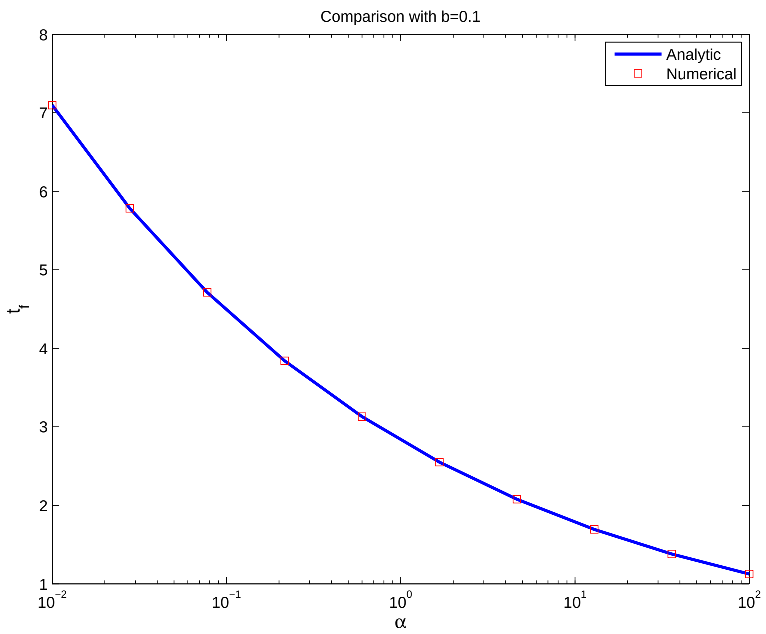
16.323 Principles of Optimal Control
Spring 2008

16.323 Lecture 7

Numerical Solution in Matlab



Simple Problem

- Performance

$$J = \int_{t_0}^{t_f} (u - x)^2 dt$$

with dynamics $\dot{x} = u$ and BC $t_0 = 0$, $x_0 = 1$, $t_f = 1$.

– So this is a fixed final time, free final state problem.

- Form Hamiltonian

$$H = (u - x)^2 + pu$$

- Necessary conditions become:

$$\dot{x} = u \quad (7.25)$$

$$\dot{p} = -2(u - x)(-1) \quad (7.26)$$

$$0 = 2(u - x) + p \quad (7.27)$$

with BC that $p(t_f) = 0$.

- Rearrange to get

$$\dot{p} = -p \quad (7.28)$$

$$\Rightarrow p(t) = c_1 e^{-t} \quad (7.29)$$

But now impose BC to get

$$p(t) = 0 \quad (7.30)$$

- This implies that $u = x$ is the optimal solution, and the closed-loop dynamics are

$$\dot{x} = x$$

with solution $x(t) = e^t$.

– Clearly this would be an unstable response on a longer timescale, but given the cost and the short time horizon, this control is the best you can do.

Simple Zermelo's Problem

- Consider ship that has to travel through a region of strong currents. The ship is assumed to have constant speed V but its heading θ can be varied. The current is assumed to be in the y direction with speed of w.

- The motion of the boat is then given by the dynamics

$$\dot{x} = V\cos\theta \tag{7.31}$$
$$\dot{y} = V\sin\theta + w \tag{7.32}$$

- The goal is to minimize time, the performance index is

$$J = \int_0^{t_f} 1dt = t_f$$

– with BC $x_0 = y_0 = 0$, $x_f = 1$, $y_f = 0$
– Final time is unspecified.

- Define costate $\mathbf{p} = [p_1\ p_2]^T$, and in this case the Hamiltonian is

$$H = 1 + p_1(V\cos\theta) + p_2(V\sin\theta + w)$$

- Now use the necessary conditions to get ($\dot{\mathbf{p}} = -H_{\mathbf{x}}^T$)

$$\dot{p}_1 = -\frac{\partial H}{\partial x} = 0 \qquad \rightarrow p_1 = c_1 \tag{7.33}$$
$$\dot{p}_2 = -\frac{\partial H}{\partial y} = 0 \qquad \rightarrow p_2 = c_2 \tag{7.34}$$

- Control input $\theta(t)$ is unconstrained, so have $(H_{\mathbf{u}} = 0)$
$$\frac{\partial H}{\partial u} = -p_1 V \sin\theta + p_2 V \cos\theta = 0 \tag{7.35}$$
which gives the control law
$$\tan\theta = \frac{p_2}{p_1} = \frac{-p_2}{-p_1} \tag{7.36}$$
  – Since p_1 and p_2 are constants, then $\theta(t)$ is also a constant.

- Optimal control is constant, so can integrate the state equations:
$$x = Vt\cos\theta \tag{7.37}$$
$$y = Vt(\sin\theta + w) \tag{7.38}$$
  – Now impose the BC to get $x(t_f) = 1$, $y(t_f) = 0$ to get
$$t_f = \frac{1}{V\cos\theta} \qquad \sin\theta = -\frac{w}{V}$$

- Rearrange to get
$$\cos\theta = \frac{\sqrt{V^2 - w^2}}{V}$$
which gives that
$$t_f = \frac{1}{\sqrt{V^2 - w^2}} \qquad \theta = -\arcsin\frac{w}{V}$$
  – Does this make sense?

Numerical Solutions

- Most of the problems considered so far have been simple. Things get more complicated by the need to solve a two-point boundary value problem when the dynamics are nonlinear.
- Numerous solution techniques exist, including `shooting methods`[13] and `collocation`
  - Will discuss the details on these later, but for now, let us look at how to solve these use existing codes
- Matlab code called `BVP4C` exists that is part of the standard package [14]
  - Solves problems of a "standard form":

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t, \mathbf{p}) \qquad a \le t \le b$$

    where $\mathbf{y}$ are the variables of interest, and $\mathbf{p}$ are extra variables in the problem that can also be optimized
  - Where the system is subject to the boundary conditions:

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = 0$$

- The solution is an approximation $\mathbf{S}(t)$ which is a continuous function that is a cubic polynomial on sub-intervals $[t_n, t_{n+1}]$ of a mesh

$$a = t_0 < t_1 < \ldots < t_{n-1} < t_n = b$$

  - This approximation satisfies the boundary conditions, so that:

$$\mathbf{g}(\mathbf{S}(a), \mathbf{S}(b)) = 0$$

  - And it satisfies the differential equations (collocates) at both ends and the mid-point of each subinterval:

$$\begin{aligned}
\dot{\mathbf{S}}(t_n) &= \mathbf{f}(\mathbf{S}(t_n), t_n) \\
\dot{\mathbf{S}}((t_n + t_{n+1})/2) &= \mathbf{f}(\mathbf{S}((t_n + t_{n+1})/2), (t_n + t_{n+1})/2) \\
\dot{\mathbf{S}}(t_{n+1}) &= \mathbf{f}(\mathbf{S}(t_{n+1}), t_{n+1})
\end{aligned}$$

---

[13] Online reference

[14] Matlab help and BVP4C Tutorial

- Now constrain continuity in the solution at the mesh points $\Rightarrow$ converts problem to a series of nonlinear algebraic equations in the unknowns
  - Becomes a "root finding problem" that can be solved iteratively (Simpson's method).

