سایت اختصاصی
مهندسی کنترل

Solving optimal control problems with MATLAB — Indirect methods

Xuezhong Wang*

1 Introduction

The theory of optimal control has been well developed for over forty years. With the advances of computer technique, optimal control is now widely used in multi-disciplinary applications such as biological systems, communication networks and socio-economic systems etc. As a result, more and more people will benefit greatly by learning to solve the optimal control problems numerically. Realizing such growing needs, books on optimal control put more weight on numerical methods. In retrospect, [1] was the first and the "classic" book for studying the theory as well as many interesting cases (time-optimal, fuel-optimal and linear quadratic regulator(LQR) problems). Necessary conditions for various systems were derived and explicit solutions were given when possible. Later, [2] proved to be a concise yet excellent book with more engineering examples. One of the distinguish features of this book is that it introduced several iterative algorithms for solving problems numerically. More recently, [3] uses MATLAB to solve problems which is easier and more precise. However, the numerical methods covered in these books are insufficient for the wide range of problems emerging from various fields. Especially, for those problems with free final time and nonlinear dynamics.

This tutorial shows common routines in MATLAB to solve both fixed and free final time problems. Specifically, the following subjects are discussed with examples:

1. How to use Symbolic Math Toolbox to derive necessary conditions and solve for explicit solutions?

2. How to solve a fixed-final-time optimal control problem with steepest descent method?

*ISE. Dept., NCSU, Raleigh, NC 27695 (xwang10@ncsu.edu)

3. How to reformulate the original problem as a Boundary Value Problem (BVP) and solve it with *b*vp4c?

4. How to get 'good enough' solutions with *bvp4c* and under what conditions will *bvp4c* fail to find a solution?

It should be noted that all the routines (except the steepest descent method) discussed in this tutorial belong to the "indirect methods" category. This means that constraints on the controls and states are not considered.[1] In other words, the control can be solved in terms of states and costates and the problem is equivalently to a BVP. The reference of all the examples used in this tutorial are stated such that the results can be compared and verified.

2 Optimal control problems with fixed-final-time

In most books [1] [2], it is free-final-time problem that being tackled first to derive the necessary conditions for optimal control. Fixed-final-time problems were treated as an equivalent variation with one more state for time. However, for numerical methods, fixed-final-time problems are the general form and we solve the free-final-time problem by converting it into a fixed-final-time problem. The reason is simple: when dealing with optimal control problems, it is inevitable to do numerical integration (either by indirect or direct methods). Therefore, a time interval must be specified for these methods.

The first problem is from [2], Example 5.1-1 from page 198 to 202. The routine deployed in this example shows how to derive necessary conditions and solve for the solutions with MATLAB Symbolic Math Toolbox.

Example 1 *The system*

$$\dot{x}_1(t) = x_2(t) \tag{1}$$
$$\dot{x}_2(t) = -x_2(t) + u(t) \tag{2}$$

(a) Consider the performance measure:

$$J(u) = \int_0^{t_f} \frac{1}{2} u^2(t)\, \mathrm{d}t$$

[1] The last example implements a simple constraint on the control to show limitations of *bvp4c* solver.

with boundary conditions:

$$\mathbf{x}(0) = \mathbf{0} \qquad \mathbf{x}(2) = \begin{bmatrix} 5 & 2 \end{bmatrix}^T$$

(b) Consider the performance measure:

$$J(u) = \frac{1}{2}(x_1(2) - 5)^2 + \frac{1}{2}(x_2(2) - 2)^2 + \int_0^2 \frac{1}{2}u^2(t)\,\mathrm{d}t$$

with boundary conditions:

$$\mathbf{x}(0) = \mathbf{0} \qquad \mathbf{x}(2) \text{ is free}$$

(c) Consider the performance measure as in (a) with following boundary conditions:

$$\mathbf{x}(0) = \mathbf{0} \qquad x_1(2) + 5x_2(2) = 15$$

The first step is to form the Hamiltonian and apply necessary conditions for optimality. With MATLAB, this can be done as follows:

```
% State equations
syms x1 x2 p1 p2 u;
Dx1 = x2;
Dx2 = -x2 + u;

% Cost function inside the integral
syms g;
g = 0.5*u^2;

% Hamiltonian
syms p1 p2 H;
H = g + p1*Dx1 + p2*Dx2;

% Costate equations
Dp1 = -diff(H,x1);
Dp2 = -diff(H,x2);

% solve for control u
du = diff(H,u);
sol_u = solve(du, 'u');
```

The MATLAB commands we used here are `diff` and `solve`. `diff` differentiates a symbolic expression and `solve` gives symbolic solution to algebraic

equations. For more details about Symbolic Math Toolbox, please refer to [5]. Applying the necessary conditions for optimality we get two equations:[2]

$$\dot{p}_i^* = -\frac{\partial H}{\partial x_i^*} \tag{3}$$

$$\frac{\partial H}{\partial u^*} = 0 \tag{4}$$

The first equation gives costate equations. From the second equation, we solve for control u in terms of states and costates. The second step is to substitute u from (4) back to the state and costate equations to get a set of $2n$ first-order ordinary differential equations (ODE's). A solution (with $2n$ arbitrary coefficients) can be obtained by using the `dsolve` command without any boundary conditions. The symbolic solution looks different from the one in [2]. By simplifying the expression and rearrange the arbitrary coefficients, it is not difficult to see that the two are the same.

```
% Substitute u to state equations
Dx2 = subs(Dx2, u, sol_u);

% convert symbolic objects to strings for using 'dsolve'
eq1 = strcat('Dx1=',char(Dx1));
eq2 = strcat('Dx2=',char(Dx2));
eq3 = strcat('Dp1=',char(Dp1));
eq4 = strcat('Dp2=',char(Dp2));

sol_h = dsolve(eq1,eq2,eq3,eq4);
```

