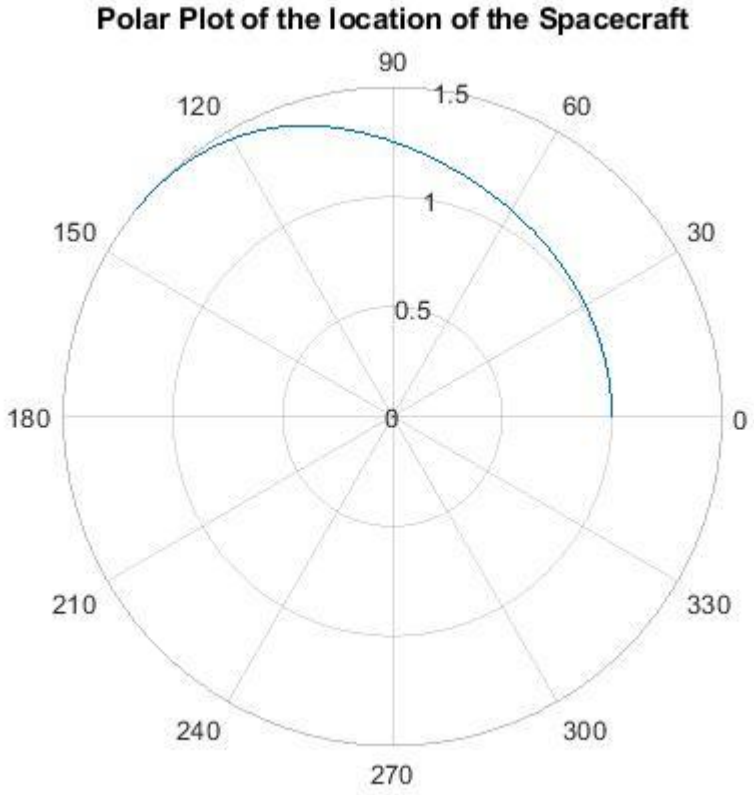
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 16





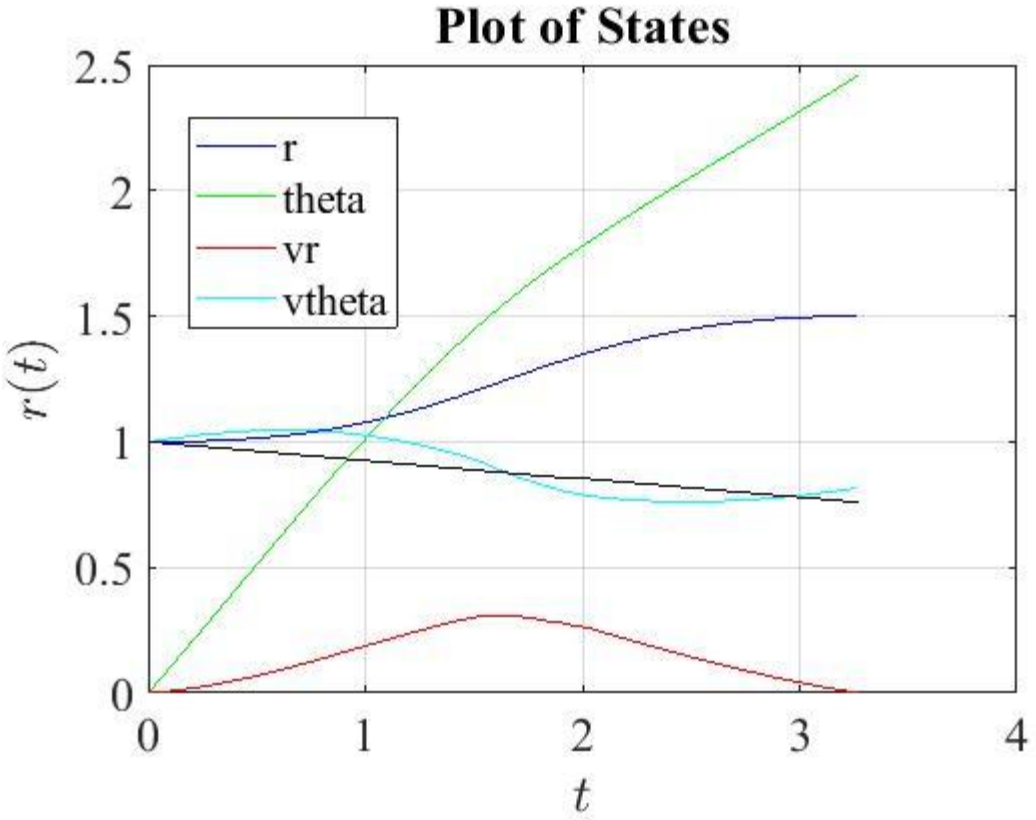


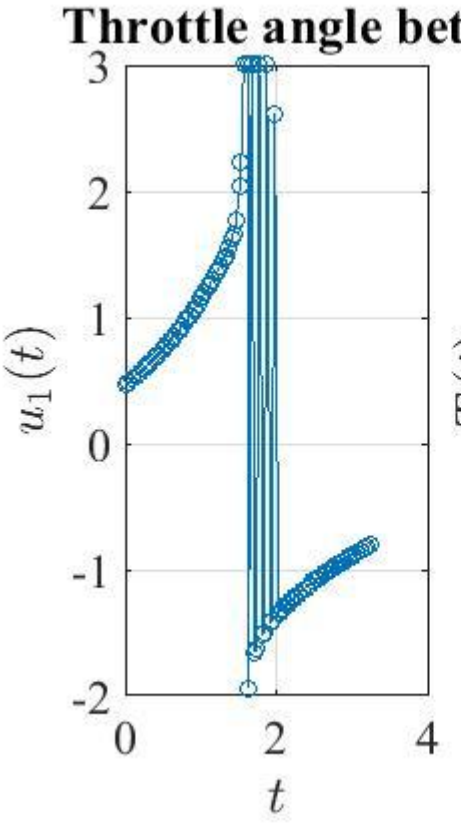


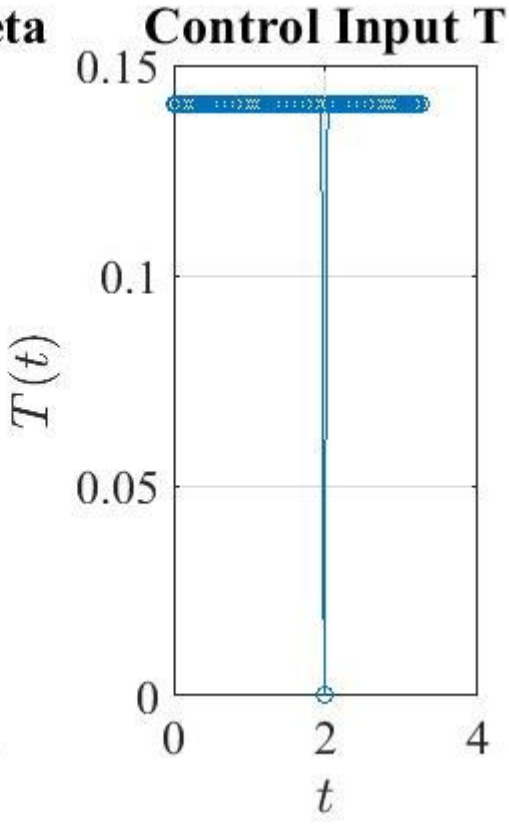


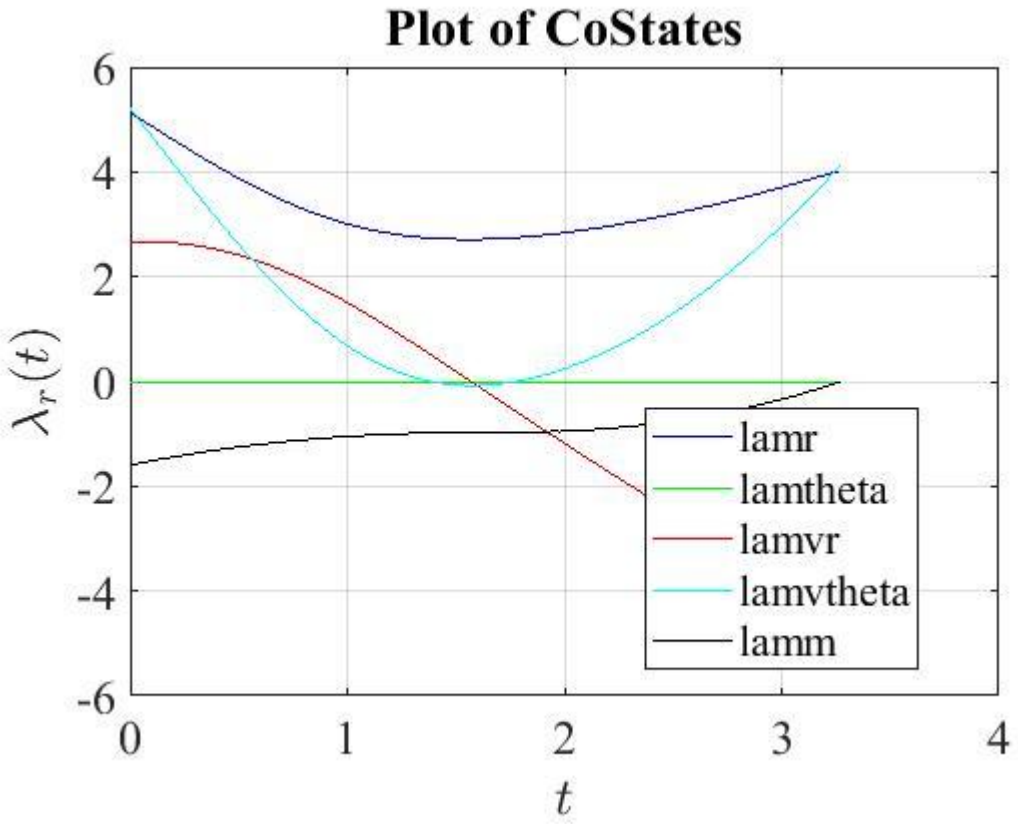
Published with MATLAB® R2021a

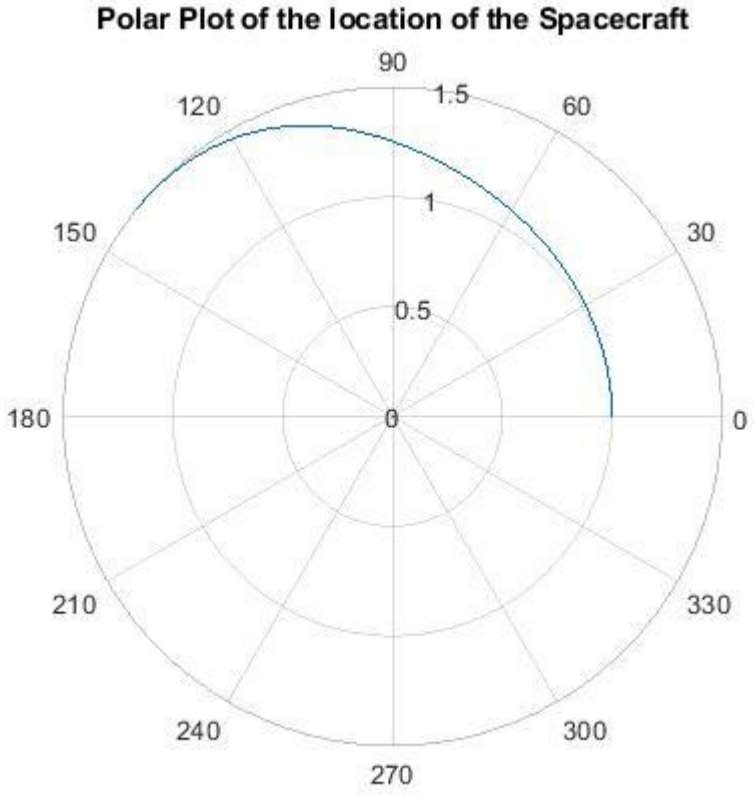
For no of degrees of the polynomial (N) = 3 and no of intervals (K) = 32