- Inputs to BVP4C are functions that evaluate the differential equation $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t)$ and the residual of the boundary condition (e.g. $y_1(a) = 1$, $y_2(a) = y_1(b)$, and $y_3(b) = 0$):

$$\begin{aligned} \texttt{function} \quad & res = \texttt{bvpbc}(ya, yb) \\ res = [\quad & ya(1) - 1 \\ & ya(2) - yb(1) \\ & yb(3)]; \end{aligned}$$

- Redo example on page 4–15 using numerical techniques
  - Finite time LQR problem with $t_f = 10$

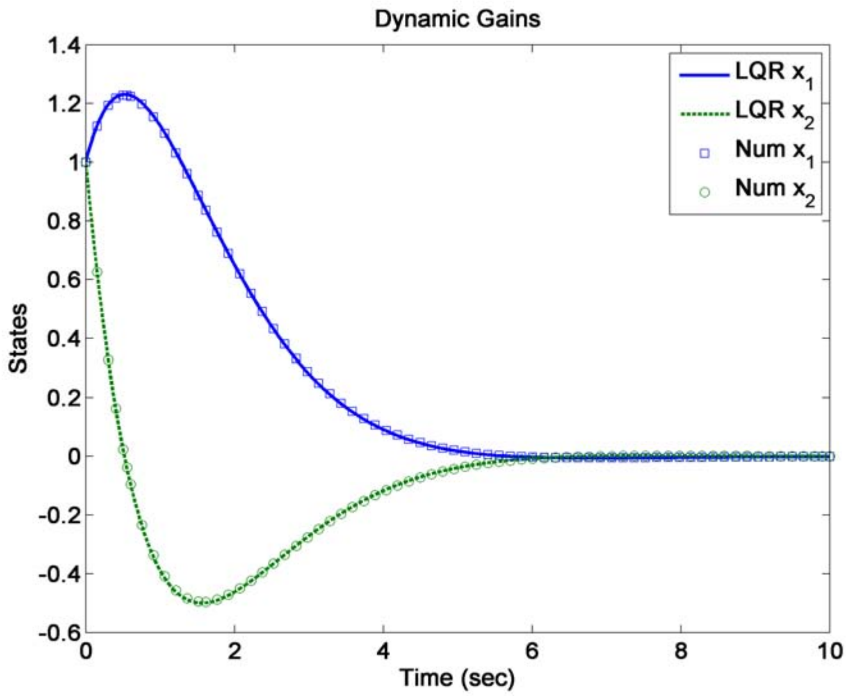


Figure 7.1: Results suggest a good comparison with the dynamic LQR result

TPBVP for LQR

```
function m = TPBVPlqr(p1,p2,p3)
global A B x0 Rxx Ruu Ptf
t_f=10;x0=[1 1]';
Rxx=p1;Ruu=p2;Ptf=p3;
solinit = bvpinit(linspace(0,t_f),@TPBVPlqrinit);
sol = bvp4c(@TPBVPlqrode,@TPBVPlqrbc,solinit);
time = sol.x;
state = sol.y([1 2],:);
adjoint = sol.y([3 4],:);
control = -inv(Ruu)*B'*sol.y([3 4],:);
m(1,:) = time;m([2 3],:) = state;m([4 5],:) = adjoint;m(6,:) = control;
%-------------------------------------------------------------------------
function dydt=TPBVPlqrode(t,y)
global A B x0 Rxx Ruu Ptf
dydt=[ A -B/Ruu*B'; -Rxx -A']*y;
%-------------------------------------------------------------------------
function res=TPBVPlqrbc(ya,yb)
global A B x0 Rxx Ruu Ptf
res=[ya(1) - x0(1);ya(2)-x0(2);yb(3:4)-Ptf*yb(1:2)];
%-------------------------------------------------------------------------
function v=TPBVPlqrinit(t)
global A B x0 b alp
v=[x0;1;0];
return

% 16.323 Spring 2007
% Jonathan How
% redo LQR example on page 4-15 using numerical approaches
clear all;close all;
set(0, 'DefaultAxesFontSize', 14, 'DefaultAxesFontWeight','demi')
set(0, 'DefaultTextFontSize', 14, 'DefaultTextFontWeight','demi')
%
global A B
Ptf=[0 0;0 4];Rxx=[1 0;0 0];Ruu=1;A=[0 1;0 -1];B=[0 1]';
tf=10;dt=.01;time=[0:dt:tf];
m=TPBVPlqr(Rxx,Ruu,Ptf); % numerical result

% integrate the P backwards for LQR result
P=zeros(2,2,length(time));K=zeros(1,2,length(time));
Pcurr=Ptf;
for kk=0:length(time)-1
  P(:,:,length(time)-kk)=Pcurr;
  K(:,:,length(time)-kk)=inv(Ruu)*B'*Pcurr;
  Pdot=-Pcurr*A-A'*Pcurr-Rxx+Pcurr*B*inv(Ruu)*B'*Pcurr;
  Pcurr=Pcurr-dt*Pdot;
end

% simulate the state
x1=zeros(2,1,length(time));xcurr1=[1 1]';
for kk=1:length(time)-1
  x1(:,:,kk)=xcurr1;
  xdot1=(A-B*K(:,:,kk))*x1(:,:,kk);
  xcurr1=xcurr1+xdot1*dt;
end

figure(3);clf
plot(time,squeeze(x1(1,1,:)),time,squeeze(x1(2,1,:)),'--','LineWidth',2),
xlabel('Time (sec)');ylabel('States');title('Dynamic Gains')
hold on;plot(m(1,:),m([2],:),'s',m(1,:),m([3],:),'o');hold off
legend('LQR x_1','LQR x_2','Num x_1','Num x_2')
print -dpng -r300 numreg2.png
```

Conversion

- BVP4C sounds good, but this standard form doesn't match many of the problems that we care about
  - In particular, free end time problems are excluded, because the time period is defined to be fixed $t \in [a, b]$

- Can convert our problems of interest into this standard form though using some pretty handy tricks.
  - U. Ascher and R. D. Russell, "Reformulation of Boundary Value Problems into "Standard" Form," *SIAM Review,* Vol. 23, No. 2, 238-254. Apr., 1981.