As stated in [2], the differences of the three cases in this problem are merely the boundary conditions. For *(a)*, the arbitrary coefficients can be determined by supplying the $2n$ boundary conditions to `dsovle`:

```
%    case a: (a) x1(0)=x2(0)=0; x1(2) = 5; x2(2) = 2;
conA1 = 'x1(0) = 0';
conA2 = 'x2(0) = 0';
conA3 = 'x1(2) = 5';
conA4 = 'x2(2) = 2';
sol_a = dsolve(eq1,eq2,eq3,eq4,conA1,conA2,conA3,conA4);
```

Again the solutions given by MATLAB and [2] look different from each other. Yet Figure 1 shows that the two are in fact equivalent. For all the figures in this problem, * represent state trajectory from [2] while symbolic solution from MATLAB is plotted with a continuous line.

[2]We use * to indicate the optimal state trajectory or control.

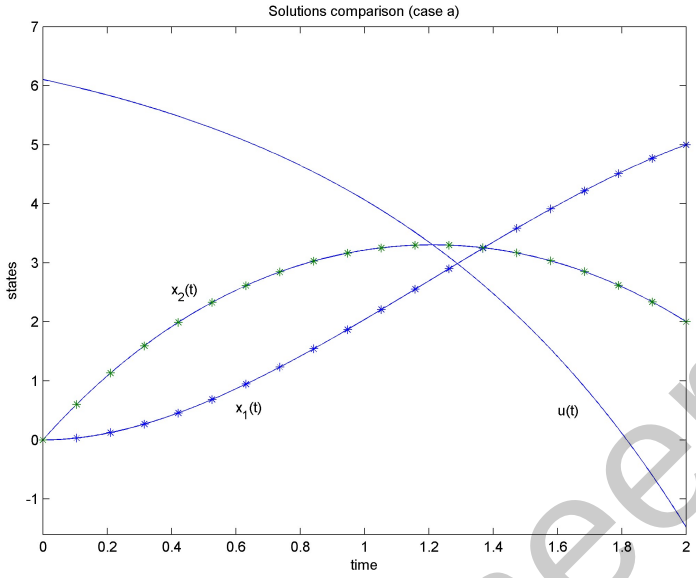

Figure 1: Trajectories for Example 1 *(a)*

For case *(b)*, we substitute $t_0 = 0,\ t_f = 2$ in the the general solution to get a set of 4 algebraic equations with 4 arbitrary coefficients. Then the four boundary conditions are supplied to determine these coefficients. `sol_b` is a structure consists of four coefficients returned by `solve`. We get the final symbolic solution to the ODE's by substituting these coefficients into the general solution. Figure 2 shows the results of both solutions from MATLAB and [2].

It should be noted that we cannot get the solution by directly supplying the boundary conditions with `dsolve` as we did in case *(a)*. Because the latter two boundary conditions: $p_1^*(2) = x_1^*(t) - 5,\ p_2^*(2) = x_2^*(t) - 2$ replaces costates $p_1,\ p_2$ with states $x_1,\ x_2$ in the ODE's which resulted in more ODE's than variables.

```
%    case b: (a) x1(0)=x2(0)=0; p1(2) = x1(2) - 5; p2(2) = x2(2) -2;
eq1b = char(subs(sol_h.x1,'t',0));
eq2b = char(subs(sol_h.x2,'t',0));
eq3b = strcat(char(subs(sol_h.p1,'t',2)),...
              '=',char(subs(sol_h.x1,'t',2)),'-5');
eq4b = strcat(char(subs(sol_h.p2,'t',2)),...
              '=',char(subs(sol_h.x2,'t',2)),'-2');

sol_b = solve(eq1b,eq2b,eq3b,eq4b);
```

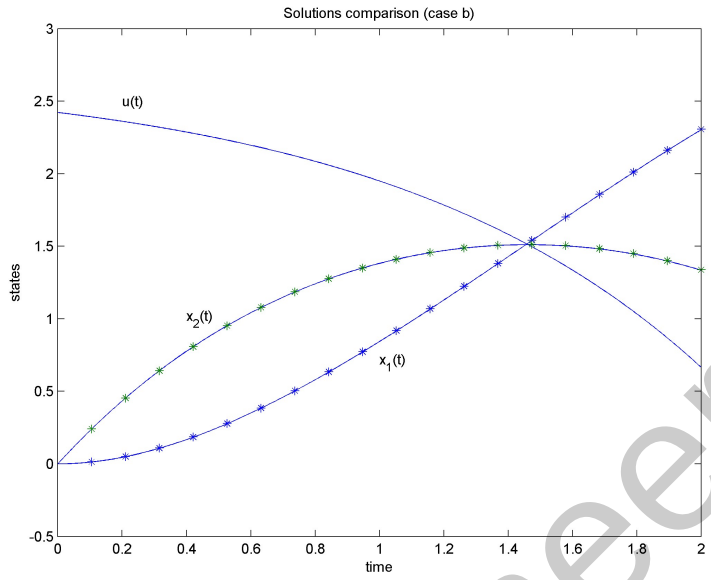

Figure 2: Trajectories for Example 1 *(b)*

```
% Substitute the coefficients
C1 = double(sol_b.C1);
C2 = double(sol_b.C2);
C3 = double(sol_b.C3);
C4 = double(sol_b.C4);
sol_b2 = struct('x1',{subs(sol_h.x1)},'x2',{subs(sol_h.x2)}, ...
                'p1',{subs(sol_h.p1)},'p2',{subs(sol_h.p2)});
```