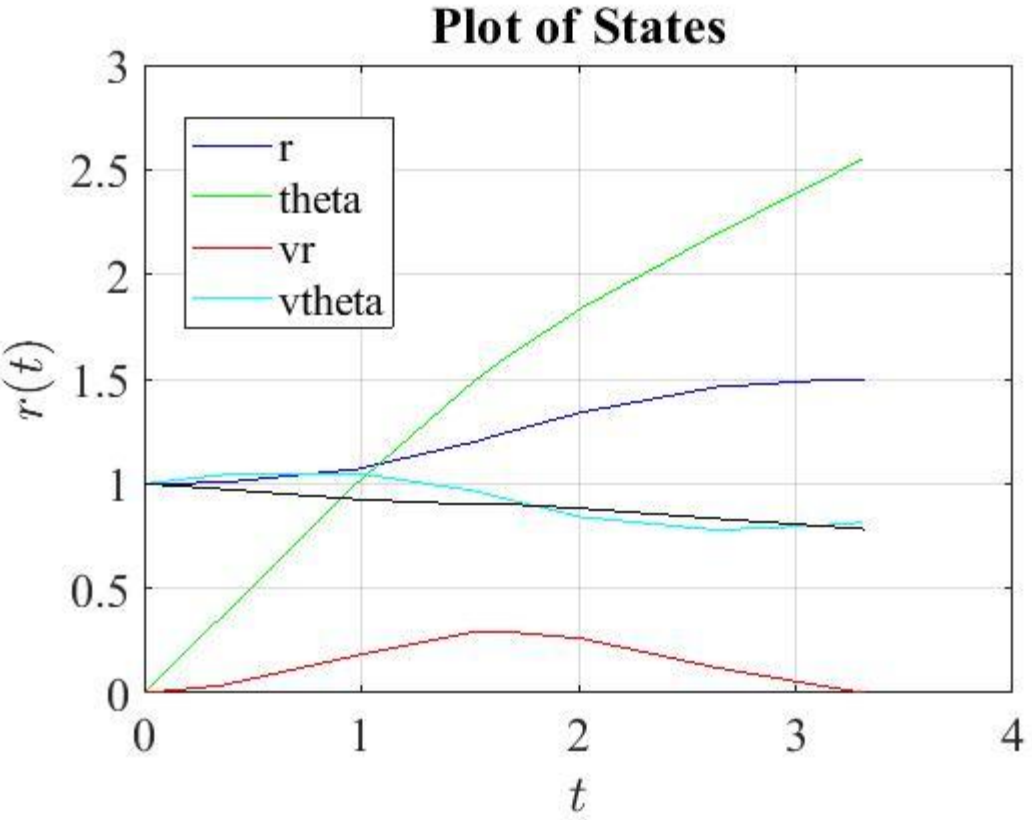


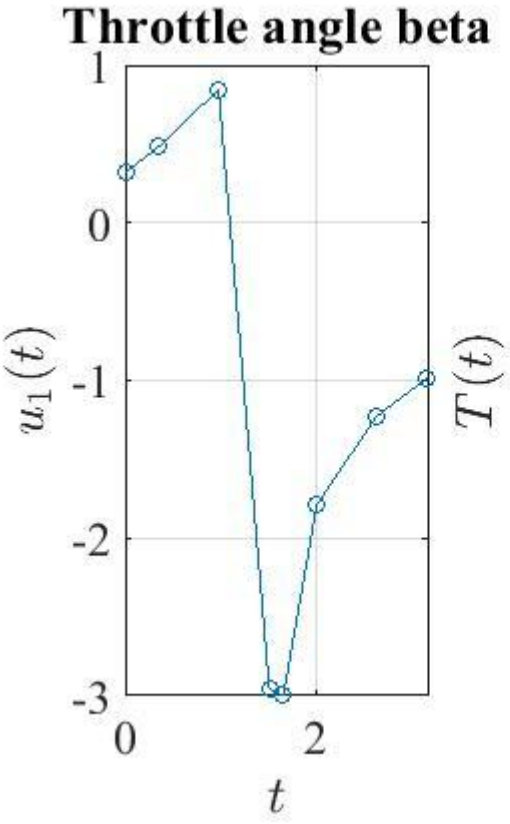


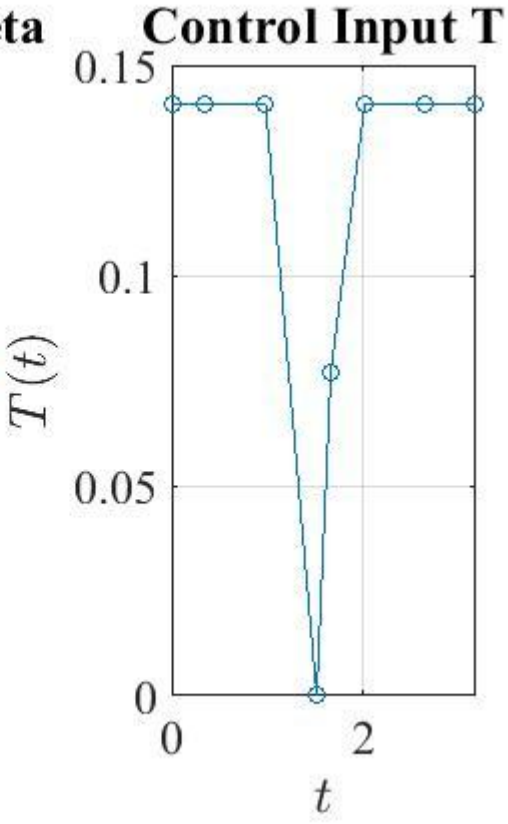


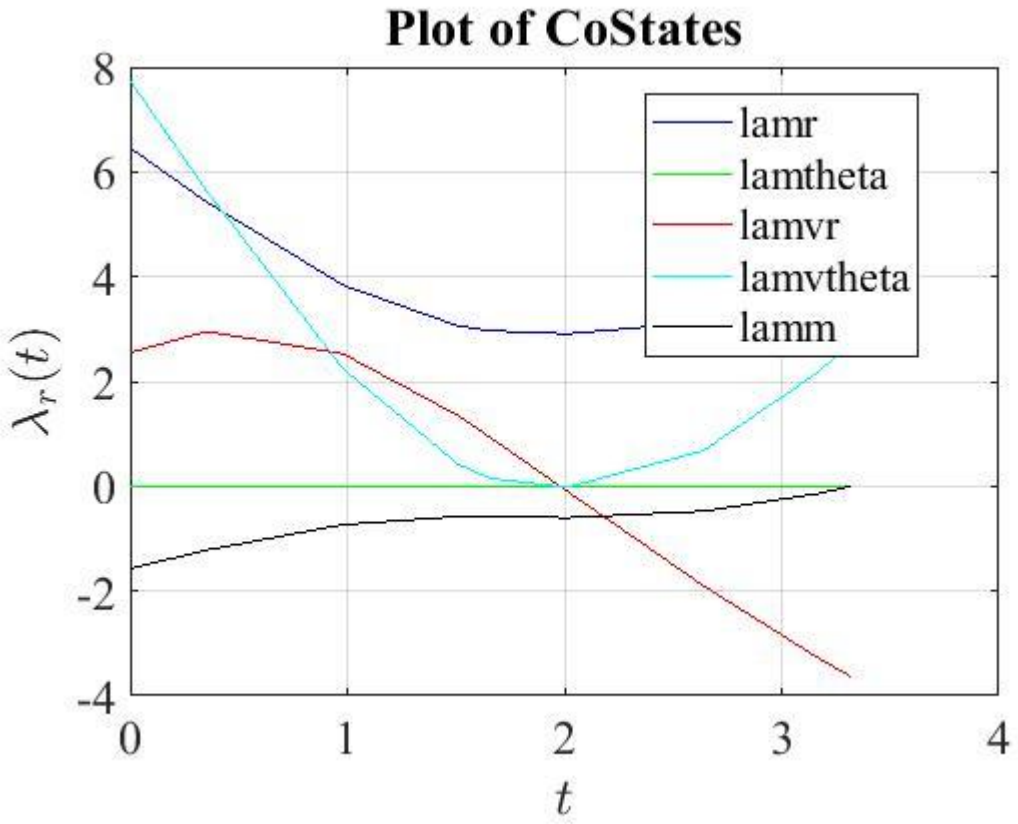
Published with MATLAB® R2021a

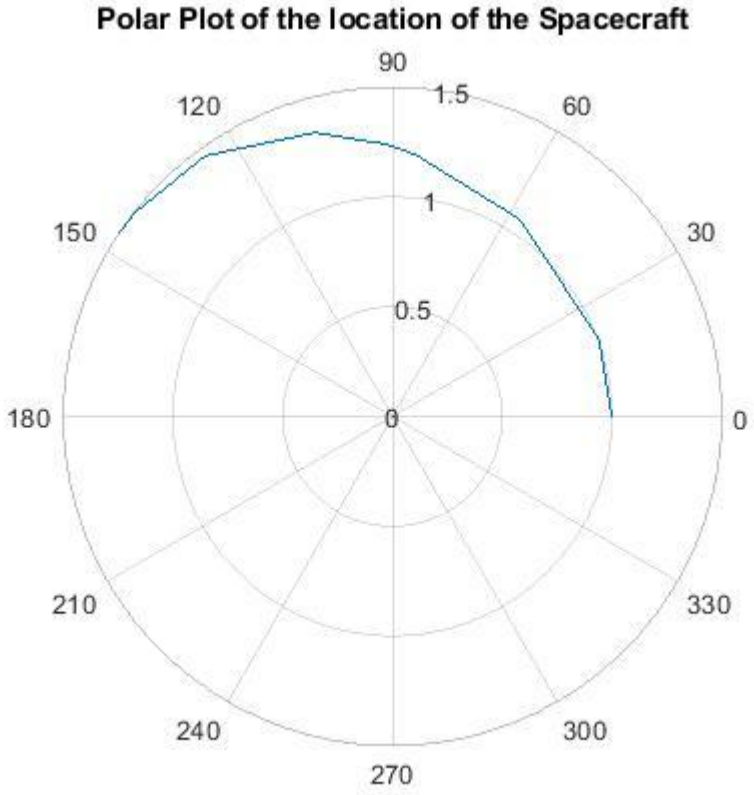
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 2