- Key step is to re-scale time so that $\tau = t/t_f$, then $\tau \in [0, 1]$.
  - Implications of this scaling are that the derivatives must be changed since $d\tau = dt/t_f$

$$\frac{d}{d\tau} = t_f \frac{d}{dt}$$

- Final step is to introduce a dummy state r that corresponds to t_f with the trivial dynamics $\dot{r} = 0$.
  - Now replace all instances of t_f in the necessary/boundary conditions for state r.
  - Optimizer will then just pick an appropriate constant for $r = t_f$

- Recall that our basic set of necessary conditions are, for $t \in [t_0, t_f]$

$$\boxed{\begin{aligned} \dot{\mathbf{x}} &= \mathbf{a}(\mathbf{x}, \mathbf{u}, t) \\ \dot{\mathbf{p}} &= -H_{\mathbf{x}}^T \\ H_{\mathbf{u}} &= 0 \end{aligned}}$$

- And we considered various boundary conditions $\mathbf{x}(t_0) = \mathbf{x}_0$, and:
  – If t_f is free: $h_t + g + \mathbf{p}^T \mathbf{a} = h_t + H(t_f) = 0$
  – If $\mathbf{x}_i(t_f)$ is fixed, then $\mathbf{x}_i(t_f) = x_{i_f}$
  – If $\mathbf{x}_i(t_f)$ is free, then $\mathbf{p}_i(t_f) = \dfrac{\partial h}{\partial x_i}(t_f)$

- Then

$$\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u}, t) \quad \Rightarrow \quad \mathbf{x}' = t_f \mathbf{a}(\mathbf{x}, \mathbf{u}, \tau)$$

and

$$\dot{\mathbf{p}} = -H_{\mathbf{x}}^T \quad \Rightarrow \quad \mathbf{p}' = -t_f H_{\mathbf{x}}^T$$

Example: 7–1

- Revisit example on page 6–6
- Linear system with performance/time weighting and free end time
  – Necessary conditions are:

$$\begin{aligned}\dot{\mathbf{x}} &= A\mathbf{x} + Bu\\ \dot{\mathbf{p}} &= -A^T\mathbf{p}\\ 0 &= bu + \begin{bmatrix} 0 & 1 \end{bmatrix}\mathbf{p}\end{aligned}$$

  with state conditions

$$\begin{aligned}\mathbf{x}_1(0) &= 10\\ \mathbf{x}_2(0) &= 0\\ \mathbf{x}_1(t_f) &= 0\\ \mathbf{x}_2(t_f) &= 0\\ -0.5bu^2(t_f) + \alpha t_f &= 0\end{aligned}$$

- Define the state of interest $\mathbf{z} = [\mathbf{x}^T \;\; \mathbf{p}^T \;\; r]^T$ and note that

$$\begin{aligned}\frac{d\mathbf{z}}{d\tau} &= t_f\frac{d\mathbf{z}}{dt}\\ &= \mathbf{z}_5 \begin{bmatrix} A & -B\begin{bmatrix} 0 & 1 \end{bmatrix}/b & 0\\ 0 & -A^T & 0\\ 0 & 0 & 0 \end{bmatrix}\mathbf{z}\\ \Rightarrow \mathbf{z}' &= f(\mathbf{z}) \quad \text{which is nonlinear}\end{aligned}$$

  with BC:

$$\begin{aligned}\mathbf{z}_1(0) &= 10\\ \mathbf{z}_2(0) &= 0\\ \mathbf{z}_1(1) &= 0\\ \mathbf{z}_2(1) &= 0\\ \frac{-0.5}{b}\mathbf{z}_4^2(1) + \alpha\mathbf{z}_5(1) &= 0\end{aligned}$$

- Code given on following pages
  - Note – it is not particularly complicated
  - Solution time/iteration count is a strong function of the initial solution – not a particularly good choice for $\mathbf{p}$ is used here

- Analytic solution gave $t_f = (1800b/\alpha)^{1/5}$
  - Numerical result give close agreement in prediction of the final time