Case *(c)* is almost the same as case *(b)* with slightly different boundary conditions. From [2], the boundary conditions are: $x_1^*(2) + 5x_2^*(2) = 15$, $p_2^*(2) = 5p_1^*(2)$. Figure 3 shows the results of both solutions from MATLAB and [2].

```
% case c: x1(0)=x2(0)=0;x1(2)+5*x2(2)=15;p2(2)= 5*p1(2);
eq1c = char(subs(sol_h.x1,'t',0));
eq2c = char(subs(sol_h.x2,'t',0));
eq3c = strcat(char(subs(sol_h.p2,'t',2)),...
              '-(',char(subs(sol_h.p1,'t',2)),')*5');
eq4c = strcat(char(subs(sol_h.x1,'t',2)),...
              '+(',char(subs(sol_h.x2,'t',2)),')*5-15');
sol_c = solve(eq1c,eq2c,eq3c,eq4c);
% Substitute the coefficients
C1 = double(sol_c.C1);
```

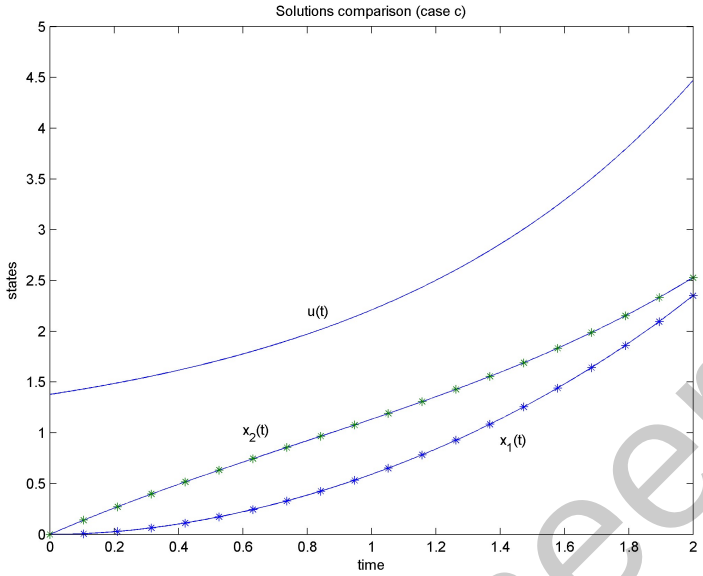

Figure 3: Trajectories for Example 1 *(c)*

```
C2 = double(sol_c.C2);
C3 = double(sol_c.C3);
C4 = double(sol_c.C4);
sol_c2 = struct('x1',{subs(sol_h.x1)},'x2',{subs(sol_h.x2)}, ...
                'p1',{subs(sol_h.p1)},'p2',{subs(sol_h.p2)});
```

If the state equations are complicated, it is usually impossible to derive explicit solutions. We depend on numerical methods to find the solutions. In the next example, we will illustrate two numerical routines: steepest descent method and convert to a BVP. The problem can be found in [2] page 338 to 339, Example 6.2-2.

Example 2 *The state equations for a continuous stirred-tank chemical reactor are given as:*

$$\begin{cases} \dot{x}_1(t) = -2[x_1(t) + 0.25] + [x_2(t) + 0.5]exp\left(\frac{25x_1(t)}{x_1(t)+2}\right) - [x_1(t) + 0.25]u(t) \\ \dot{x}_2(t) = 0.5 - x_2(t) - [x_2(t) + 0.5]exp\left(\frac{25x_1(t)}{x_1(t)+2}\right) \end{cases} \tag{5}$$

The flow of a coolant through a coil inserted in the reactor is to control the first-order, irreversible exothermic reaction taking place in the reactor. $x_1(t) = T(t)$ *is the deviation from the steady-state temperature and* $x_2(t) =$

$C(t)$ is the deviation from the steady-state concentration. $u(t)$ is the normalized control variable, representing the effect of coolant flow on the chemical reaction.
The performance measure to be minimized is:

$$J = \int_0^{0.78} [x_1^2(t) + x_2^2(t) + Ru^2(t)]\,\mathrm{d}t,$$

with the boundary conditions

$$\mathbf{x}(0) = \begin{bmatrix} 0.05 & 0 \end{bmatrix}^T, \ \mathbf{x}(t_f) \text{ is free}$$

First, we will implement the steepest descent method based on the scheme outlined in [2]. The algorithm consists of 4 steps:

1. Subdivide the interval $[t_0,\ t_f]$ into N equal subintervals and assume a piecewise-constant control $u^{(0)}(t) = u^{(0)}(t_k), \quad t \in [t_k,\ t_{k+1}] \quad k = 0, 1, \cdots, N-1$

2. Applying the assumed control $u^{(i)}$ to integrate the state equations from t_0 to t_f with initial conditions $\mathbf{x}(t_0) = \mathbf{x}_0$ and store the state trajectory $\mathbf{x}^{(i)}$.

3. Applying $u^{(i)}$ and $\mathbf{x}^{(i)}$ to integrate costate equations backward, i.e., from $[t_f,\ t_0]$. The "initial value" $\mathbf{p}^{(i)}(t_f)$ can be obtained by:

$$\mathbf{p}^{(i)}(t_f) = \frac{\partial h}{\partial \mathbf{x}}\left(\mathbf{x}^{(i)}(t_f)\right).$$

   Evaluate $\partial H^{(i)}(t)/\partial u$, $t \in [t_0,\ t_f]$ and store this vector.