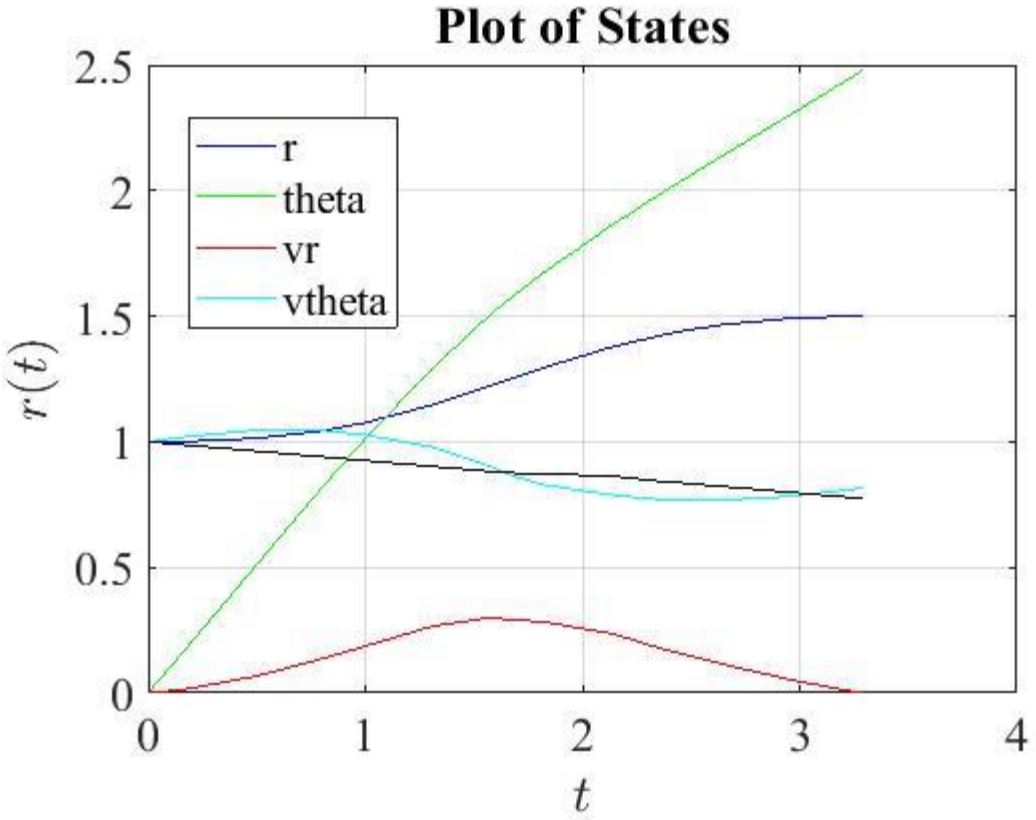


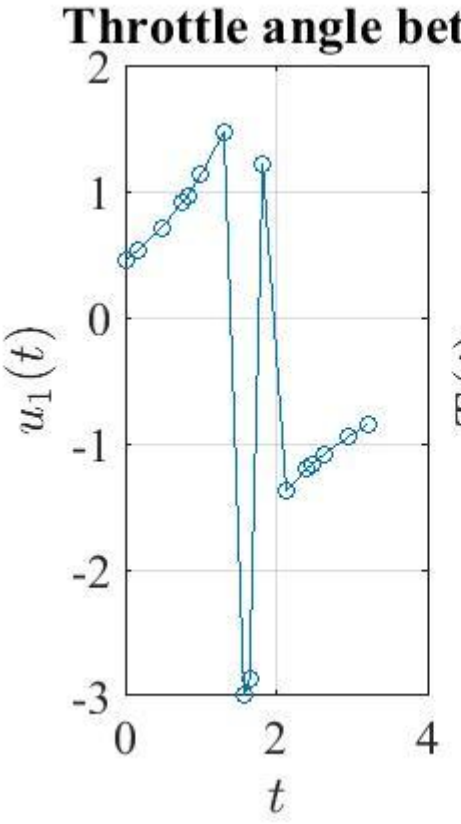


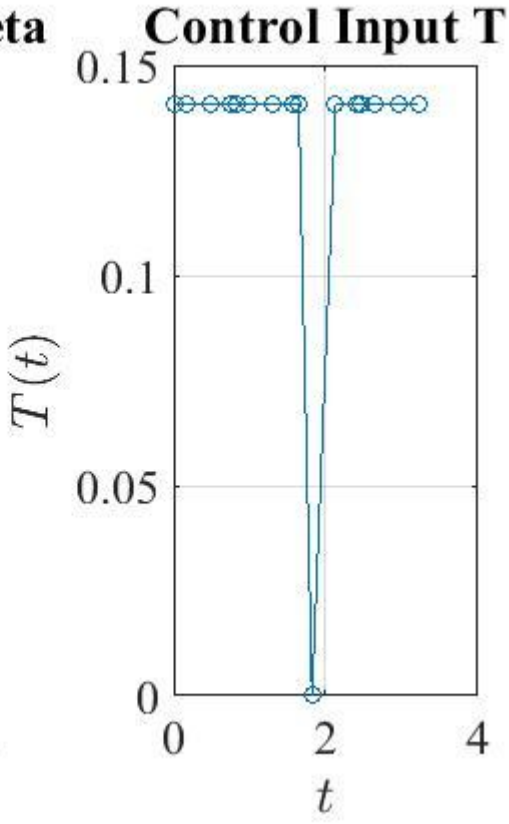


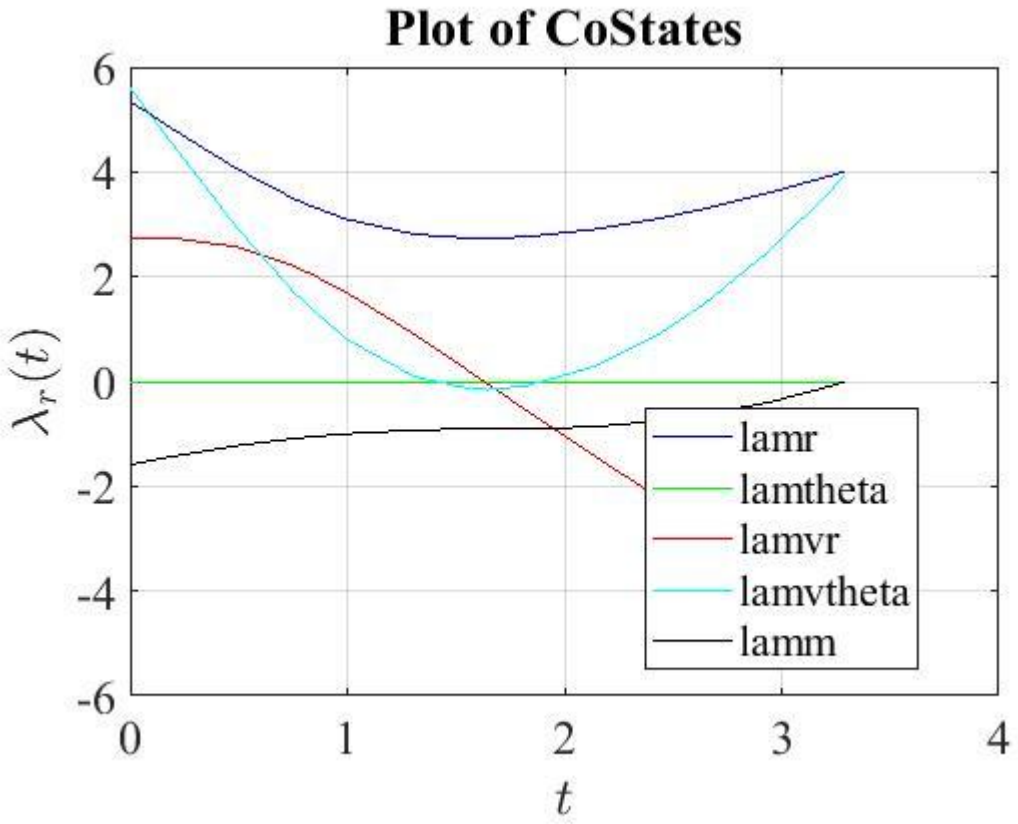
Published with MATLAB® R2021a

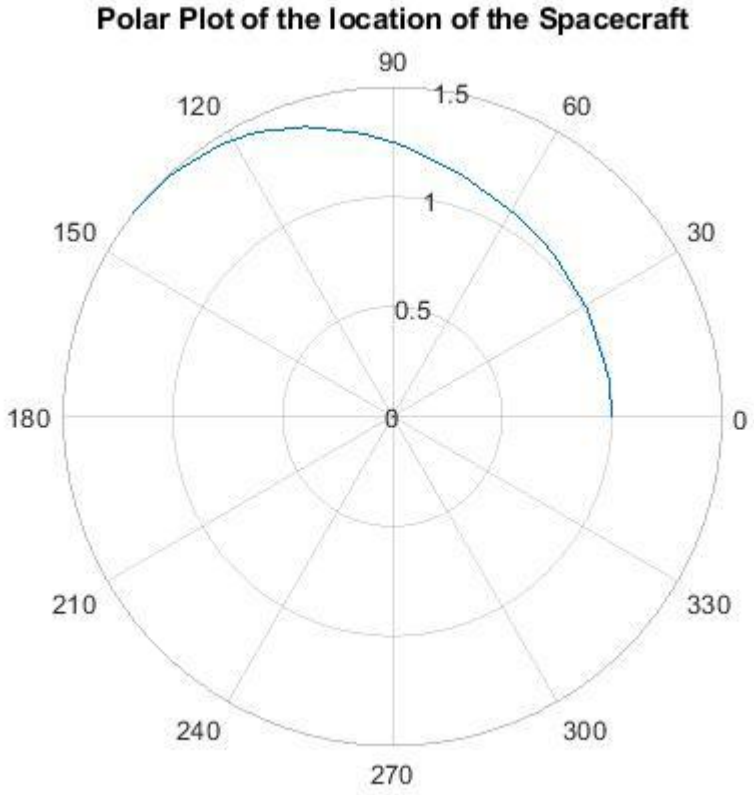
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 4