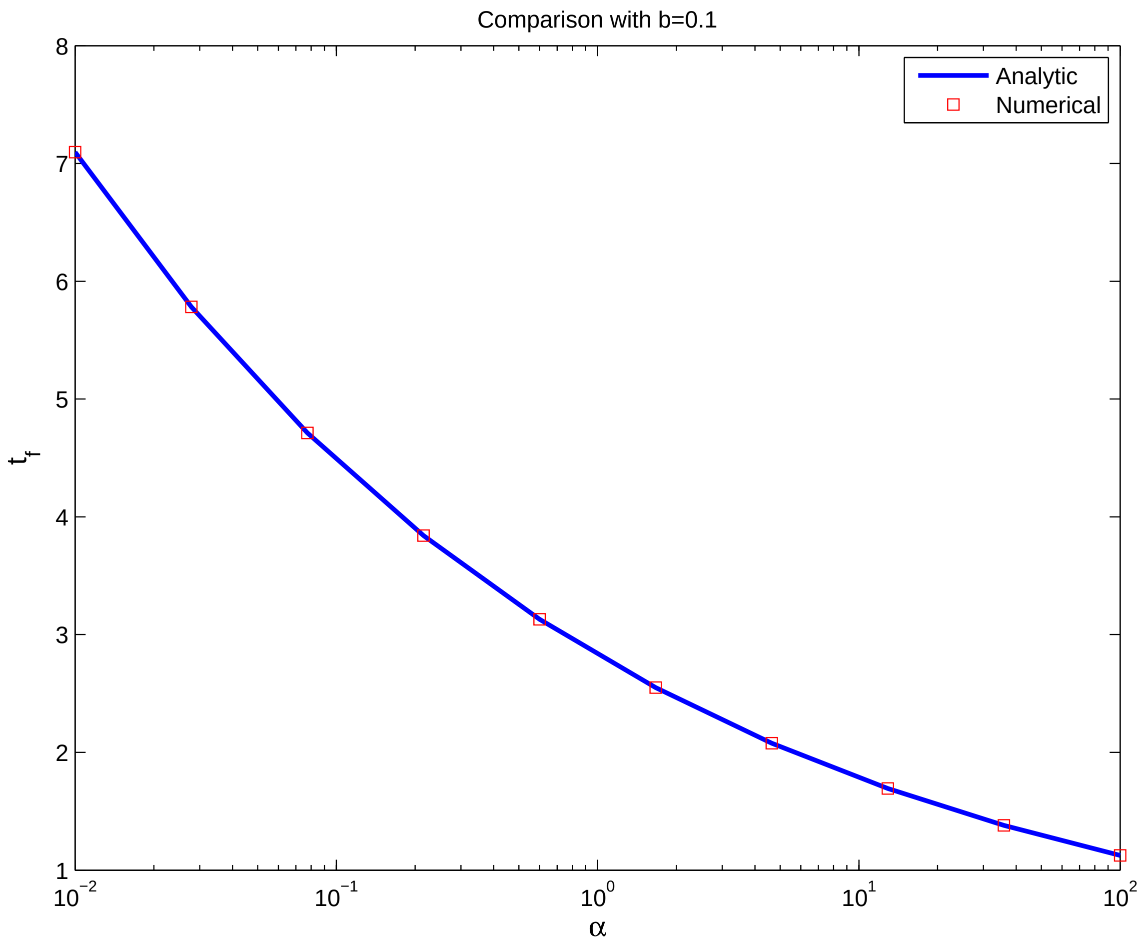


Figure 7.2: Comparison of the predicted completion times for the maneuver

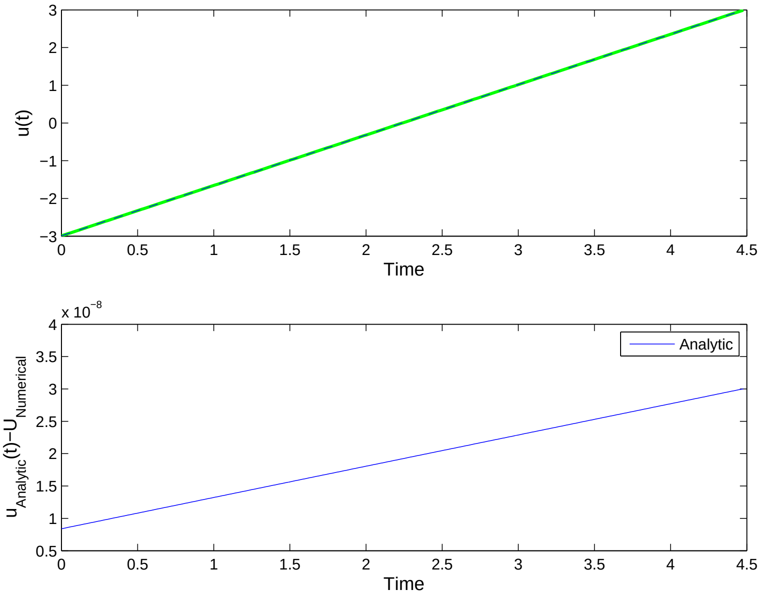


Figure 7.3: Control Inputs

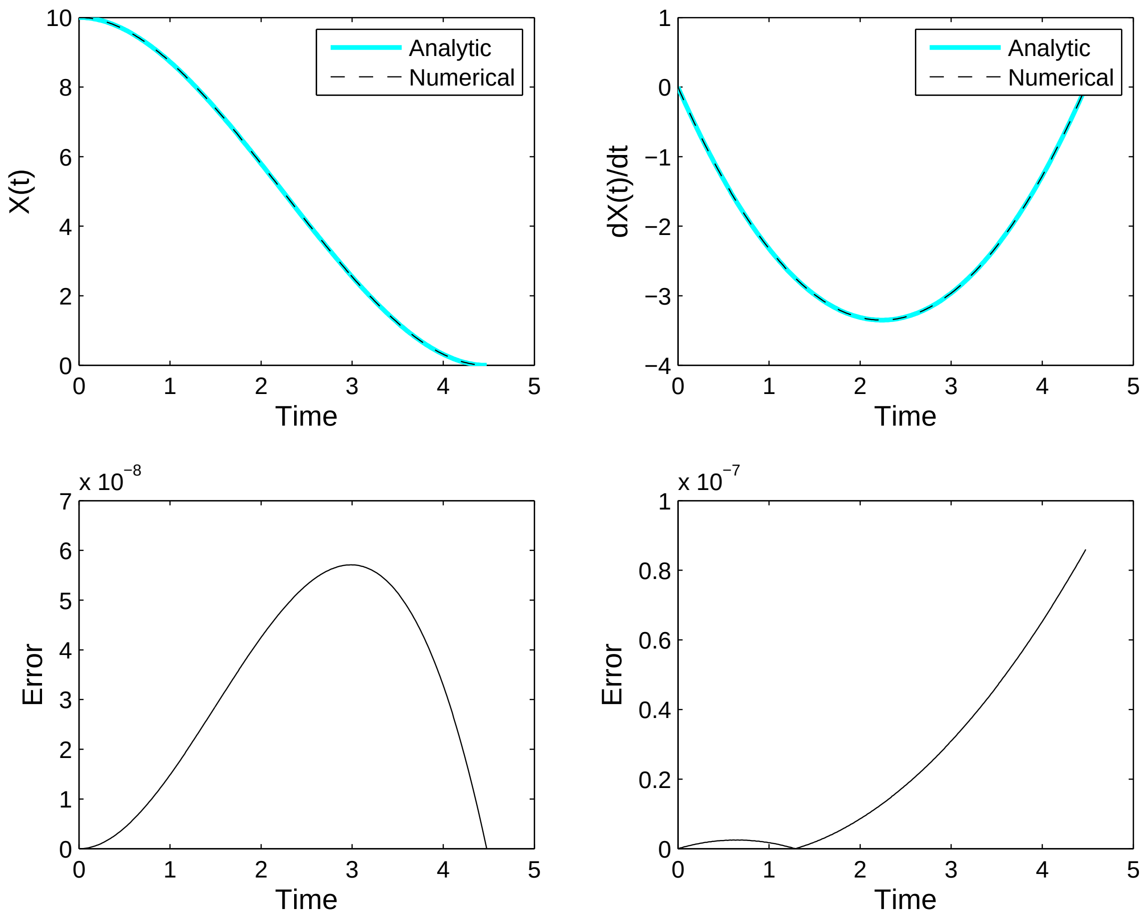


Figure 7.4: State response

TPBVP

```
function m = TPBVP(p1,p2)
% 16.323 Spring 2007
% Jonathan How
%
global A B x0 b alp

A=[0 1;0 0];
B=[0 1]';
x0=[10 0]';
b=p1;
alp=p2;

solinit = bvpinit(linspace(0,1),@TPBVPinit);
sol = bvp4c(@TPBVPode,@TPBVPbc,solinit);

time = sol.y(5)*sol.x;
state = sol.y([1 2],:);
adjoint = sol.y([3 4],:);
control = -(1/b)*sol.y(4,:);
m(1,:) = time;
m([2 3],:) = state;
m([4 5],:) = adjoint;
m(6,:) = control;

%--------------------------------------------------------------------------
function dydt=TPBVPode(t,y)
global A B x0 b alp
dydt=y(5)*[ A -B*[0 1]/b zeros(2,1); zeros(2,2) -A' zeros(2,1);zeros(1,5)]*y;

%--------------------------------------------------------------------------
function res=TPBVPbc(ya,yb)
global A B x0 b alp
res=[ya(1) - x0(1);ya(2)-x0(2);yb(1);yb(2);-0.5*yb(4)^2/b+ alp*yb(5)];

%--------------------------------------------------------------------------
function v=TPBVPinit(t)
global A B x0 b alp
v=[x0;1;0;1];

return