4. If

$$\left\| \frac{\partial H^{(i)}}{\partial u} \right\| \ \leq \ \gamma \tag{6}$$

$$\left\| \frac{\partial H^{(i)}}{\partial u} \right\|^2 \ \equiv \ \int_{t_0}^{t_f} \left[\left\| \frac{\partial H^{(i)}}{\partial u} \right\| \right]^T \left[\left\| \frac{\partial H^{(i)}}{\partial u} \right\| \right] \mathrm{d}t \tag{7}$$

   then stop the iterative procedure. Here γ is a preselected small positive constant used as a tolerance.
   If (6) is not satisfied, adjust the piecewise-constant control function by:

$$u^{(i+1)}(t_k) = u^{(i)}(t_k) - \tau \frac{\partial H^{(i)}}{\partial u}(t_k), \quad k = 0, 1, \cdots, N-1$$

   Replace $u^{(i)}$ by $u^{(i+1)}$ and return to step 2. Here, τ is the step size.

The main loop in MATLAB is as follows:

```
for i = 1:max_iteration
   % 1) start with assumed control u and move forward
   [Tx,X] = ode45(@(t,x) stateEq(t,x,u,Tu), [t0 tf], ...
                  initx, options);

   % 2) Move backward to get the trajectory of costates
   x1 = X(:,1); x2 = X(:,2);
   [Tp,P] = ode45(@(t,p) costateEq(t,p,u,Tu,x1,x2,Tx), ...
                  [tf t0], initp, options);
   p1 = P(:,1);
   % Important: costate is stored in reverse order. The dimension of
   % costates may also different from dimension of states
   % Use interploate to make sure x and p is aligned along the time axis
   p1 = interp1(Tp,p1,Tx);

   % Calculate deltaH with x1(t), x2(t), p1(t), p2(t)
   dH = pH(x1,p1,Tx,u,Tu);
   H_Norm = dH'*dH;

   % Calculate the cost function
   J(i,1) = tf*(((x1')*x1 + (x2')*x2)/length(Tx) + ...
                0.1*(u*u')/length(Tu));

   % if dH/du < epslon, exit
   if H_Norm < eps
       % Display final cost
       J(i,1)
       break;
   else
       % adjust control for next iteration
       u_old = u;
       u = AdjControl(dH,Tx,u_old,Tu,step);
   end;
end
```

Because the step size of *ode45* is not predetermined, interpolation is used to make sure $\mathbf{x}(t)$, $\mathbf{p}(t)$ and $u(t)$ are aligned along the time axis.

```
% State equations
function dx = stateEq(t,x,u,Tu)
dx = zeros(2,1);
u = interp1(Tu,u,t); % Interploate the control at time t
```

```
dx(1) = -2*(x(1) + 0.25) + (x(2) + 0.5)*exp(25*x(1)/(x(1)+2)) ...
        - (x(1) + 0.25).*u;
dx(2) = 0.5 - x(2) -(x(2) + 0.5)*exp(25*x(1)/(x(1)+2));

% Costate equations
function dp = costateEq(t,p,u,Tu,x1,x2,xt)
dp = zeros(2,1);
x1 = interp1(xt,x1,t);   % Interploate the state varialbes
x2 = interp1(xt,x2,t);
u = interp1(Tu,u,t);     % Interploate the control
dp(1) = p(1).*(u + exp((25*x1)/(x1 + 2)).*((25*x1)/(x1 + 2)^2 - ...
        25/(x1 + 2))*(x2 + 1/2) + 2) - ...
        2*x1 - p(2).*exp((25*x1)/(x1 + 2))*((25*x1)/(x1 + 2)^2 - ...
        25/(x1 + 2))*(x2 + 1/2);
dp(2) = p(2).*(exp((25*x1)/(x1 + 2)) + 1) - ...
        p(1).*exp((25*x1)/(x1 + 2)) - 2*x2;
```

In step 4, we calculate $\partial H/\partial u|_t = 2Ru(t) - p_1(t)[x_1(t) + 0.25]$ on each time subinterval (`function pH`) and compare $\|\partial H/\partial u\|^2$ with the preselected γ. If $\|\partial H/\partial u\|^2 > \gamma$, adjust u (`function AdjControl`).

```
% Partial derivative of H with respect to u
function dH = pH(x1,p1,tx,u,Tu)
% interploate the control
u = interp1(Tu,u,tx);
R = 0.1;
dH = 2*R*u - p1.*(x1 + 0.25);

% Adjust the control
function u_new = AdjControl(pH,tx,u,tu,step)
% interploate dH/du
pH = interp1(tx,pH,tu);
u_new = u - step*pH;
```

The step size τ is set as a constant in this example by some post hoc investigation. A better strategy is to select τ with a line search method which will maximize the reduction of performance measure with given $\partial H^{(i)}/\partial u$ in each iteration. Figure 4 shows the optimal state trajectory and control over the time. The value of performance measure as a function of iteration number is shown in Figure 5. In [2], two more numerical algorithms were introduced: variation of extremals and quasilinearization. These two methods basically reformulate and solve the original problem as a Boundary Value Problem (BVP). In MATLAB, a BVP is typically solved with *bvp4c*. [4] is an excellent reference on using *bvp4c*. For fix-final-time problems, u can always be solved