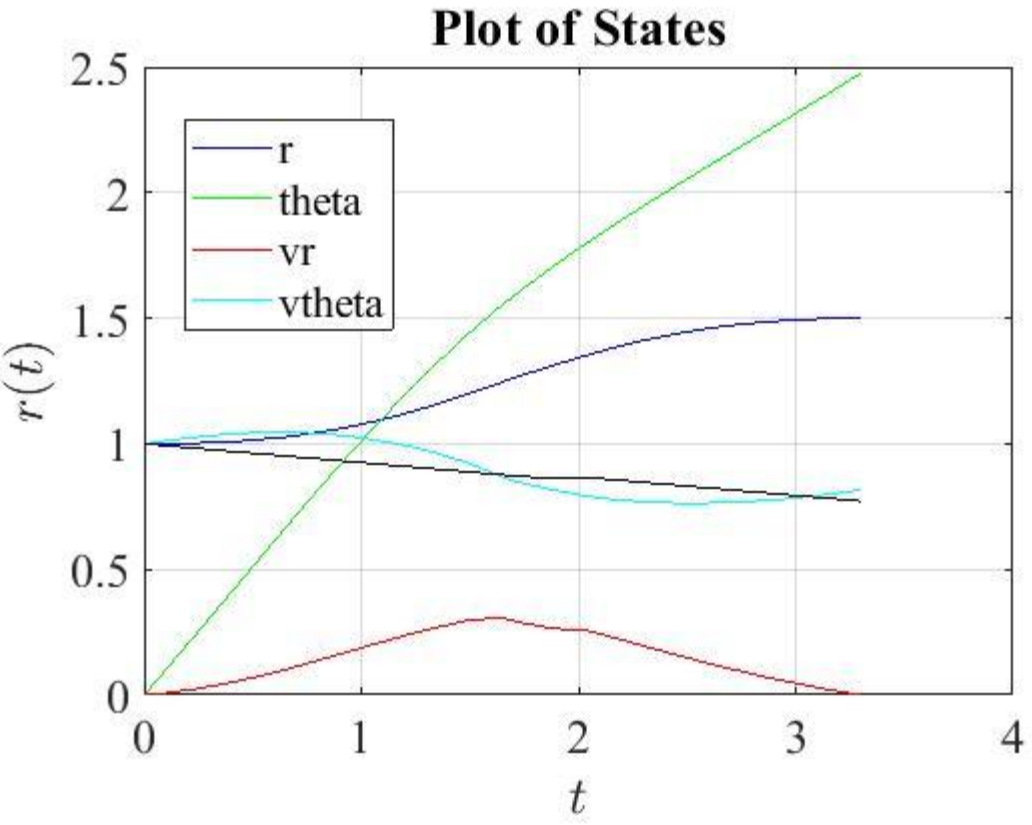


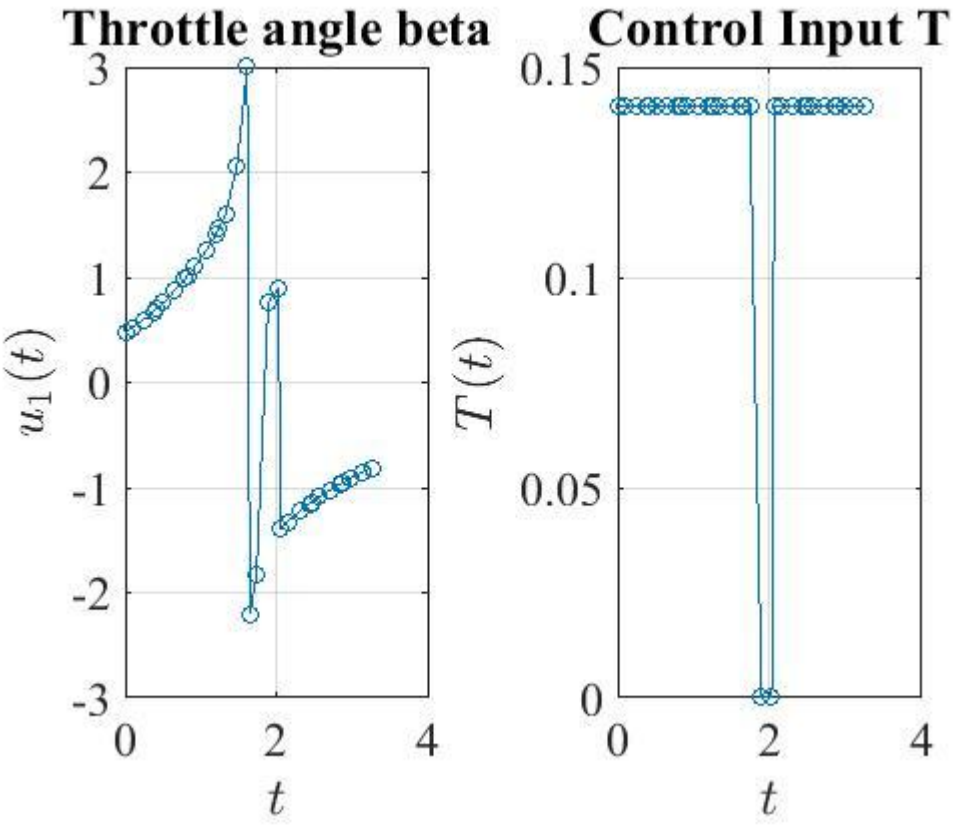


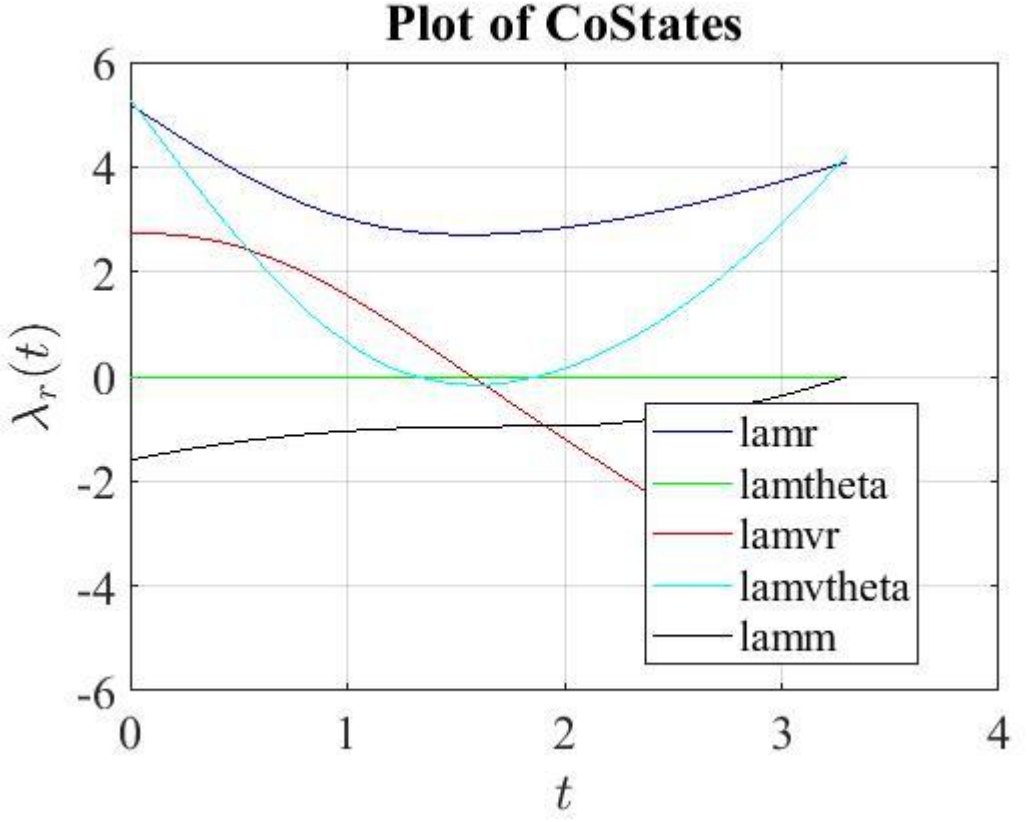


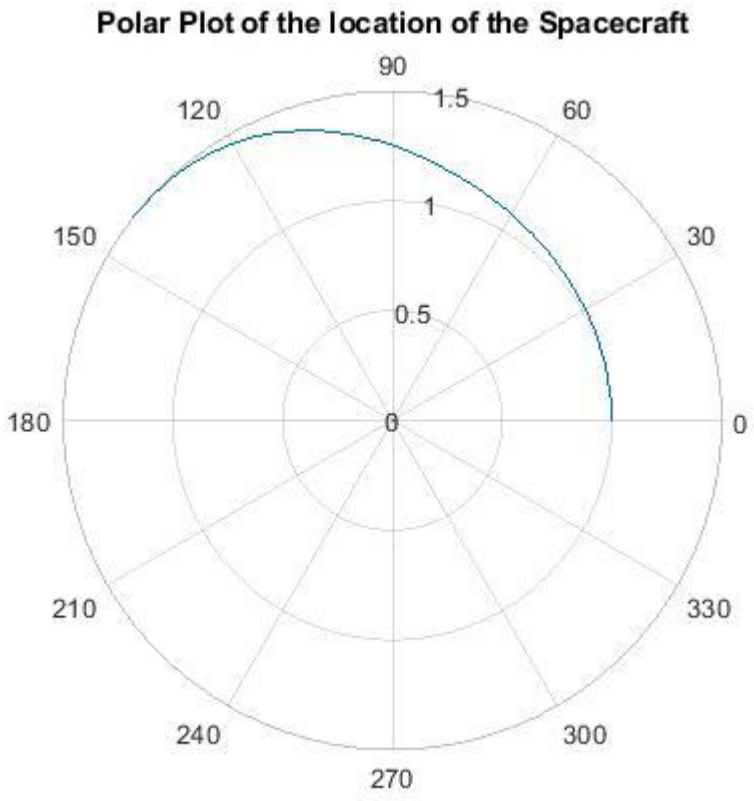


For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 8



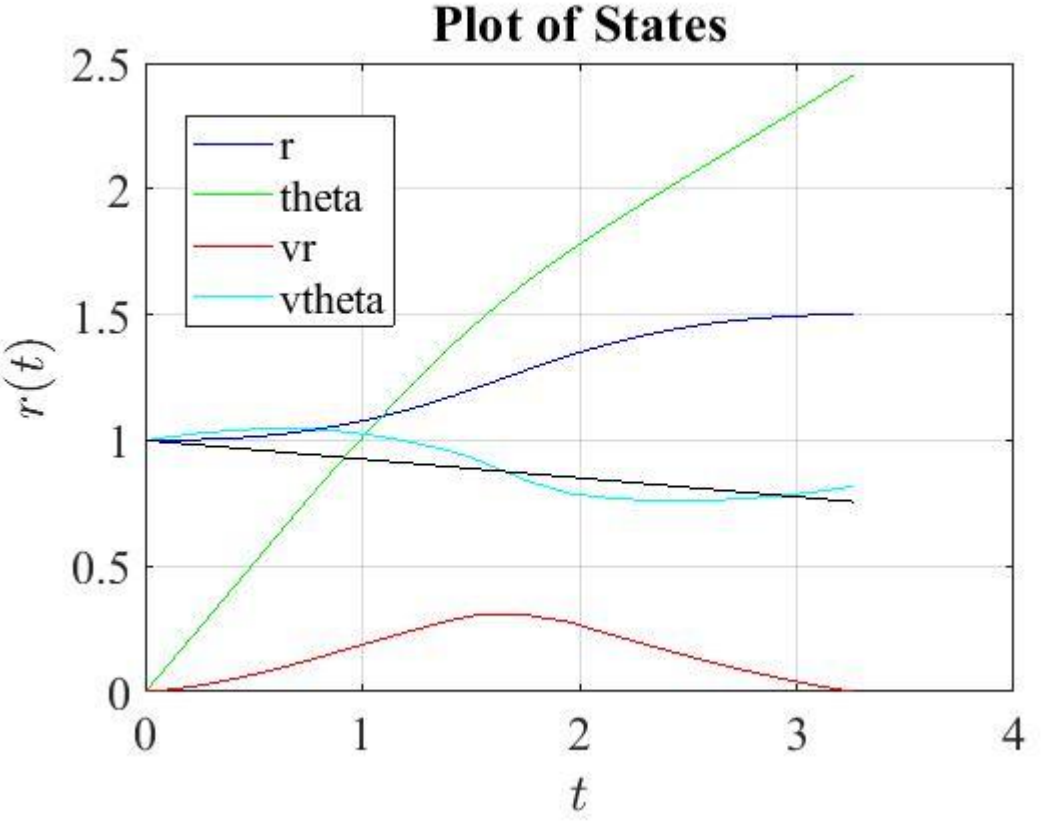

Throttle angle beta
Control Input T
$u_1(t)$
$T(t)$
t
t

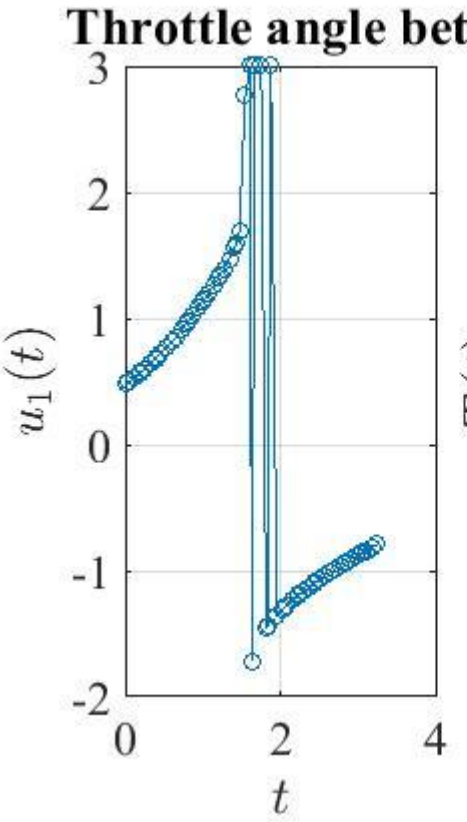

Plot of CoStates
$\lambda_r(t)$
t
lamr
lamtheta
lamvr
lamvtheta
lamm

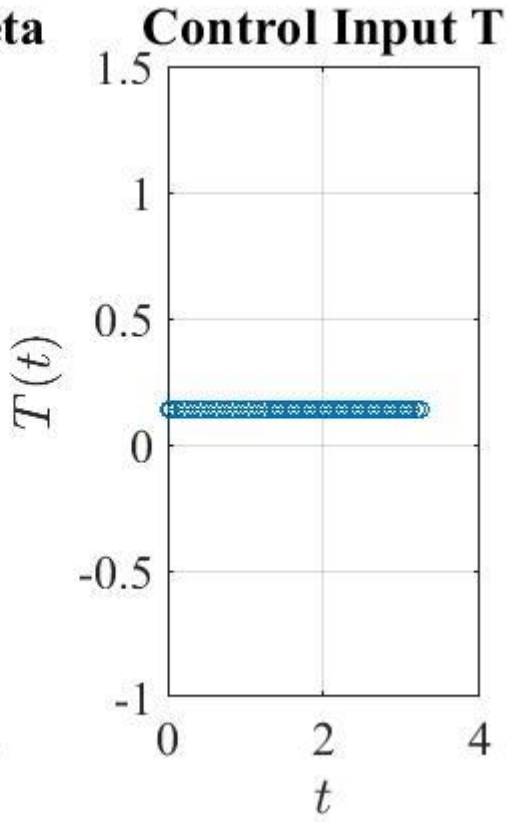


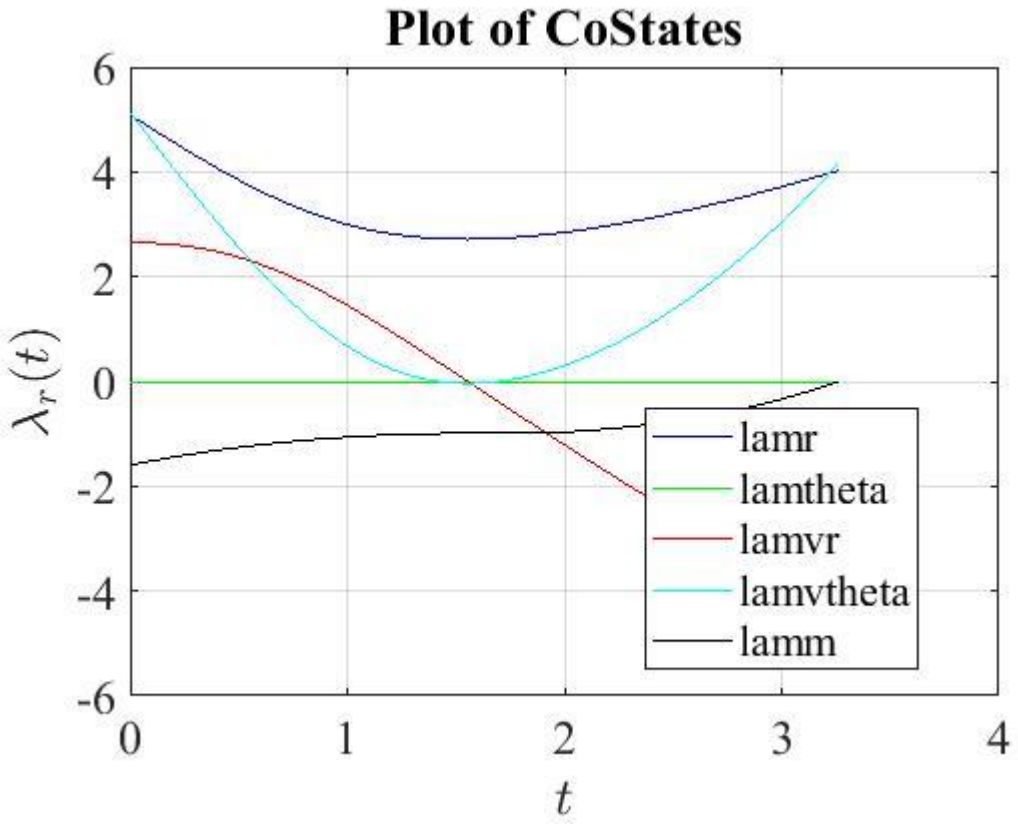
Published with MATLAB® R2021a

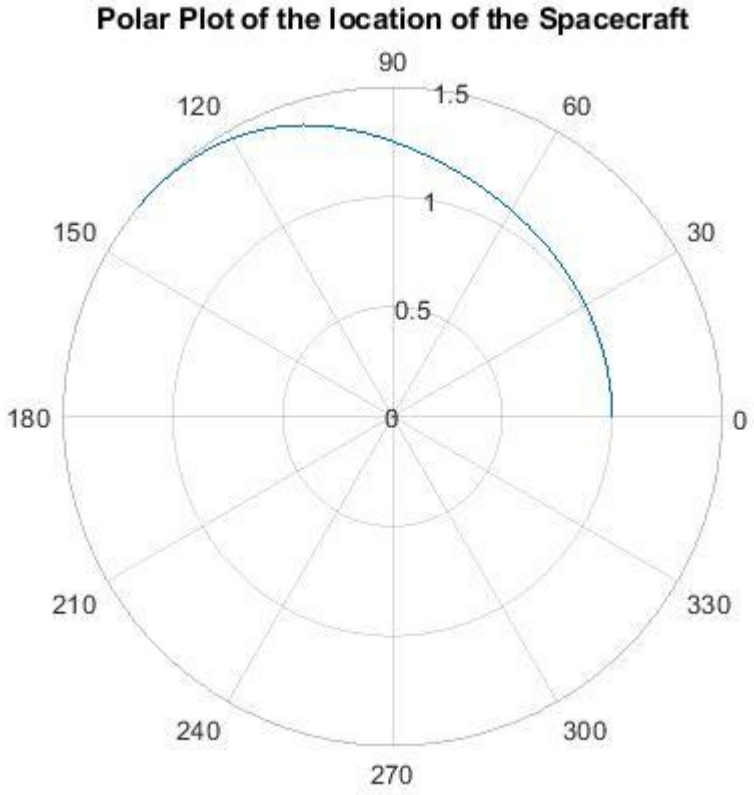
For no of degrees of the polynomial (N) = 4 and no of intervals (K) = 16


Plot of States
r
theta
vr
vtheta
$r(t)$
t


Throttle angle beta
$u_1(t)$
t


Control Input T
$T(t)$
t





Published with MATLAB® R2021a

CODE –

(1) For the objective to maximize mass at final time (M_tf) using U1, U2 and T as control inputs

```
% ------------------------------------------------%
%            Orbit-Transfer Problem                %
% ------------------------------------------------%
% Solve the following optimal control problem:     %
% Minimize t_f                                     %
% subject to the differential equation constraints %
%   dr/dt = v_r                                    %
%   d\theta/dt = v_\theta/r                        %
%   dv_r/dt = v_\theta^2/r-\mu/r^2+a u_1           %
%   dv_\theta/dt = -v_rv_\theta/r+a u_2/m          %
%   dm/dt = - a/v_e
% the equality path constraint                     %
%   u_1^2 + u_2^2 = 1                              %
% and the boundary conditions                      %
%   r(0) = 1                                       %
%   \theta(0) = 0                                  %
%   v_r(0) = 0                                     %
%   v_\theta(0) = sqrt(mu/r(0))                    %
%   m(0) = 1                                       %
%   r(t_f) = 1.5                                   %
%   v_r(t_f) = 0                                   %
%   v_\theta(t_f) = sqrt(mu/r(t_f)                 %
% ------------------------------------------------ %
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------ %
global igrid CONSTANTS psStuff nstates ncontrols npaths
% ------------------------------------------------ %
% END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------ %

CONSTANTS.MU = 1;
CONSTANTS.m0 = 1;
CONSTANTS.mdot = 0.0749;
CONSTANTS.ve = 1.8758344;
nstates = 5;
ncontrols = 3;
npaths = 1;

% Bounds on State and Control
r0 = 1; theta0 = 0; vr0 = 0; vtheta0 = sqrt(CONSTANTS.MU/r0); m0 = 1;
rf = 1.5; vrf = 0;  vthetaf = sqrt(CONSTANTS.MU/rf);
rmin = 0.5; rmax = 2;
thetamin = 0; thetamax = 4*pi;
vrmin = -10; vrmax = 10;
vthetamin = -10; vthetamax = 10;
mmin = 0.5; mmax = m0;
u1min = -10; u1max = 10;
```

```
u2min = -10; u2max = 10;
Tmin = 0; Tmax = 0.1405;
t0min = 0; t0max = 0;
tfmin = 0; tfmax = 50;

%---------------------------------------------------------------------%
%      Compute Points, Weights, and Differentiation Matrix            %
%---------------------------------------------------------------------%
%---------------------------------------------------------------------%
% Choose Polynomial Degree and Number of Mesh Intervals               %
% numIntervals = 1 ===> p-method                                      %
% numIntervals > 1 ===> h-method                                      %
%---------------------------------------------------------------------%
N = 3;
numIntervals = 2;
%---------------------------------------------------------------------%
% 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 NLP variables.
zrmin = rmin*ones(length(tau),1);
zrmax = rmax*ones(length(tau),1);
zrmin(1) = r0; zrmax(1) = r0;
zrmin(end) = rf; zrmax(end) = rf;

zthetamin = thetamin*ones(length(tau),1);
zthetamax = thetamax*ones(length(tau),1);
zthetamin(1) = theta0; zthetamax(1) = theta0;

zvrmin = vrmin*ones(length(tau),1);
zvrmax = vrmax*ones(length(tau),1);
zvrmin(1) = vr0; zvrmax(1) = vr0;
zvrmin(end) = vrf; zvrmax(end) = vrf;

zvthetamin = vthetamin*ones(length(tau),1);
zvthetamax = vthetamax*ones(length(tau),1);
zvthetamin(1) = vtheta0; zvthetamax(1) = vtheta0;
zvthetamin(end) = vthetaf; zvthetamax(end) = vthetaf;

zmmin = mmin*ones(length(tau),1);
zmmax = mmax*ones(length(tau),1);
zmmin(1) = mmax; zmmax(1) = mmax;

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);

zTmin = Tmin*ones(length(tau)-1,1);
zTmax = Tmax*ones(length(tau)-1,1);

zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zu2min; zTmin; tOmin; tfmin];
zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zu2max; zTmax; tOmax; tfmax];

% Set the bounds on the NLP constraints
% There are NSTATES sets of defect constraints.
defectMin = zeros(nstates*(length(tau)-1),1);
defectMax = zeros(nstates*(length(tau)-1),1);
% There is one path constraint
pathMin = ones(length(tau)-1,1); pathMax = ones(length(tau)-1,1);
% There is one nonlinear event constraint
bcMin = 0; bcMax = 0;
objMin = -inf; objMax = inf;
Fmin = [objMin; defectMin; pathMin];
Fmax = [objMax; defectMax; pathMax];

% Supply an initial guess
rguess = linspace(r0,rf,NLGR+1).';
thetaguess = linspace(theta0,theta0,NLGR+1).';
vrguess = linspace(vr0,vrf,NLGR+1).';
vthetaguess = linspace(vtheta0,vthetaf,NLGR+1).';
mguess = linspace(mmax,mmin,NLGR+1).';
u1guess = linspace(1,1,NLGR).';
u2guess = linspace(0,0,NLGR).';
Tguess = linspace(Tmax,Tmax,NLGR).';
tOguess = 0;
tfguess = 3.2481;
z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;u2guess;Tguess;tOguess;tfguess];

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

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

%-------------------------------------------------------------------%
% set IPOPT callback functions
%-------------------------------------------------------------------%
funcs.objective   = @(Z)OrbitTransferObj(Z);
funcs.gradient    = @(Z)OrbitTransferGrd(Z);
```

```
funcs.constraints = @(Z)OrbitTransferCon(Z);
funcs.jacobian    = @(Z)OrbitTransferJac(Z);
funcs.jacobianstructure = @()OrbitTransferJacPat(S_jac);
options.ipopt.hessian_approximation = 'limited-memory';

%-----------------------------------------------------------------%
% Set IPOPT Options %
%-----------------------------------------------------------------%
options.ipopt.tol = 1e-8;
options.ipopt.linear_solver = 'ma57';
options.ipopt.max_iter = 2000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['OrbitTransfer','IPOPTinfo.txt']; % print output file