```

TPBVP Main

```
% 16.323 Spring 2007
% Jonathan How
% TPmain.m
%
b=0.1;
%alp=[.05 .1 1 10 20];
alp=logspace(-2,2,10);
t=[];
for alpha=alp
    m=TPBVP(b,alpha);
    t=[t;m(1,end)];
end

figure(1);clf
semilogx(alp,(1800*b./alp).^0.2,'-','Linewidth',2)
hold on;semilogx(alp,t,'rs');hold off
xlabel('\alpha','FontSize',12);ylabel('t_f','FontSize',12)
legend('Analytic','Numerical')
title('Comparison with b=0.1')
print -depsc -f1 TPBVP1.eps;jpdf('TPBVP1')

% code from opt1.m on the analytic solution
b=0.1;alpha=0.1;
m=TPBVP(b,alpha);
tf=(1800*b/alpha)^0.2;
c1=120*b/tf^3;
c2=60*b/tf^2;
u=(-c2+c1*m(1,:))/b;
A=[0 1;0 0];B=[0 1]';C=eye(2);D=zeros(2,1);G=ss(A,B,C,D);X0=[10 0]';
[y3,t3]=lsim(G,u,m(1,:),X0);

figure(2);clf
subplot(211)
plot(m(1,:),u,'g-','LineWidth',2);
xlabel('Time','FontSize',12);ylabel('u(t)','FontSize',12)
hold on;plot(m(1,:),m(6,:),'--');hold off
subplot(212)
plot(m(1,:),abs(u-m(6,:)),'-')
xlabel('Time','FontSize',12)
ylabel('u_{Analytic}(t)-U_{Numerical}','FontSize',12)
legend('Analytic','Numerical')
print -depsc -f2 TPBVP2.eps;jpdf('TPBVP2')

figure(3);clf
subplot(221)
plot(m(1,:),y3(:,1),'c-','LineWidth',2);
xlabel('Time','FontSize',12);ylabel('X(t)','FontSize',12)
hold on;plot(m(1,:),m([2],:),'k--');hold off
legend('Analytic','Numerical')
subplot(222)
plot(m(1,:),y3(:,2),'c-','LineWidth',2);
xlabel('Time','FontSize',12);ylabel('dX(t)/dt','FontSize',12)
hold on;plot(m(1,:),m([3],:),'k--');hold off
legend('Analytic','Numerical')
subplot(223)
plot(m(1,:),abs(y3(:,1)-m(2,:)'),'k-')
xlabel('Time','FontSize',12);ylabel('Error','FontSize',12)
subplot(224)
plot(m(1,:),abs(y3(:,2)-m(3,:)'),'k-')
xlabel('Time','FontSize',12);ylabel('Error','FontSize',12)
print -depsc -f3 TPBVP3.eps;jpdf('TPBVP3')
```

Zermelo's Problem

- Simplified dynamics of a UAV flying in a horizontal plane can be modeled as:

$$\begin{aligned}\dot{x}(t) &= V\cos\theta(t)\\ \dot{y}(t) &= V\sin\theta(t) + w\end{aligned}$$

where $\theta(t)$ is the heading angle (control input) with respect to the x axis, V is the speed.

- Objective: fly from point A to B in minimum time:

$$\min J = \int_0^{t_f} (1)dt$$

where t_f is free.

– Initial conditions are:

$$x(0) = x_0 \qquad\qquad y(0) = y_0$$

– Final conditions are:

$$x(t_f) = x_1 \qquad\qquad y(t_f) = y_1$$

- Apply the standard necessary conditions with

$$H = 1 + p_1 V(\cos\theta(t)) + p_2(V\sin\theta(t) + w)$$

$$\boxed{\begin{aligned} \dot{\mathbf{x}} &= \mathbf{a}(\mathbf{x}, \mathbf{u}, t) \\ \dot{\mathbf{p}} &= -H_{\mathbf{x}}^T \\ H_{\mathbf{u}} &= 0 \end{aligned}}$$

$$\begin{aligned} \dot{x}(t) &= V\cos\theta(t) \\ \dot{y}(t) &= V\sin\theta(t) + w \end{aligned}$$

$$\begin{aligned} \dot{p_1}(t) &= 0 \\ \dot{p_2}(t) &= 0 \\ 0 &= -p_1 \sin\theta(t) + p_2 \cos\theta(t) \end{aligned}$$

  – Then add extra state for the time.

- Since t_f is free, must add terminal condition that $H(t_f) = 0$, which gives a total of 5 conditions (2 initial, 3 terminal).