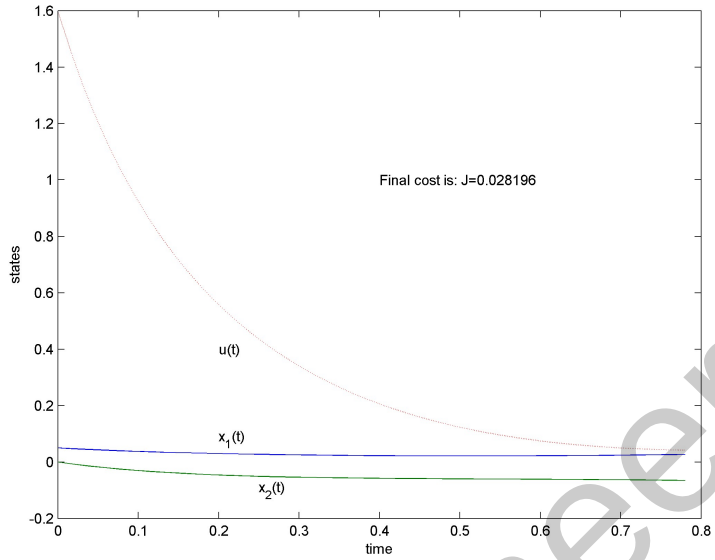

Figure 4: Example 2 Steepest descent method

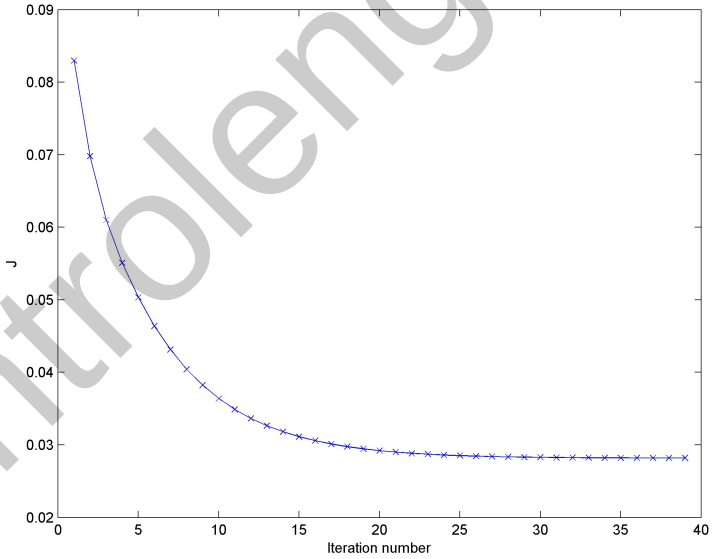

Figure 5: Performance measure reduction

with respect to $\mathbf{x}$ and $\mathbf{p}$ by applying the necessary conditions (3) and (4). And we will have $2n$ ODE's and $2n$ boundary conditions.

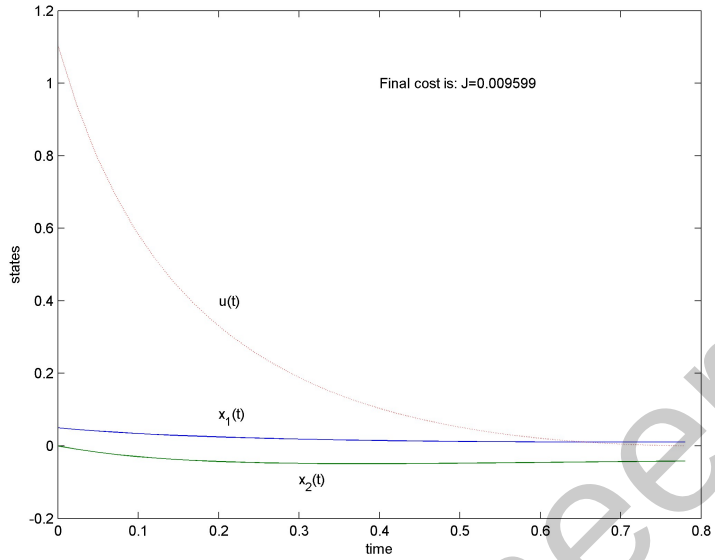

Figure 6: Solution from *bfv4c* for Problem 2

```
% Initial guess for the solution
solinit = bvpinit(linspace(0,0.78,50), ...
                  [0 0 0.5 0.5]);
options = bvpset('Stats','on','RelTol',1e-1);
global R;
R = 0.1;
sol = bvp4c(@BVP_ode, @BVP_bc, solinit, options);
t = sol.x;
y = sol.y;

% Calculate u(t) from x1,x2,p1,p2
ut = (y(3,:).*(y(1,:) + 1/4))/(2*0.1);
n = length(t);
% Calculate the cost
J = 0.78*(y(1,:)*y(1,:)' + y(2,:)*y(2,:)' + ...
        ut*ut'*0.1)/n;
```

The ODE's and boundary conditions are two major considerations in using *bvp4c*. A rule of thumb is that the number of ODE's must equal to the number of boundary conditions such that the problem is solvable. Once the optimal control problem is converted to a BVP, it is very simple to solve.

```
%-----------------------------------------------
% ODE's for states and costates
```

```
function dydt = BVP_ode(t,y)
global R;
t1 = y(1)+.25;
t2 = y(2)+.5;
t3 = exp(25*y(1)/(y(2)+2));
t4 = 50/(y(1)+2)^2;
u =  y(3)*t1/(2*R);

dydt = [-2*t1+t2*t3-t2*u
        0.5-y(2)-t2*t3
        -2*y(1)+2*y(3)-y(3)*t2*t4*t3+y(3)*u+y(4)*t2*t4*t3
        -2*y(2)-y(3)*t3+y(4)*(1+t3)];

% ------------------------------------------------
% The boundary conditions:
% x1(0) = 0.05, x2(0) = 0, tf = 0.78, p1(tf) = 0, p2(tf) = 0;
function res = BVP_bc(ya,yb)
res = [ ya(1) - 0.05
        ya(2) - 0
        yb(3) - 0
        yb(4) - 0 ];
```

In this example, *bvp4c* works perfectly. It is faster and gives better results, i.e. a smaller performance measure J comparing to the steepest descent method (see Figure 6). In the following section, we will solely use *bvp4c* when numerical solutions are needed.

3 Optimal control problems with free-final-time

Now we are prepared to deal with free-final-time problems. We will use both Symbolic Math Toolbox and *bvp4c* in the next example, which can be find from [3] on page 77, Example 2.14.