options.ipopt.print_level = 5; % set print level default

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

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

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

% Extract the state and control from the decision vector z.
% Remember that the state is approximated at the LGR points
% plus the final point, while the control is only approximated
% at only the LGR points.
r = z(1:NLGR+1);
theta = z(NLGR+2:2*(NLGR+1));
vr = z(2*(NLGR+1)+1:3*(NLGR+1));
vtheta = z(3*(NLGR+1)+1:4*(NLGR+1));
m = z(4*(NLGR+1)+1:5*(NLGR+1));
u1 = z(5*(NLGR+1)+1:5*(NLGR+1)+NLGR);
u2 = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
T = z(5*(NLGR+1)+2*NLGR+1:5*(NLGR+1)+3*NLGR);
beta = 180/pi*atan2(u1,u2);
t0 = z(end-1);
tf = z(end);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);
%-----------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                  %
% the defect constraints.                                         %
%-----------------------------------------------------------------%
multipliersDefects = Fmul(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];
lamr = costate(:,1); lamtheta = costate(:,2);
lamvr = costate(:,3); lamvtheta = costate(:,4);
lamm = costate(:,5);

%-------------%
% Plot Results
%-------------%
figure(1)
plot(t,r,'r',t,theta,'b',t,vr,'m',t,vtheta,'k', t, m,'g');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('states','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
title('Plot of States');
set(gca,'FontName','Times','FontSize',18);
grid on;

figure(2);
plot(tLGR,beta,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$\beta(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
title('Throttle angle beta');
grid on;

figure(3);
subplot(1,3,1);
plot(tLGR,u1,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$u_1(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
grid on;

subplot(1,3,2);
plot(tLGR,u2,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$u_2(t)$','Interpreter','LaTeX');
```

```
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
grid on;

subplot(1,3,3);
plot(tLGR,T,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$T(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
grid on;

figure(4);
plot(t,lamr,'r',t,lamtheta,'b',t,lamvr,'m',t,lamvtheta,'k', t, lamm,'g');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('Costates','Interpreter','LaTeX');
title('Plot of CoStates');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
grid on;

figure (5);
polarplot(theta,r);
title('Position of the Spacecraft');

%OrbitTransferFun.m
function C = OrbitTransferFun(z)

%------------------------------------------------------------------%
% Objective and constraint functions for the orbit-transfer        %
% 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 npaths CONSTANTS                  %
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE!         %
%------------------------------------------------------------------%

%------------------------------------------------------------------%
%         Extract the constants used in the problem.               %
%------------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot; ve = CONSTANTS.ve;

%------------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%--------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
control = reshape(controlVector,N,ncontrols);

%--------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR. %
%--------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
u2 = control(:,2);
T =control(:,3);

%--------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
%--------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%--------------------------------------------------------------------%
```

```
% The orbit-raising problem contains one nonlinear boundary       %
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%-----------------------------------------------------------------%
rF = stateF(1);
vrF = stateF(3);
vthetaF = stateF(4);
mF = stateF(5);

% a = T./m;

%-----------------------------------------------------------------%
% 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.                              %
%-----------------------------------------------------------------%
rdot = vr;
thetadot = vtheta./r;
vrdot = vtheta.^2./r-MU./r.^2+T.*u1./m;
vthetadot = -vr.*vtheta./r+T.*u2./m;
m_dot = -T./ve;
diffeqRHS = [rdot, thetadot, vrdot, vthetadot, m_dot];

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

%-----------------------------------------------------------------%
% Construct the path constraints at the N LGR points.            %
%-----------------------------------------------------------------%
```

```
paths = u1.^2+u2.^2;

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

%--------------------------------------------------------------------%
% Reshape the path contraints into a column vector.              %
%--------------------------------------------------------------------%
paths = reshape(paths,N*npaths,1);

%--------------------------------------------------------------------%
% Compute the nonlinear boundary condition.                         %
%--------------------------------------------------------------------%
bcs = [];

%--------------------------------------------------------------------%
% Construct the objective function plus constraint vector.          %
%--------------------------------------------------------------------%
C = [-mF;defects;paths];
end

%OrbitTransferCon.m
function constraints = OrbitTransferCon(Z)
% computes the constraints

output      = OrbitTransferFun(Z);
constraints = output;

end

%OrbitTransferGrd.m
function grd = OrbitTransferGrd(Z)
% computes the gradient

output = OrbitTransferObj_Jac(Z);
grd    = output;

end

%OrbitTransferJac.m
function jac = OrbitTransferJac(Z)
% computes the jacobian

[jac,~] = OrbitTransferFun_Jac(Z);

end

%OrbitTransferObj.m
function obj = OrbitTransferObj(z)
% Computes the objective function of the problem

global psStuff nstates ncontrols CONSTANTS
```

```
%-------------------------------------------------------------------%
%         Extract the constants used in the problem.                %
%-------------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot;  ve = CONSTANTS.ve;

%-------------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%-------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
control = reshape(controlVector,N,ncontrols);

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