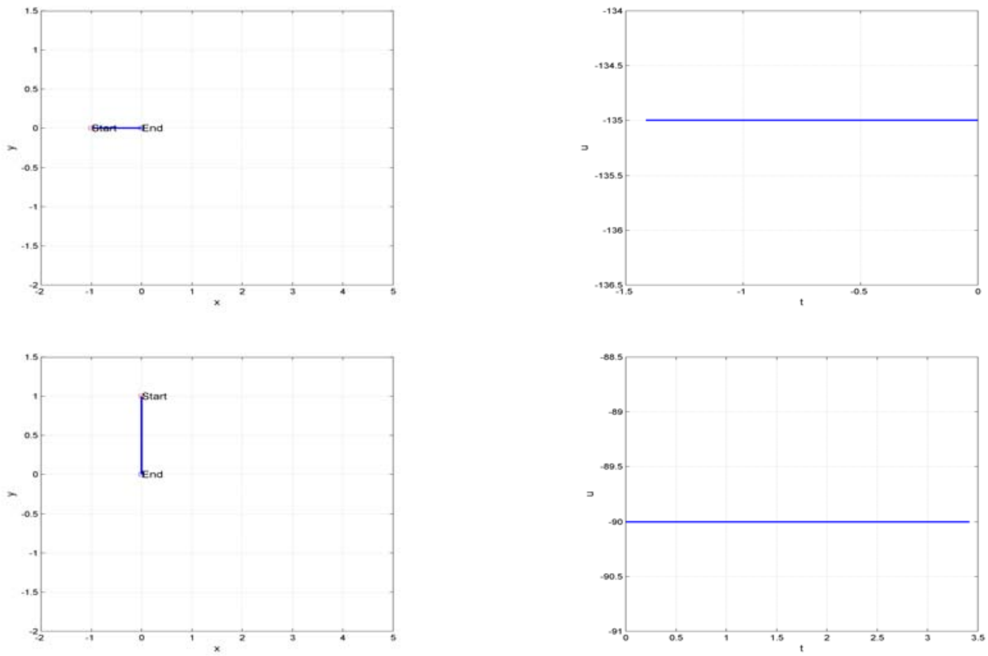


Figure 7.5: Zermelo examples

TPBVPZermelo

```
function m = TPBVPzermelo(p1,p2)
global x0 x1 V w

solinit = bvpinit(linspace(0,1),@TPBVPinit);
sol = bvp6c(@TPBVPode,@TPBVPbc,solinit);

time = sol.y(5)*sol.x;
state = sol.y([1 2],:);
adjoint = sol.y([3 4],:);
control = atan2(-sol.y([4],:),-sol.y([3],:));

m(1,:) = time;
m([2 3],:) = state;
m([4 5],:) = adjoint;
m(6,:) = control;
return

%------------------------------------------------------------------------
function dydt=TPBVPode(t,y)
global x0 x1 V w

% x y p1 p2 t
% minimizing form
sinth=-y(4)/sqrt(y(3)^2+y(4)^2);
costh=-y(3)/sqrt(y(3)^2+y(4)^2);

dydt=y(5)*[V*costh ; V*sinth+w;0;0;0];
%------------------------------------------------------------------------
function res=TPBVPbc(ya,yb)
global x0 x1 V w
% x y p1 p2 t
% minimizing form
costhb=-yb(3)/sqrt(yb(3)^2+yb(4)^2);
sinthb=-yb(4)/sqrt(yb(3)^2+yb(4)^2);

res=[ya(1) - x0(1);ya(2)-x0(2);
     yb(1) - x1(1);yb(2)-x1(2);
    1+V*costhb*yb(3)+V*(sinthb+w)*yb(4)];

%------------------------------------------------------------------------
function v=TPBVPinit(t)
global x0 x1 V w
%v=[x0;-1;-1;norm(x1-x0)/(V-w)];
v=[x0;1;1;norm(x1-x0)/(V-w)];
return

clear all
global x0 x1 V w
w=1/sqrt(2);
x0=[-1 0]';x1=[0 0]';V = 1;
mm=TPBVPzermelo;

figure(1);clf
plot(mm(2,:),mm([3],:),'LineWidth',2);axis('square');grid on
axis([-2 5 -2 1.5 ])
xlabel('x','FontSize',12);ylabel('y','FontSize',12);
hold on;
plot(x0(1),x0(2),'rs');plot(x1(1),x1(2),'bs');
text(x0(1),x0(2),'Start','FontSize',12)
text(x1(1),x1(2),'End','FontSize',12)
hold off

figure(2);clf
plot(mm(1,:),180/pi*mm([6],:),'LineWidth',2);grid on;axis('square')
xlabel('t','FontSize',12);ylabel('u','FontSize',12);

print -dpng -r300 -f1 BVP_zermelo.png;
```

```
print -dpng -r300 -f2 BVP_zermelo2.png;

clear all
global x0 x1 V w
w=1/sqrt(2);
x0=[0 1]';x1=[0 0]';V = 1;
mm=TPBVPzermelo;

figure(1);clf
plot(mm(2,:),mm([3],:),'LineWidth',2);axis('square');grid on
axis([-2 5 -2 1.5 ])
xlabel('x','FontSize',12);ylabel('y','FontSize',12);
hold on;
plot(x0(1),x0(2),'rs');plot(x1(1),x1(2),'bs');
text(x0(1),x0(2),'Start','FontSize',12)
text(x1(1),x1(2),'End','FontSize',12)
hold off

figure(2);clf
plot(mm(1,:),180/pi*mm([6],:),'LineWidth',2);grid on;axis('square')
xlabel('t','FontSize',12);ylabel('u','FontSize',12);

print -dpng -r300 -f1 BVP_zermelo3.png;
print -dpng -r300 -f2 BVP_zermelo4.png;
```

Orbit Raising Example

- **Goal:** (Bryson page 66) determine the maximum radius orbit transfer in a given time t_f assuming a constant thrust rocket (thrust T).[15]
  - Must find the thrust direction angle $\phi(t)$
  - Assume a circular orbit for the initial and final times

- **Nomenclature:**
  - r – radial distance from attracting center, with gravitational constant μ
  - v, u tangential, radial components of the velocity
  - m mass of s/c, and $\dot{m}$ is the fuel consumption rate (constant)

- **Problem:** find $\phi(t)$ to maximize $r(t_f)$ subject to:

$$\begin{aligned} \text{Dynamics}: \quad \dot{r} &= u \\ \dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T\sin\phi}{m_0 - |\dot{m}|t} \\ \dot{v} &= -\frac{uv}{r} + \frac{T\cos\phi}{m_0 - |\dot{m}|t} \end{aligned}$$

with initial conditions

$$r(0) = r_0 \qquad u(0) = 0 \qquad v(0) = \sqrt{\frac{\mu}{r_0}}$$

and terminal conditions

$$u(t_f) = 0 \qquad v(t_f) - \sqrt{\frac{\mu}{r(t_f)}} = 0$$

- With $\mathbf{p}^T = [p_1 \;\; p_2 \;\; p_3]$ this gives the Hamiltonian (since $g = 0$)

$$H = \mathbf{p}^T \begin{bmatrix} u \\ \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T\sin\phi}{m_0 - |\dot{m}|t} \\ -\frac{uv}{r} + \frac{T\cos\phi}{m_0 - |\dot{m}|t} \end{bmatrix}$$

[15] Thanks to Geoff Huntington

– Then $H_{\mathbf{u}} = 0$ with $\mathbf{u}(t) = \phi(t)$ gives

$$p_2 \left(\frac{T\cos\phi}{m_0 - |\dot{m}|t} \right) + p_3 \left(\frac{-T\sin\phi}{m_0 - |\dot{m}|t} \right) = 0$$

which gives that

$$\tan\phi = \frac{p_2(t)}{p_3(t)}$$

that can be solved for the control input given the costates.

- Note that this is a problem of the form on 6–6, with

$$\mathbf{m} = \begin{bmatrix} u(t_f) \\ v(t_f) - \sqrt{\frac{\mu}{r(t_f)}} \end{bmatrix} = 0$$

which gives

$$w = -r + \nu_1 u(t_f) + \nu_2 \left(v(t_f) - \sqrt{\frac{\mu}{r(t_f)}} \right)$$