Example 3 *Given a double integral system as:*

$$\dot{x}_1(t) = x_2(t) \tag{8}$$
$$\dot{x}_2(t) = u(t) \tag{9}$$

The performance measure is:

$$J = \frac{1}{2}\int_0^{t_f} u^2(t)\mathrm{d}t$$

find the optimal control given the boundary conditions as:

$$\mathbf{x}(0) = \begin{bmatrix} 1 & 2 \end{bmatrix}^T,\ x_1(t_f) = 3,\ x_2(t_f) \text{ is free}$$

To use the Symbolic Math Toolbox, the routine is very similar to Problem 1. We first supply the ODE's and boundary conditions on states and costates to `dsolve`. The only difference is that the final time t_f itself is now a variable. As a result, the solution is a function of t_f. Next, we introduce four more variables, namely $x_1(t_f)$, $x_2(t_f)$, $p_1(t_f)$, $p_2(t_f)$ into the solution obtained above. With one additional boundary condition from

$$H\left(\mathbf{x}^*(t_f),\, \mathbf{u}^*(t_f),\, \mathbf{p}^*(t_f),\, t_f\right) + \frac{\partial h}{\partial t}\left(\mathbf{x}^*(t_f),\, t_f\right) = 0$$

For this problem, $h \equiv 0$ and we have $p_1(t_f)x_2(t_f) - 0.5p_2^2(t_f) = 0$. Now we have 5 algebraic equations with 5 unknowns. And `solve` comes in handy to solve this problem. Figure 7 shows the results from MATLAB and the analytical solution in [3]. Although Symbolic Math Toolbox works fine in this example, it should be pointed out that in most problems, it is impossible to get explicit solutions. For example, there is no explicit solutions for Example 1 even though the state equations are similar to those of Example 3.

```
sol = dsolve('Dx1 = x2, Dx2 = -p2, Dp1 = 0, Dp2 = -p1',...
           'x1(0) = 1, x2(0) = 2, x1(tf) = 3, p2(tf) = 0');
eq1 = subs(sol.x1) - 'x1tf';
eq2 = subs(sol.x2) - 'x2tf';
eq3 = subs(sol.p1) - 'p1tf';
eq4 = subs(sol.p2) - 'p2tf';
eq5 = sym('p1tf*x2tf - 0.5*p2tf^2');
%%
sol_2 = solve(eq1, eq2, eq3, eq4, eq5);
tf = sol_2.tf;
x1tf = sol_2.x1tf;
x2tf = sol_2.x2tf;

x1 = subs(sol.x1);
x2 = subs(sol.x2);
p1 = subs(sol.p1);
p2 = subs(sol.p2);
```

Because of the limitations of symbolic method, numerical methods are more useful in dealing with more general problems. However, when we try to use a numerical method such as *bvp4c*, we immediately encountered with

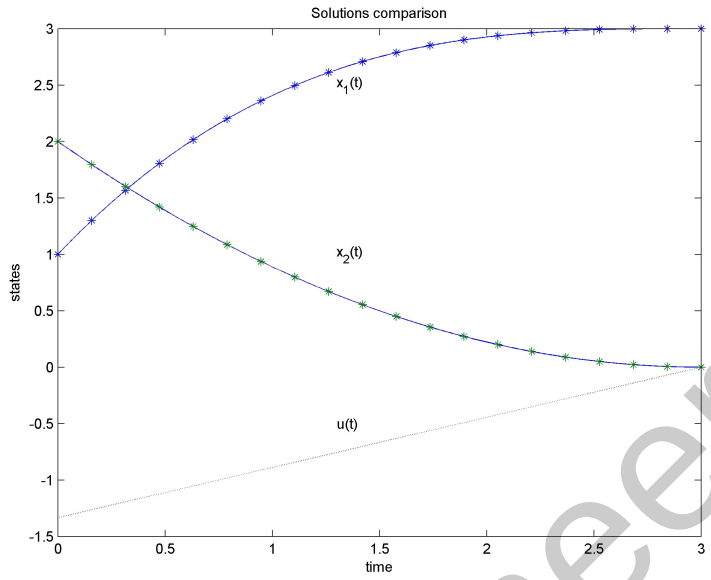

Figure 7: Example 3 Symbolic method

a problem: the time interval is not known. One common treatment [4] [7] for such a situation is to change the independent variable t to $\tau = t/T$, the augmented state and costate equations will then become $\dot{\tilde{\mathbf{x}}} = Tf(\mathbf{x}, \mathbf{q}, \tau)$.[3] Now the problem is posed on fixed interval $[0,\ 1]$. This can be implemented in *bvp4c* by treating T as an auxiliary variable. The following code snippet shows the details.

```
solinit = bvpinit(linspace(0,1),[2;3;1;1;2]);

sol = bvp4c(@ode, @bc, solinit);
y = sol.y;
time = y(5)*sol.x;
ut = -y(4,:);

% ---------------------------------------------------------------------
% ODE's of augmented states
function dydt = ode(t,y)
dydt = y(5)*[ y(2);-y(4);0;-y(3);0 ];

% ---------------------------------------------------------------------
% boundary conditions: x1(0)=1;x2(0)=2, x1(tf)=3, p2(tf)=0;
```

[3] f denotes the ODE's for state and costates.

```
%                       p1(tf)*x2(tf)-0.5*p2(2)^2
function res = bc(ya,yb)
res = [ ya(1) - 1; ya(2) - 2; yb(1) - 3; yb(4);
        yb(3)*yb(2)-0.5*yb(4)^2];
```