```
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
u2 = control(:,2);
T = control(:,3);

%--------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
%--------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%--------------------------------------------------------------------%
% (Edit) The orbit-transfer problem contains one nonlinear boundary%
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%--------------------------------------------------------------------%
tF = tf;

% Cost Function
J   = -stateF(5);
obj = J;

end
```

(2) For the objective to maximize mass at final time (M_tf) using beta and T as control inputs

```
% ------------------------------------------------------%
%             Orbit-Transfer Problem                    %
% ------------------------------------------------------%
% Solve the following optimal control problem:       %
% Minimize t_f                                       %
% subject to the differential equation constraints   %
%   dr/dt = v_r                                      %
%   d\theta/dt = v_\theta/r                          %
%   dv_r/dt = v_\theta^2/r-\mu/r^2+a u_1             %
%   dv_\theta/dt = -v_rv_\theta/r+a u_2/m            %
%   dm/dt = - a/v_e
% the equality path constraint                       %
%   u_1^2 + u_2^2 = 1                                %
% and the boundary conditions                        %
%   r(0) = 1                                         %
%   \theta(0) = 0                                    %
%   v_r(0) = 0                                       %
%   v_\theta(0) = sqrt(mu/r(0))                      %
%   m(0) = 1                                         %
%   r(t_f) = 1.5                                     %
%   v_r(t_f) = 0                                     %
%   v_\theta(t_f) = sqrt(mu/r(t_f)                   %
```

```
% ------------------------------------------------- %
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------- %
global igrid CONSTANTS psStuff nstates ncontrols npaths
% ------------------------------------------------- %
% END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------- %

CONSTANTS.MU = 1;
CONSTANTS.m0 = 1;
CONSTANTS.mdot = 0.0749;
CONSTANTS.ve = 1.8758344;
nstates = 5;
ncontrols = 2;
npaths = 1;

% Bounds on State and Control
r0 = 1; theta0 = 0; vr0 = 0; vtheta0 = sqrt(CONSTANTS.MU/r0); m0 = 1;
rf = 1.5; vrf = 0;  vthetaf = sqrt(CONSTANTS.MU/rf);
rmin = 0.5; rmax = 2;
thetamin = 0; thetamax = 4*pi;
vrmin = -10; vrmax = 10;
vthetamin = -10; vthetamax = 10;
mmin = 0.5; mmax = m0;
u1min = -10; u1max = 10;
u2min = -10; u2max = 10;
Tmin = 0; Tmax = 0.1405;
t0min = 0; t0max = 0;
tfmin = 0; tfmax = 50;

%-------------------------------------------------------------------%
%      Compute Points, Weights, and Differentiation Matrix          %
%-------------------------------------------------------------------%
%-------------------------------------------------------------------%
% Choose Polynomial Degree and Number of Mesh Intervals             %
% numIntervals = 1 ===> p-method                                    %
% numIntervals > 1 ===> h-method                                    %
%-------------------------------------------------------------------%
N = 3;
numIntervals = 2;
%-------------------------------------------------------------------%
% 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 NLP variables.
zrmin = rmin*ones(length(tau),1);
zrmax = rmax*ones(length(tau),1);
```

```
zrmin(1) = r0; zrmax(1) = r0;
zrmin(end) = rf; zrmax(end) = rf;

zthetamin = thetamin*ones(length(tau),1);
zthetamax = thetamax*ones(length(tau),1);
zthetamin(1) = theta0; zthetamax(1) = theta0;

zvrmin = vrmin*ones(length(tau),1);
zvrmax = vrmax*ones(length(tau),1);
zvrmin(1) = vr0; zvrmax(1) = vr0;
zvrmin(end) = vrf; zvrmax(end) = vrf;

zvthetamin = vthetamin*ones(length(tau),1);
zvthetamax = vthetamax*ones(length(tau),1);
zvthetamin(1) = vtheta0; zvthetamax(1) = vtheta0;
zvthetamin(end) = vthetaf; zvthetamax(end) = vthetaf;

zmmin = mmin*ones(length(tau),1);
zmmax = mmax*ones(length(tau),1);
zmmin(1) = mmax; zmmax(1) = mmax;

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);

zTmin = Tmin*ones(length(tau)-1,1);
zTmax = Tmax*ones(length(tau)-1,1);

zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zTmin; t0min; tfmin];
zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zTmax; t0max; tfmax];

% Set the bounds on the NLP constraints
% There are NSTATES sets of defect constraints.
defectMin = zeros(nstates*(length(tau)-1),1);
defectMax = zeros(nstates*(length(tau)-1),1);
% There are no path contstraints in this problem
% pathMin = ones(length(tau)-1,1); pathMax = ones(length(tau)-1,1);
% There is one nonlinear event constraint
bcMin = 0; bcMax = 0;
objMin = -inf; objMax = inf;
Fmin = [objMin; defectMin];
Fmax = [objMax; defectMax];

% Supply an initial guess
rguess = linspace(r0,rf,NLGR+1).';
thetaguess = linspace(theta0,theta0,NLGR+1).';
vrguess = linspace(vr0,vrf,NLGR+1).';
vthetaguess = linspace(vtheta0,vthetaf,NLGR+1).';
mguess = linspace(mmax,mmin,NLGR+1).';
u1guess = linspace(1,1,NLGR).';
% u2guess = linspace(0,0,NLGR).';
Tguess = linspace(Tmax,Tmax,NLGR).';
```

```
t0guess = 0;
tfguess = 3.2481;
z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;Tguess;t0guess;tfguess];

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

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

%-------------------------------------------------------------------%
% set IPOPT callback functions
%-------------------------------------------------------------------%
funcs.objective   = @(Z)OrbitTransferObj(Z);
funcs.gradient    = @(Z)OrbitTransferGrd(Z);
funcs.constraints = @(Z)OrbitTransferCon(Z);
funcs.jacobian    = @(Z)OrbitTransferJac(Z);
funcs.jacobianstructure = @()OrbitTransferJacPat(S_jac);
options.ipopt.hessian_approximation = 'limited-memory';

%-------------------------------------------------------------------%
% Set IPOPT Options %
%-------------------------------------------------------------------%
options.ipopt.tol = 1e-8;
options.ipopt.linear_solver = 'ma57';
options.ipopt.max_iter = 2000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['OrbitTransfer','IPOPTinfo.txt']; % print output file
options.ipopt.print_level = 5; % set print level default

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

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

%-------------------------------------------------------------------%
% extract lagrange multipliers from ipopt output, info
```

```
%-------------------------------------------------------------------%
Fmul = info.lambda;

% Extract the state and control from the decision vector z.
% Remember that the state is approximated at the LGR points
% plus the final point, while the control is only approximated
% at only the LGR points.
r = z(1:NLGR+1);
theta = z(NLGR+2:2*(NLGR+1));
vr = z(2*(NLGR+1)+1:3*(NLGR+1));
vtheta = z(3*(NLGR+1)+1:4*(NLGR+1));
m = z(4*(NLGR+1)+1:5*(NLGR+1));
u1 = z(5*(NLGR+1)+1:5*(NLGR+1)+NLGR);
T = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
% T = z(5*(NLGR+1)+2*NLGR+1:5*(NLGR+1)+3*NLGR);
% beta = 180/pi*atan2(u1,u2);
t0 = z(end-1);
tf = z(end);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);
%-------------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                    %
% the defect constraints.                                           %
%-------------------------------------------------------------------%
multipliersDefects = Fmul(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];
lamr = costate(:,1); lamtheta = costate(:,2);
lamvr = costate(:,3); lamvtheta = costate(:,4);
lamm = costate(:,5);

%-------------%
% Plot Results
%-------------%
figure(1)
plot(t,r,'r',t,theta,'b',t,vr,'m',t,vtheta,'k', t, m,'g');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('states','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
title('Plot of States');
set(gca,'FontName','Times','FontSize',18);
grid on;
```

```
figure(2);
subplot(2,1,1);
plot(tLGR,u1,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$u_1(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
title('Throttle angle');
grid on;

subplot(2,1,2);
plot(tLGR,T,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$T(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
grid on;

figure(3);
plot(t,lamr,'r',t,lamtheta,'b',t,lamvr,'m',t,lamvtheta,'k', t, lamm,'g');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('Costates','Interpreter','LaTeX');
title('Plot of CoStates');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
grid on;

figure (4);
polarplot(theta,r);
title('Position of the Spacecraft');

%OrbitTransferFun.m
function C = OrbitTransferFun(z)
%---------------------------------------------------------------------%
% Objective and constraint functions for the orbit-transfer        %
% 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 npaths CONSTANTS                 %
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE!        %
%---------------------------------------------------------------------%

%---------------------------------------------------------------------%
%         Extract the constants used in the problem.              %
%---------------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot; ve = CONSTANTS.ve;

%---------------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%-------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
control = reshape(controlVector,N,ncontrols);

%-------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR.   %
%-------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
T = control(:,2);
% T =control(:,3);
```

```
%-------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
%-------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%-------------------------------------------------------------------%
% The orbit-raising problem contains one nonlinear boundary       %
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%-------------------------------------------------------------------%
rF = stateF(1);
vrF = stateF(3);
vthetaF = stateF(4);
mF = stateF(5);

% a = T./m;

%-------------------------------------------------------------------%
% 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.                              %
%-------------------------------------------------------------------%
rdot = vr;
thetadot = vtheta./r;
vrdot = vtheta.^2./r-MU./r.^2+T.*sin(u1)./m;
vthetadot = -vr.*vtheta./r+T.*cos(u1)./m;
m_dot = -T./ve;
diffeqRHS = [rdot, thetadot, vrdot, vthetadot, m_dot];

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

%--------------------------------------------------------------------%
% Construct the path constraints at the N LGR points.              %
%--------------------------------------------------------------------%
% paths = [];

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

%--------------------------------------------------------------------%
% Reshape the path contraints into a column vector.              %
%--------------------------------------------------------------------%
% paths = reshape(paths,N*npaths,1);

%--------------------------------------------------------------------%
% Compute the nonlinear boundary condition.                         %
%--------------------------------------------------------------------%
bcs = [];