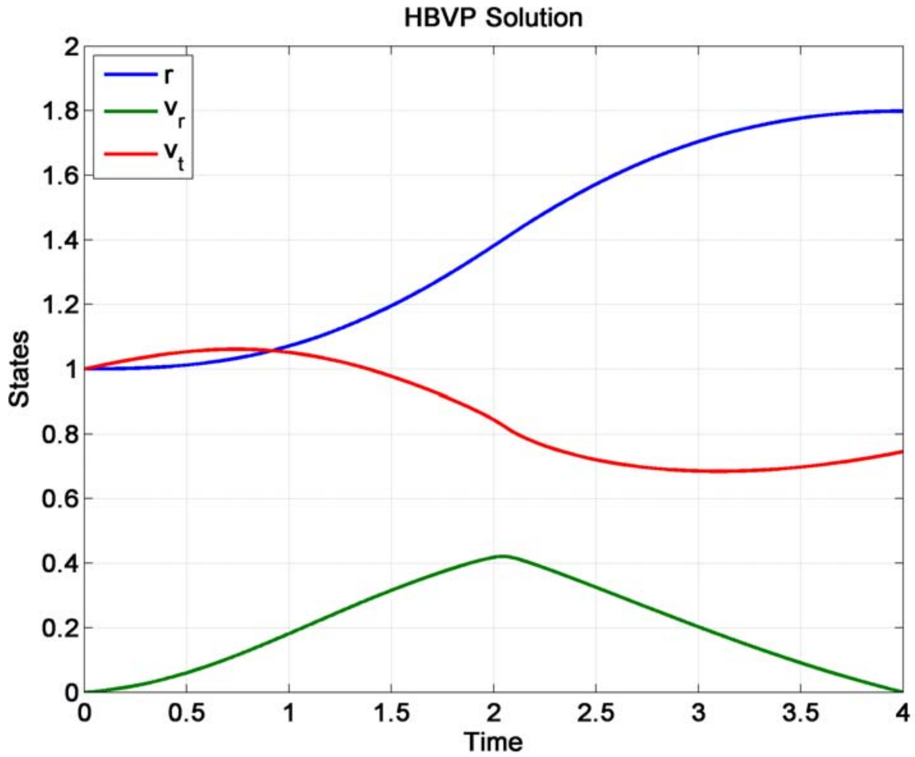
- Since the first state r is not specified at the final time, must have that

$$p_1(t_f) = \frac{\partial w}{\partial r}(t_f) = -1 + \frac{\nu_2}{2}\sqrt{\frac{\mu}{r(t_f)^3}}$$

– And note that

$$p_3(t_f) = \frac{\partial w}{\partial v}(t_f) = \nu_2$$

which gives ν_2 in terms of the costate.