Alternatively, we can accomplish this by treating T as a parameter for *bvp4c* [4] The difference between the two lies in the parameter list of `bvpinit`, and function definition of the ODE's and boundary conditions. Figure 8 shows the result from numerical method which is the same as the analytical solution.

```
solinit = bvpinit(linspace(0,1),[2;3;1;1],2);

sol = bvp4c(@ode, @bc, solinit);
y = sol.y;
time = sol.parameters*sol.x;
ut = -y(4,:);

% ---------------------------------------------------------
% ODE's of augmented states
function dydt = ode(t,y,T)
dydt = T*[ y(2);-y(4);0;-y(3) ];

% ---------------------------------------------------------
% boundary conditions: x1(0)=1;x2(0)=2, x1(tf)=3, p2(tf)=0;
%                      p1(tf)*x2(tf)-0.5*p2(2)^2
function res = bc(ya,yb,T)
res = [ ya(1) - 1; ya(2) - 2; yb(1) - 3; yb(4);
        yb(3)*yb(2)-0.5*yb(4)^2];
```

Now it is the time to talk about the limitations of *bvp4c*. Thought it works well so far, we must bear in mind that the quality of the solution from *bvp4c* is heavily dependent on the initial guess. A bad initial guess may result in inaccurate solutions, or no solutions, or solutions which make no sense. For example, you may try the initial guess $p_1(0) = 0$ and compare the results with the analytical solution to see the difference. By looking at the ODE's closely, we find that $\dot{p}_1(t) = 0$. When supplied with an initial guess of 0, *bvp4c* fails to find the solution due to the state singularity.

The last problem is to further illustrate the capability, as well as the limitation, of *bvp4c* in solving optimal control problems. This example can be find from [6] which is a time-optimal problem for the double integral system with a simple control constraint.

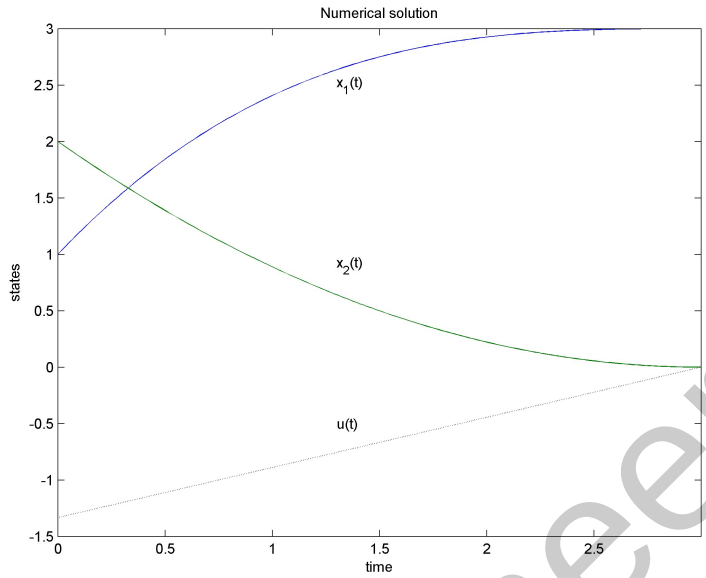

Figure 8: Example 3 Numerical method

Example 4 *Given a double integral system as:*

$$\dot{x}_1(t) = x_2(t) \tag{10}$$
$$\dot{x}_2(t) = u(t) \tag{11}$$

Minimize the final time:

$$T = \int_0^{t_f} \mathrm{d}t$$

to drive the states to the origin:

$$\mathbf{x}(0) = \begin{bmatrix} 2 & 2 \end{bmatrix}^T, \mathbf{x}(t_f) = \begin{bmatrix} 0 & 0 \end{bmatrix}^T, |u| \leq 1$$

To formulate this problem as a BVP, control domain smoothing technique must be applied first [6]. The resulted ODE's are listed below:

$$\dot{x}_1 = x_5 \left(x_2 + \frac{\mu x_3}{\sqrt{\mu x_3^2 + x_4^2}} \right) \tag{12}$$
$$\dot{x}_2 = x_5 \left(\frac{x_4}{\sqrt{\mu x_3^2 + x_4^2}} \right) \tag{13}$$
$$\dot{x}_3 = 0 \tag{14}$$
$$\dot{x}_4 = -x_5 x_3 \tag{15}$$
$$\dot{x}_5 = 0 \tag{16}$$

The boundary conditions are:

$$x_1(0) = 2,\ x_2(0) = 2,\ x_1(1) = 0,\ x_2(1) = 0,\ x_3(1)^2 + x_4(1)^2 = 1.$$

Auxiliary variable $x_5 = T$, and $u = x_4/\sqrt{\mu x_3^2 + x_4^2}$. It is well known that the optimal control for this problem is the piecewise continuous "bang-bang" control, which means that u will take either the maximum or the minimum value within the admissible control set, i.e. $+1$ or -1 in this problem. Because it is a linear second order system, there will be only one switch point and u is not continuous at the switch point. The MATLAB code is as follows

```
global mu;
mu = 0.5;
solinit = bvpinit(linspace(0,1,10),[2;2;0;-1],4);

sol = bvp4c(@ode,@bc,solinit);
...
% the solution for one value of mu is used as guess for the next.
for i=2:3
  if i==2
      mu = 0.3;
  else
      mu = .1;
  end;
  % After creating function handles, the new value of mu
  % will be used in nested functions.
  sol = bvp4c(@ode,@bc,sol);
...
end
% --------------------------------------------------------
function dydt = ode(t,y,T)
global mu;
term = sqrt(mu*y(3)^2+y(4)^2);
dydt = T*[ y(2) + mu*y(3)/term
           y(4)/term
           0
           -y(3)];

% --------------------------------------------------------------
% boundary conditions, with 4 states and 1 parameters, 5 conditions are
% needed: x1(0) =2, x2(0) = 2; x1(1) = 0; x2(1) = 0; x3(1)^2+x4(1)^2 = 1;
function res = bc(ya,yb,T)
res = [ya(1)-2
       ya(2)-2
```