%--------------------------------------------------------------------%
% Construct the objective function plus constraint vector.         %
%--------------------------------------------------------------------%
C = [-mF;defects];
end

%OrbitTransferCon.m
function constraints = OrbitTransferCon(Z)
% computes the constraints

output      = OrbitTransferFun(Z);
constraints = output;

end

%OrbitTranferObj.m
function obj = OrbitTransferObj(z)
% Computes the objective function of the problem

global psStuff nstates ncontrols CONSTANTS

%--------------------------------------------------------------------%
%         Extract the constants used in the problem.                %
%--------------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot;  ve = CONSTANTS.ve;

%--------------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%-------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
control = reshape(controlVector,N,ncontrols);

%-------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR. %
%-------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
T = control(:,2);
% T = control(:,3);

%-------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
```

```
%-------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%-------------------------------------------------------------------%
% (Edit) The orbit-transfer problem contains one nonlinear boundary%
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%-------------------------------------------------------------------%
tF = tf;

% Cost Function
J   = -stateF(5);
obj = J;

end
```

(3) For the objective to minimize final time (tf) using U1, U2 ,and T as control inputs

```
% ------------------------------------------------%
%             Orbit-Transfer Problem               %
% ------------------------------------------------%
% Solve the following optimal control problem:       %
% Minimize t_f                                       %
% subject to the differential equation constraints   %
%   dr/dt = v_r                                      %
%   d\theta/dt = v_\theta/r                          %
%   dv_r/dt = v_\theta^2/r-\mu/r^2+a u_1             %
%   dv_\theta/dt = -v_rv_\theta/r+a u_2/m            %
%   dm/dt = - a/v_e
% the equality path constraint                       %
%   u_1^2 + u_2^2 = 1                                %
% and the boundary conditions                        %
%   r(0) = 1                                         %
%   \theta(0) = 0                                    %
%   v_r(0) = 0                                       %
%   v_\theta(0) = sqrt(mu/r(0))                      %
%   m(0) = 1                                         %
%   r(t_f) = 1.5                                     %
%   v_r(t_f) = 0                                     %
%   v_\theta(t_f) = sqrt(mu/r(t_f)                   %
% ------------------------------------------------ %
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------ %
global igrid CONSTANTS psStuff nstates ncontrols npaths
% ------------------------------------------------ %
% END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------ %

CONSTANTS.MU = 1;
CONSTANTS.m0 = 1;
CONSTANTS.mdot = 0.0749;
```

```
CONSTANTS.ve = 1.8758344;
nstates = 5;
ncontrols = 3;
npaths = 1;

% Bounds on State and Control
r0 = 1; theta0 = 0; vr0 = 0; vtheta0 = sqrt(CONSTANTS.MU/r0); m0 = 1;
rf = 1.5; vrf = 0;  vthetaf = sqrt(CONSTANTS.MU/rf);
rmin = 0.5; rmax = 2;
thetamin = 0; thetamax = 4*pi;
vrmin = -30; vrmax = 30;
vthetamin = -30; vthetamax = 30;
mmin = 0.1; mmax = m0;
u1min = -3; u1max = 3;
u2min = -3; u2max = 3;
Tmin = 0; Tmax = 0.1405;
t0min = 0; t0max = 0;
tfmin = 0; tfmax = 50;

%---------------------------------------------------------------------%
%      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 = 8;
%---------------------------------------------------------------------%
% 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 NLP variables.
zrmin = rmin*ones(length(tau),1);
zrmax = rmax*ones(length(tau),1);
zrmin(1) = r0; zrmax(1) = r0;
zrmin(end) = rf; zrmax(end) = rf;

zthetamin = thetamin*ones(length(tau),1);
zthetamax = thetamax*ones(length(tau),1);
zthetamin(1) = theta0; zthetamax(1) = theta0;

zvrmin = vrmin*ones(length(tau),1);
zvrmax = vrmax*ones(length(tau),1);
zvrmin(1) = vr0; zvrmax(1) = vr0;
zvrmin(end) = vrf; zvrmax(end) = vrf;
```

```
zvthetamin = vthetamin*ones(length(tau),1);
zvthetamax = vthetamax*ones(length(tau),1);
zvthetamin(1) = vtheta0; zvthetamax(1) = vtheta0;
zvthetamin(end) = vthetaf; zvthetamax(end) = vthetaf;

zmmin = mmin*ones(length(tau),1);
zmmax = mmax*ones(length(tau),1);
zmmin(1) = mmax; zmmax(1) = mmax;

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);

zTmin = Tmin*ones(length(tau)-1,1);
zTmax = Tmax*ones(length(tau)-1,1);

zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zu2min; zTmin; tOmin; tfmin];
zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zu2max; zTmax; tOmax; tfmax];

% Set the bounds on the NLP constraints
% There are NSTATES sets of defect constraints.
defectMin = zeros(nstates*(length(tau)-1),1);
defectMax = zeros(nstates*(length(tau)-1),1);
% There is one path constraint
pathMin = ones(length(tau)-1,1); pathMax = ones(length(tau)-1,1);
% There is one nonlinear event constraint
bcMin = 0; bcMax = 0;
objMin = tfmin; objMax = tfmax;
Fmin = [objMin; defectMin; pathMin];
Fmax = [objMax; defectMax; pathMax];

% Supply an initial guess
rguess = linspace(r0,rf,NLGR+1).';
thetaguess = linspace(theta0,theta0,NLGR+1).';
vrguess = linspace(vr0,vrf,NLGR+1).';
vthetaguess = linspace(vtheta0,vthetaf,NLGR+1).';
mguess = linspace(mmax,mmin,NLGR+1).';
u1guess = linspace(0,0,NLGR).';
u2guess = linspace(1,1,NLGR).';
Tguess = linspace(Tmax,Tmax,NLGR).';
t0guess = 0;
tfguess = 100;
z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;u2guess;Tguess;t0guess;tfguess];

%--------------------------------------------------------------------%
% Generate derivatives and sparsity pattern using Adigator        %
%--------------------------------------------------------------------%
% - Constraint Function Derivatives
xsize  = size(z0);
x      = adigatorCreateDerivInput(xsize,'z0');
output = adigatorGenJacFile('OrbitTransferFun',{x});
```

```
S_jac  = output.JacobianStructure;
[iGfun,jGvar] = find(S_jac);

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

%-------------------------------------------------------------------%
% set IPOPT callback functions
%-------------------------------------------------------------------%
funcs.objective   = @(Z)OrbitTransferObj(Z);
funcs.gradient    = @(Z)OrbitTransferGrd(Z);
funcs.constraints = @(Z)OrbitTransferCon(Z);
funcs.jacobian    = @(Z)OrbitTransferJac(Z);
funcs.jacobianstructure = @()OrbitTransferJacPat(S_jac);
options.ipopt.hessian_approximation = 'limited-memory';

%-------------------------------------------------------------------%
% Set IPOPT Options %
%-------------------------------------------------------------------%
options.ipopt.tol = 1e-8;
options.ipopt.linear_solver = 'ma57';
options.ipopt.max_iter = 2000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['OrbitTransfer','IPOPTinfo.txt']; % print output file
options.ipopt.print_level = 5; % set print level default

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

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

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

% Extract the state and control from the decision vector z.
% Remember that the state is approximated at the LGR points
% plus the final point, while the control is only approximated
% at only the LGR points.
r = z(1:NLGR+1);
theta = z(NLGR+2:2*(NLGR+1));
vr = z(2*(NLGR+1)+1:3*(NLGR+1));
vtheta = z(3*(NLGR+1)+1:4*(NLGR+1));
```

```
m = z(4*(NLGR+1)+1:5*(NLGR+1));
u1 = z(5*(NLGR+1)+1:5*(NLGR+1)+NLGR);
u2 = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
T = z(5*(NLGR+1)+2*NLGR+1:5*(NLGR+1)+3*NLGR);
beta = 180/pi*atan2(u1,u2);
t0 = z(end-1);
tf = z(end);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);
%--------------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                     %
% the defect constraints.                                            %
%--------------------------------------------------------------------%
multipliersDefects = Fmul(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];
lamr = costate(:,1); lamtheta = costate(:,2);
lamvr = costate(:,3); lamvtheta = costate(:,4);
lamm = costate(:,5);

%-------------%
% Plot Results
%-------------%
figure(1);
plot(t,r,'b', t, theta, 'g', t, vr, 'r', t, vtheta, 'c', t, m, 'k' );
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$r(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
title('Plot of States');
legend('r','theta','vr','vtheta','location','best');
set(gca,'FontName','Times','FontSize',18);
grid on;

figure(2);
plot(tLGR,beta,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$\beta(t)$','Interpreter','LaTeX');
title('Throttle angle Beta');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
grid on;
```

```
figure(3);
subplot(1,3,1);
plot(tLGR,u1,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$u_1(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
title('Control Input u_1');
grid on;

subplot(1,3,2);
plot(tLGR,u2,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$u_2(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
title('Control Input u_2');
grid on;

subplot(1,3,3);
plot(tLGR,T,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$T(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
title('Control Input T');
grid on;

figure(4);
plot(t,lamr,'b', t, lamtheta, 'g', t, lamvr, 'r', t, lamvtheta, 'c', t, lamm, 'k' );
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$\lambda_r(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
legend('lamr','lamtheta','lamvr','lamvtheta','lamm','location','best');
title('Plot of CoStates');
grid on;

figure (5);
polarplot(theta,r);
title('Polar Plot of the location of the Spacecraft');

%OrbitTransferFun.m
function C = OrbitTransferFun(z)

%------------------------------------------------------------------%
% Objective and constraint functions for the orbit-transfer       %
% 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 npaths CONSTANTS                  %
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE!        %
%-------------------------------------------------------------------%

%-------------------------------------------------------------------%
%         Extract the constants used in the problem.              %
%-------------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot; ve = CONSTANTS.ve;

%-------------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%-------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
control = reshape(controlVector,N,ncontrols);
```

```
%-------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR. %
%-------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
u2 = control(:,2);
T =control(:,3);

%-------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
%-------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%-------------------------------------------------------------------%
% The orbit-raising problem contains one nonlinear boundary       %
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%-------------------------------------------------------------------%
rF = stateF(1);
vrF = stateF(3);
vthetaF = stateF(4);

% a = T./m;

%-------------------------------------------------------------------%
% 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.                              %
%-------------------------------------------------------------------%
rdot = vr;
thetadot = vtheta./r;
vrdot = vtheta.^2./r-MU./r.^2+T.*u1./m;
vthetadot = -vr.*vtheta./r+T.*u2./m;
m_dot = -T./ve;
diffeqRHS = [rdot, thetadot, vrdot, vthetadot, m_dot];

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

```
% 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;

%-----------------------------------------------------------------%
% Construct the path constraints at the N LGR points.            %
%-----------------------------------------------------------------%
paths = u1.^2+u2.^2;

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

%-----------------------------------------------------------------%
% Reshape the path contraints into a column vector.             %
%-----------------------------------------------------------------%
paths = reshape(paths,N*npaths,1);

%-----------------------------------------------------------------%
% Compute the nonlinear boundary condition.                       %
%-----------------------------------------------------------------%
bcs = [];

%-----------------------------------------------------------------%
% Construct the objective function plus constraint vector.        %
%-----------------------------------------------------------------%
C = [tf;defects;paths];
end

%OrbitTransferObj.m
function obj = OrbitTransferObj(z)
% Computes the objective function of the problem

global psStuff nstates ncontrols CONSTANTS

%-----------------------------------------------------------------%
%         Extract the constants used in the problem.              %
%-----------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot;  ve = CONSTANTS.ve;

%-----------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%-------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
control = reshape(controlVector,N,ncontrols);

%-------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR.   %
%-------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
u2 = control(:,2);
T = control(:,3);
```

```
%-------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
%-------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%-------------------------------------------------------------------%
% (Edit) The orbit-transfer problem contains one nonlinear boundary%
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%-------------------------------------------------------------------%
tF = tf;

% Cost Function
J   = tF;
obj = J;

end

%OrbitTransferCon.m
function constraints = OrbitTransferCon(Z)
% computes the constraints

output      = OrbitTransferFun(Z);
constraints = output;

end

%OrbitTransferGrd.m
function grd = OrbitTransferGrd(Z)
% computes the gradient

output = OrbitTransferObj_Jac(Z);
grd    = output;

end

%OrbitTransferJac.m
function jac = OrbitTransferJac(Z)
% computes the jacobian

[jac,~] = OrbitTransferFun_Jac(Z);

end

%OrbitTransferJacPat.m
function jacpat = OrbitTransferJacPat(S_jac)
% computes the jacobian structure

jacpat = S_jac;

end
```

