```
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

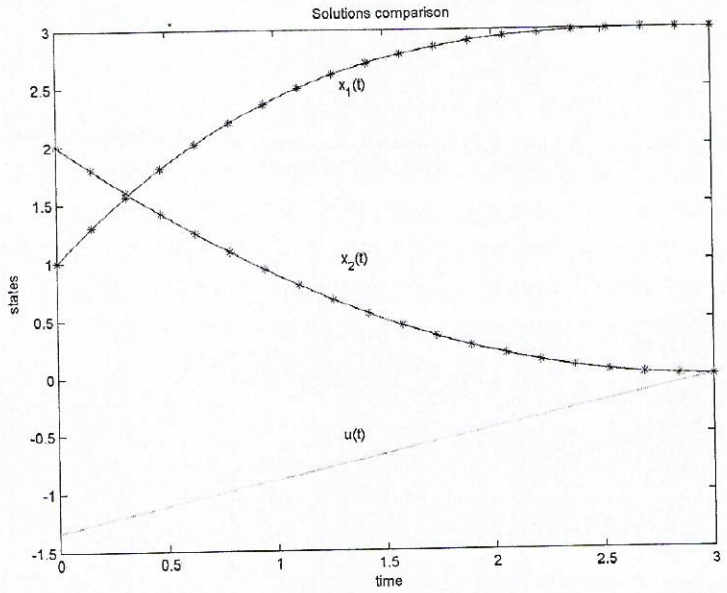

Figure 7: Example 3 Symbolic method

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.