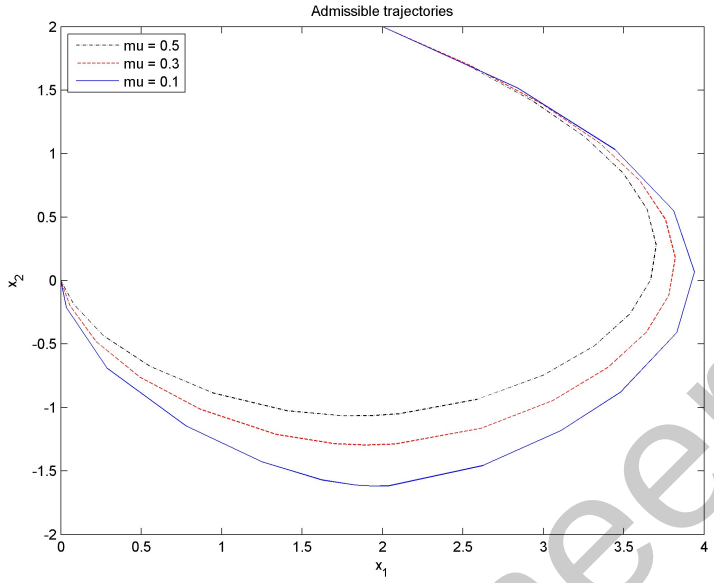

Figure 9: Example 4 State trajectory

```
yb(1)
yb(2)
yb(3)^2+yb(4)^2-1];
```

As $\mu \to 0$, u appears to be more and more stiff at the switch point, approximating the "bang-bang" control. Three values were tested: $\mu = 0.5$, $\mu = 0.3$, $\mu = 0.1$. We uses continuation to solve this problem, by which means the solution for $\mu = 0.5$ is used as guess for the BVP with $\mu = 0.3$. The trajectory and control corresponding to different μ values are shown in Figure 9 and Figure 10.

We can clearly see that u is approximating a "bang-bang" control at switch point $t = 4$ and the state trajectory resembles the optimal trajectory more and more closely as μ decrease. However, we cannot keep decreasing μ to get $\mathbf{x}$ and $\mathbf{u}$ arbitrarily close to the optimal $\mathbf{x}^*$ and $\mathbf{u}^*$. For example, if we set $\mu = 10^{-3}$ or smaller, *bvp4c* will fail due to the state singularity. In other words, when the optimal control is piecewise continuous, *bvp4c* will fail at discontinuous points.

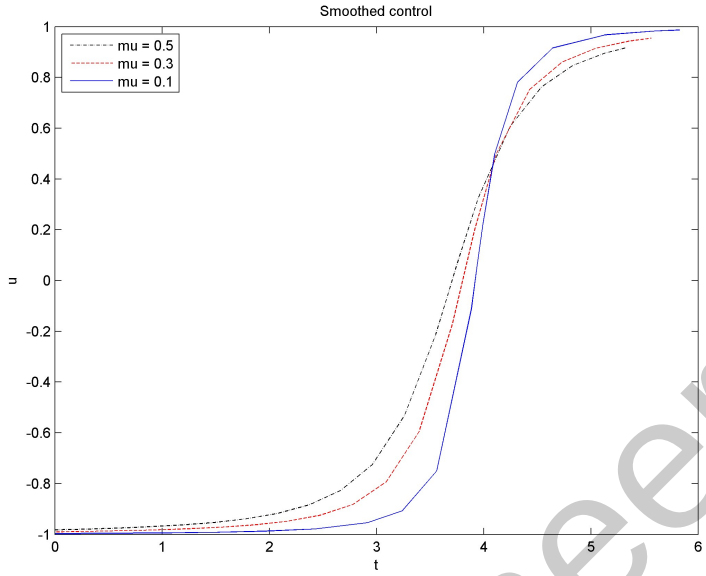

Figure 10: Example 4 Control

4 Discussion

For many optimal control problems, *bvp4c* is the best option we have. Once the problem is reformatted into a BVP, *bvp4c* can usually solve it efficiently. However, we must pay attention to the limitations of *bvp4c*, more specifically

- Good initial guess is important to get an accurate solution, if a solution exists.
- Discontinuous points (either in state/costate equations or control function) will cause *bvp4c* to fail.

For optimal control problems with no constraints on states or control, the discontinuity is not that severe a problem. By applying the minimum principle (4), we can convert the problem into a BVP and solve it with indirect methods. Nevertheless, for problems with constraints, (4) does not necessarily hold although the Hamiltonian still achieves the minimum within the admissible control set. In this case, the optimal control problem cannot be formulated as a BVP and indirect methods are not applicable. These general optimal control problems are typically handled by direct methods, most of which are closely related to nonlinear programming. Several packages for MATLAB are commercially available for such applications. Particularly, an open-source software GPOPS attracts more and more people

and improves with time. More information on GPOPS can be found in `http://www.gpops.org/`.

References

[1] M. Athans and P. Falb (2007). *Optimal control: an introduction to the theory and its applications.* Dover Publications, Inc.

[2] D. E. Kirk (2004). *Optimal control theory: an introduction.* Dover Publications, Inc.

[3] D. S. Naidu (2003). *Optimal control systems.* CRC Press LLC.

[4] L. F. Shampine., J. Kierzenka and M. W. Reichelt (2000). *Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c.*

[5] The MATHWORKS, INC., *Symbolic Math Toolbox User's Guide.*

[6] S. N. Avvakumov and Yu.N. Kiselev *Boundary value problem for ordinary differential equations with applications to optimal control.*

[7] Prof.Jonathan How *MIT Open Courses Ware: 16.323 Principles of optimal control, lecture 7.*