(4) For the objective to minimize final time (tf) using beta and T as control inputs

```
% ------------------------------------------------%
%             Orbit-Transfer Problem               %
% ------------------------------------------------%
% Solve the following optimal control problem:     %
% Minimize t_f                                     %
% subject to the differential equation constraints %
%   dr/dt = v_r                                    %
%   d\theta/dt = v_\theta/r                        %
%   dv_r/dt = v_\theta^2/r-\mu/r^2+a u_1           %
%   dv_\theta/dt = -v_rv_\theta/r+a u_2/m          %
%   dm/dt = - a/v_e
% the equality path constraint                     %
%   u_1^2 + u_2^2 = 1                              %
% and the boundary conditions                      %
%   r(0) = 1                                       %
%   \theta(0) = 0                                  %
%   v_r(0) = 0                                     %
%   v_\theta(0) = sqrt(mu/r(0))                    %
%   m(0) = 1                                       %
%   r(t_f) = 1.5                                   %
%   v_r(t_f) = 0                                   %
%   v_\theta(t_f) = sqrt(mu/r(t_f)                 %
% ------------------------------------------------ %
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------ %
global igrid CONSTANTS psStuff nstates ncontrols npaths
% ------------------------------------------------ %
% END:   DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% ------------------------------------------------ %

CONSTANTS.MU = 1;
CONSTANTS.m0 = 1;
CONSTANTS.mdot = 0.0749;
CONSTANTS.ve = 1.8758344;
nstates = 5;
ncontrols = 2;
npaths = 0;

% Bounds on State and Control
r0 = 1; theta0 = 0; vr0 = 0; vtheta0 = sqrt(CONSTANTS.MU/r0); m0 = 1;
rf = 1.5; vrf = 0;  vthetaf = sqrt(CONSTANTS.MU/rf);
rmin = 0.5; rmax = 2;
thetamin = 0; thetamax = 4*pi;
vrmin = -30; vrmax = 30;
vthetamin = -30; vthetamax = 30;
mmin = 0.1; mmax = m0;
u1min = -3; u1max = 3;
Tmin = 0; Tmax = 0.1405;
t0min = 0; t0max = 0;
tfmin = 0; tfmax = 50;
```

```
%-------------------------------------------------------------------%
%      Compute Points, Weights, and Differentiation Matrix        %
%-------------------------------------------------------------------%
%-------------------------------------------------------------------%
% Choose Polynomial Degree and Number of Mesh Intervals            %
% numIntervals = 1 ===> p-method                                   %
% numIntervals > 1 ===> h-method                                   %
%-------------------------------------------------------------------%
N = 3;
numIntervals = 2;
%-------------------------------------------------------------------%
% 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 NLP variables.
zrmin = rmin*ones(length(tau),1);
zrmax = rmax*ones(length(tau),1);
zrmin(1) = r0; zrmax(1) = r0;
zrmin(end) = rf; zrmax(end) = rf;

zthetamin = thetamin*ones(length(tau),1);
zthetamax = thetamax*ones(length(tau),1);
zthetamin(1) = theta0; zthetamax(1) = theta0;

zvrmin = vrmin*ones(length(tau),1);
zvrmax = vrmax*ones(length(tau),1);
zvrmin(1) = vr0; zvrmax(1) = vr0;
zvrmin(end) = vrf; zvrmax(end) = vrf;

zvthetamin = vthetamin*ones(length(tau),1);
zvthetamax = vthetamax*ones(length(tau),1);
zvthetamin(1) = vtheta0; zvthetamax(1) = vtheta0;
zvthetamin(end) = vthetaf; zvthetamax(end) = vthetaf;

zmmin = mmin*ones(length(tau),1);
zmmax = mmax*ones(length(tau),1);
zmmin(1) = mmax; zmmax(1) = mmax;

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);

zTmin = Tmin*ones(length(tau)-1,1);
zTmax = Tmax*ones(length(tau)-1,1);
```

```
zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zTmin; t0min; tfmin];
zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zTmax; t0max; tfmax];

% Set the bounds on the NLP constraints
% There are NSTATES sets of defect constraints.
defectMin = zeros(nstates*(length(tau)-1),1);
defectMax = zeros(nstates*(length(tau)-1),1);
% There is no path contraint in this problem
% pathMin = ones(length(tau)-1,1); pathMax = ones(length(tau)-1,1);
% There is one nonlinear event constraint
bcMin = 0; bcMax = 0;
objMin = tfmin; objMax = tfmax;
Fmin = [objMin; defectMin];
Fmax = [objMax; defectMax];

% Supply an initial guess
rguess = linspace(r0,rf,NLGR+1).';
thetaguess = linspace(theta0,theta0,NLGR+1).';
vrguess = linspace(vr0,vrf,NLGR+1).';
vthetaguess = linspace(vtheta0,vthetaf,NLGR+1).';
mguess = linspace(mmax,mmin,NLGR+1).';
u1guess = linspace(0,0,NLGR).';
u2guess = linspace(1,1,NLGR).';
Tguess = linspace(Tmax,Tmax,NLGR).';
t0guess = 0;
tfguess = 100;
z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;Tguess;t0guess;tfguess];

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

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

%-------------------------------------------------------------------%
% set IPOPT callback functions
%-------------------------------------------------------------------%
funcs.objective   = @(Z)OrbitTransferObj(Z);
funcs.gradient    = @(Z)OrbitTransferGrd(Z);
funcs.constraints = @(Z)OrbitTransferCon(Z);
funcs.jacobian    = @(Z)OrbitTransferJac(Z);
funcs.jacobianstructure = @()OrbitTransferJacPat(S_jac);
options.ipopt.hessian_approximation = 'limited-memory';
```

```
%-------------------------------------------------------------------%
% Set IPOPT Options %
%-------------------------------------------------------------------%
options.ipopt.tol = 1e-8;
options.ipopt.linear_solver = 'ma57';
options.ipopt.max_iter = 2000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['OrbitTransfer','IPOPTinfo.txt']; % print output file
options.ipopt.print_level = 5; % set print level default

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

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

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

% Extract the state and control from the decision vector z.
% Remember that the state is approximated at the LGR points
% plus the final point, while the control is only approximated
% at only the LGR points.
r = z(1:NLGR+1);
theta = z(NLGR+2:2*(NLGR+1));
vr = z(2*(NLGR+1)+1:3*(NLGR+1));
vtheta = z(3*(NLGR+1)+1:4*(NLGR+1));
m = z(4*(NLGR+1)+1:5*(NLGR+1));
u1 = z(5*(NLGR+1)+1:5*(NLGR+1)+NLGR);
T = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
% T = z(5*(NLGR+1)+2*NLGR+1:5*(NLGR+1)+3*NLGR);
t0 = z(end-1);
tf = z(end);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);
%-------------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding                    %
% the defect constraints.                                           %
%-------------------------------------------------------------------%
multipliersDefects = Fmul(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];
lamr = costate(:,1); lamtheta = costate(:,2);
lamvr = costate(:,3); lamvtheta = costate(:,4);
lamm = costate(:,5);

%-------------%
% Plot Results
%-------------%
figure(1);
plot(t,r,'b', t, theta, 'g', t, vr, 'r', t, vtheta, 'c', t, m, 'k' );
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$r(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
title('Plot of States');
legend('r','theta','vr','vtheta','location','best');
set(gca,'FontName','Times','FontSize',18);
grid on;

figure(2);
subplot(1,2,1);
plot(tLGR,u1,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$u_1(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
title('Throttle angle beta');
grid on;

subplot(1,2,2);
plot(tLGR,T,'-o');
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$T(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
title('Control Input T');
grid on;

figure(3);
plot(t,lamr,'b', t, lamtheta, 'g', t, lamvr, 'r', t, lamvtheta, 'c', t, lamm, 'k' );
xl = xlabel('$t$','Interpreter','LaTeX');
yl = ylabel('$\lambda_r(t)$','Interpreter','LaTeX');
set(xl,'FontSize',18);
set(yl,'FontSize',18);
set(gca,'FontName','Times','FontSize',18);
legend('lamr','lamtheta','lamvr','lamvtheta','lamm','location','best');
```

```
title('Plot of CoStates');
grid on;

figure (4);
polarplot(theta,r);
title('Polar Plot of the location of the Spacecraft');

%OrbitTransferFun.m
function C = OrbitTransferFun(z)

%--------------------------------------------------------------------%
% Objective and constraint functions for the orbit-transfer       %
% 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 npaths CONSTANTS                  %
%      DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE!        %
%--------------------------------------------------------------------%

%--------------------------------------------------------------------%
%         Extract the constants used in the problem.              %
%--------------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot; ve = CONSTANTS.ve;

%--------------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%--------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
control = reshape(controlVector,N,ncontrols);

%------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR. %
%------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
T = control(:,2);


%------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
%------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%------------------------------------------------------------------%
% The orbit-raising problem contains one nonlinear boundary       %
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%------------------------------------------------------------------%
rF = stateF(1);
vrF = stateF(3);
vthetaF = stateF(4);

% a = T./m;

%------------------------------------------------------------------%
% 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.                                    %
%-----------------------------------------------------------------------%
rdot = vr;
thetadot = vtheta./r;
vrdot = vtheta.^2./r-MU./r.^2+T.*sin(u1)./m;
vthetadot = -vr.*vtheta./r+T.*cos(u1)./m;
m_dot = -T./ve;
diffeqRHS = [rdot, thetadot, vrdot, vthetadot, m_dot];

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

%-----------------------------------------------------------------------%
% Construct the path constraints at the N LGR points.                  %
%-----------------------------------------------------------------------%
paths = 0;

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

%-----------------------------------------------------------------------%
% Reshape the path contraints into a column vector.                  %
%-----------------------------------------------------------------------%
% paths = reshape(paths,N*npaths,1);

%-----------------------------------------------------------------------%
% Compute the nonlinear boundary condition.                             %
%-----------------------------------------------------------------------%
bcs = [];

%-----------------------------------------------------------------------%
% Construct the objective function plus constraint vector.              %
%-----------------------------------------------------------------------%
C = [tf;defects];
end
```

```
%OrbitTransferObj.m
function obj = OrbitTransferObj(z)
% Computes the objective function of the problem

global psStuff nstates ncontrols CONSTANTS

%--------------------------------------------------------------------%
%         Extract the constants used in the problem.                 %
%--------------------------------------------------------------------%
MU = CONSTANTS.MU; mdot = CONSTANTS.mdot;  ve = CONSTANTS.ve;

%--------------------------------------------------------------------%
% 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);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);

%--------------------------------------------------------------------%
% 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);
stateLGR = statePlusEnd(1:end-1,:);
```

```
control = reshape(controlVector,N,ncontrols);

%-------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR. %
%-------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
u1 = control(:,1);
T = control(:,2);

%-------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final      %
% time, tf, which corresponds to the state at $\tau=1$.           %
%-------------------------------------------------------------------%
stateF = statePlusEnd(end,:);
%-------------------------------------------------------------------%
% (Edit) The orbit-transfer problem contains one nonlinear boundary%
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$.  Because $r(t)$  %
% and $v_\theta(t)$ are the first and fourth components of the    %
% state, it is necessary to extract stateF(1) and stateF(4) in    %
% order to compute this boundary condition function.              %
%-------------------------------------------------------------------%
tF = tf;

% Cost Function
J   = tF;
obj = J;

end

%OrbitTransferCon.m
function constraints = OrbitTransferCon(Z)
% computes the constraints

output      = OrbitTransferFun(Z);
constraints = output;

end

%OrbitTransferGrd.m
function grd = OrbitTransferGrd(Z)
% computes the gradient

output = OrbitTransferObj_Jac(Z);
grd    = output;

end

%OrbitTransferJac.m
function jac = OrbitTransferJac(Z)
% computes the jacobian
```

```
[jac,~] = OrbitTransferFun_Jac(Z);

end

%OrbitTransferJacPat.m
function jacpat = OrbitTransferJacPat(S_jac)
% computes the jacobian structure

jacpat = S_jac;

end
```