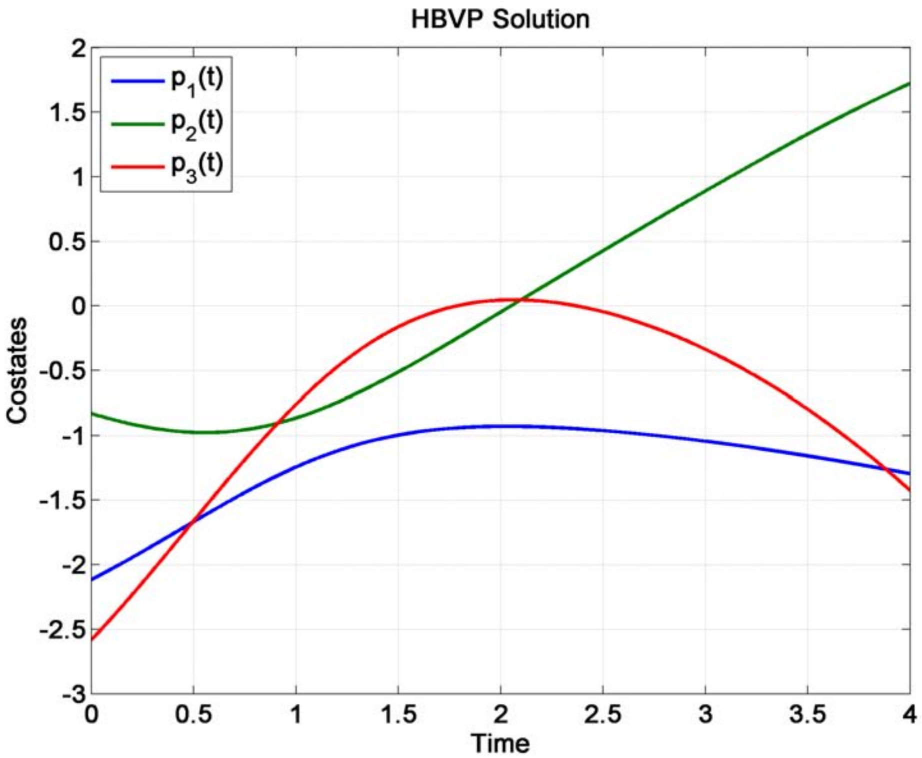


Figure 7.6: Orbit raising examples

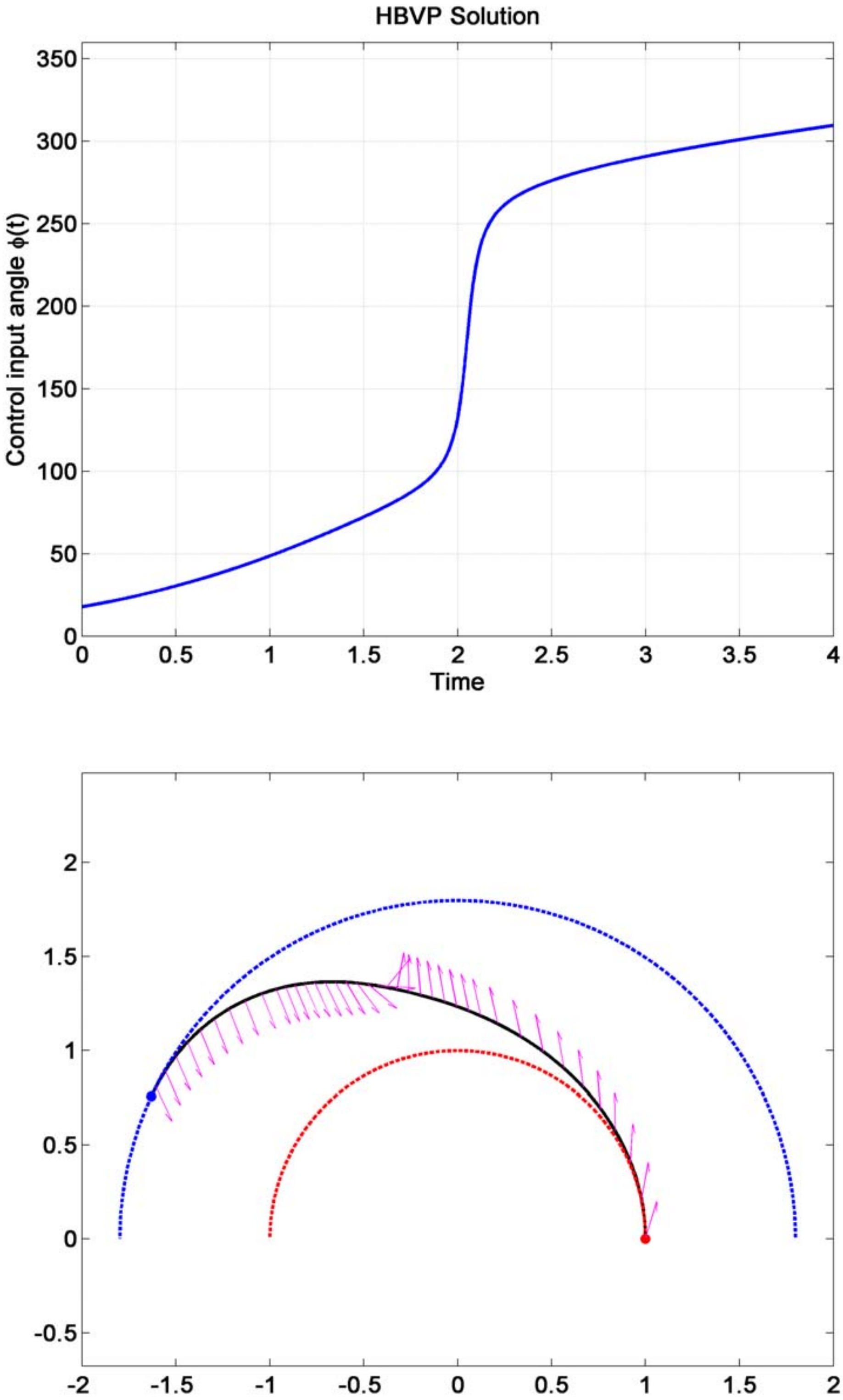


Figure 7.7: Orbit raising examples

Orbit Raising

```
%orbit_bvp_how created by Geoff Huntington 2/21/07
%Solves the Hamiltonian Boundary Value Problem for the orbit-raising optimal
%control problem (p.66 Bryson & Ho). Computes the solution using BVP4C
%Invokes subroutines orbit_ivp and orbit_bound
clear all;%close all;
set(0, 'DefaultAxesFontSize', 14, 'DefaultAxesFontWeight','demi')
set(0, 'DefaultTextFontSize', 14, 'DefaultTextFontWeight','demi')

%Fixed final time %Tf = 3.3155;
Tf = 4;
four=0; % not four means use bvp6c

%Constants
global mu m0 m1 T
mu=1;  m0=1; m1=-0.07485; T= 0.1405;
%mu=1;  m0=1; m1=-.2; T= 0.1405;

%Create initial Guess
n=100;
y = [ones(1,n);  %r
     zeros(1,n); %vr
     ones(1,n);  %vt
    -ones(1,n);  %lambda_r
    -ones(1,n);  %lambda_vr
    -ones(1,n)]; %lambda_vt
x = linspace(0,Tf,n); %time
solinit.x = x;solinit.y = y;
%Set optimizer options
tol = 1E-10;
options = bvpset('RelTol',tol,'AbsTol',[tol tol tol tol tol tol],'Nmax', 2000);

%Solve
if four
    sol = bvp4c(@orbit_ivp,@orbit_bound,solinit,options);
    Nstep=40;
else
    sol = bvp6c(@orbit_ivp,@orbit_bound,solinit,options);
    Nstep=30;
end

%Plot results
figure(1);clf
plot(sol.x,sol.y(1:3,:),'LineWidth',2)
legend('r','v_r','v_t','Location','NorthWest')
grid on;
axis([0 4 0 2])
title('HBVP Solution')
xlabel('Time');ylabel('States')

figure(2);clf
plot(sol.x,sol.y(4:6,:),'LineWidth',2)
legend('p_1(t)','p_2(t)','p_3(t)','Location','NorthWest')
grid on;
axis([0 4 -3 2])
title('HBVP Solution')
xlabel('Time');ylabel('Costates')

ang2=atan2(sol.y([5],:),sol.y([6],:))+pi;
figure(3);clf
plot(sol.x,180/pi*ang2','LineWidth',2)
grid on;
axis([0 4 0 360])
title('HBVP Solution')
xlabel('Time');ylabel('Control input angle \phi(t)')
norm([tan(ang2')-(sol.y([5],:)./sol.y([6],:))'])

print -f1 -dpng -r300 orbit1.png
print -f2 -dpng -r300 orbit2.png
print -f3 -dpng -r300 orbit3.png

% Code below adapted inpart from Bryson "Dynamic Optimization"
```

```
dt=diff(sol.x);
dth=(sol.y(3,1:end-1)./sol.y(1,1:end-1)).*dt; % \dot \theta = v_t/r
th=0+cumsum(dth');
pathloc=[sol.y(1,1:end-1)'.*cos(th) sol.y(1,1:end-1)'.*sin(th)];

figure(4);clf
plot(pathloc(:,1),pathloc(:,2),'k-','LineWidth',2)
hold on
zz=exp(sqrt(-1)*[0:.01:pi]');
r0=sol.y(1,1);rf=sol.y(1,end);
plot(r0*real(zz),r0*imag(zz),'r--','LineWidth',2)
plot(rf*real(zz),rf*imag(zz),'b--','LineWidth',2)
plot(r0,0,'ro','MarkerFace','r')
plot(rf*cos(th(end)),rf*sin(th(end)),'bo','MarkerFace','b')
fact=0.2;ep=ones(size(th,1),1)*pi/2+th-ang2(1:end-1)';
xt=pathloc(:,1)+fact*cos(ep); yt=pathloc(:,2)+fact*sin(ep);
for i=1:Nstep:size(th,1),
 pltarrow([pathloc(i,1);xt(i)],[pathloc(i,2);yt(i)],.05,'m','-');
end;
%axis([-1.6 1.6 -.1 1.8]);
axis([-2 2 -.1 1.8]);
axis('equal')
hold off

print -f4 -dpng -r300 orbit4.png;
```

```
function [dx] = orbit_ivp(t,x)
global mu m0 m1 T

%State
r = x(1);u = x(2);v = x(3);
lamr = x(4);lamu = x(5);lamv = x(6);

%Substitution for control
sinphi = -lamu./sqrt(lamu.^2+lamv.^2);
cosphi = -lamv./sqrt(lamu.^2+lamv.^2);

%Dynamic Equations
dr = u;
du = v^2/r - mu/r^2 + T*sinphi/(m0 + m1*t);
dv = -u*v/r + T*cosphi/(m0 + m1*t);

dlamr = -lamu*(-v^2/r^2 + 2*mu/r^3) - lamv*(u*v/r^2);
dlamu = -lamr + lamv*v/r;
dlamv = -lamu*2*v/r + lamv*u/r;

dx = [dr; du; dv; dlamr; dlamu; dlamv];
```

```
function [res] = orbit_bound(x,x2)
global mu m0 m1 T

%Initial State
r = x(1);u = x(2);v = x(3);
lamr = x(4);lamu = x(5);lamv = x(6);

%Final State
r2 = x2(1);u2 = x2(2);v2 = x2(3);
lamr2 = x2(4);lamu2 = x2(5);lamv2 = x2(6);

%Boundary Constraints
b1 = r - 1;
b2 = u;
b3 = v - sqrt(mu/r);
b4 = u2;
b5 = v2 - sqrt(mu/r2);
b6 = lamr2 + 1 - lamv2*sqrt(mu)/2/r2^(3/2);

%Residual
res = [b1;b2;b3;b4;b5;b6];
```