

Lecture Notes for ECE/MAE 7360
Robust and Optimal Control (part 2)

Fall 2003

Jinsong Liang

Nov. 20, 2003

Numerical Methods for Optimization Problems

- Static optimization
 - Status: well-developed
 - Available software package: Matlab optimization toolbox, Tomlab, NEOS server, . . .
- Dynamic optimization
 - Status: not as "matured" as static optimization
 - Available software package: SOCS, RIOTS, MISER, DIRCOL, . . .

Why Optimal Control Software?

1. Analytical solution can be hard.
2. If the controls and states are discretized, then an optimal control problem is converted to a nonlinear programming problem. However . . .
 - Discretization and conversion is professional.
 - Selection and use of nonlinear programming package is an art.
 - For large scale problems, direct discretization may not be feasible.

Classification of methods for solving optimal control problems

A technique is often classified as either a *direct method* or an *indirect method*.

- An indirect method attempts to solve the optimal control necessary conditions. Thus, for an indirect method, it is necessary to explicitly derive the costate equations, the control equations, and all of the transversality conditions.
- A direct method treats an OCP as a mathematical programming problem after discretization. Direct method does not require explicit derivation and construction of the necessary conditions.

What is RIOTS?

RIOTS is a group of programs and utilities, written mostly in C, Fortran, and M-file scripts and designed as a toolbox for Matlab, that provides an interactive environment for solving a very broad class of optimal control problems.

Main contributions and features of RIOTS:

- The first implementation of consistent approximation using discretization methods based on Runge-Kutta integration
- Solves a very large class of finite-time optimal control problems
 - trajectory and endpoint constraints
 - control bounds
 - variable initial conditions and free final time problems
 - problems with integral and/or endpoint cost functions

Main contributions and features of RIOTS: (cont.)

- System functions can be supplied by the user as either C-files or M-files
- System dynamics can be integrated with fixed step-size Runge-Kutta integration, a discrete-time solver or a variable step-size method.
- The controls are represented as splines, allowing for a high degree of function approximation accuracy without requiring a large number of control parameters.

Main contributions and features of RIOTS: (cont.)

- The optimization routines use a coordinate transformation, resulting in a significant reduction in the number of iterations required to solve a problem and an increase in the solution accuracy.
- There are three main optimization routines suited for different levels of generality of the optimal control problem.
- There are programs that provide estimates of the integration error.

Main contributions and features of RIOTS: (cont.)

- The main optimization routine includes a special feature for dealing with singular optimal control problems.
- The algorithms are all founded on rigorous convergence theory.

History of RIOTS:

1. RIOTS for Sun (Adam L. Schwartz)

- Compiler: Sun C compiler
- Matlab Version: Matlab 4
- MEX version: v4

History of RIOTS: (cont.)

2. RIOTS for DOS and Windows (YangQuan Chen)

- Compiler: Watcom C
- Matlab Version: Matlab 4, 5
- MEX version: v4

History of RIOTS:(cont.)

3. RIOTS for Windows (rebuild) and Linux (Jinsong Liang)

- compiler: Microsoft Visual C++ (Windows version), GNU gcc (Linux version)
- Matlab Version: Matlab 6.5
- MEX version: v6

Summary of RIOTS usage:

$$\min_{(u, \xi) \in L_{\infty}^m[a, b] \times \mathbf{IR}^n} \left\{ f(u, \xi) \doteq g_o(\xi, x(b)) + \int_a^b l_o(t, x, u) dt \right\}$$

Subject to:

$$\dot{x} = h(t, x, u), \quad x(a) = \xi, \quad t \in [a, b]$$

$$u_{min}^j(t) \leq u^j(t) \leq u_{max}^j(t), \quad j = 1, \dots, m, \quad t \in [a, b]$$

$$\xi_{min}^j \leq \xi^j \leq \xi_{max}^j, \quad j = 1, \dots, n,$$

$$l_{ti}^{\nu}(t, x(t), u(t)) \leq 0, \quad \nu \in \mathbf{q}_{ti}, \quad t \in [a, b],$$

$$g_{ei}^{\nu}(\xi, x(b)) \leq 0, \quad \nu \in \mathbf{q}_{ei},$$

$$g_{ee}^{\nu}(\xi, x(b)) = 0, \quad \nu \in \mathbf{q}_{ee},$$

Summary of RIOTS usage: (cont.)

General procedures:

- C or m-file?
- Functions needed: (sys_)acti, (sys_)init, (sys_)h, (sys_)Dh, (sys_)l, (sys_)Dl, (sys_)g, (sys_)Dg
- Set initial condition, discretization level, spline order, integration scheme, bounds on inputs
- Call riots

Example No. 1: Rayleigh problem

$$J(u) = \int_0^{2.5} (x_1^2 + u^2) dt$$

Subject to:

$$\begin{aligned} \dot{x}_1(t) &= x_2(t), & x_1(0) &= -5 \\ \dot{x}_2(t) &= -x_1(t) + [1.4 - 0.14x_2^2(t)]x_2(t) + 4u(t), & x_2(0) &= -5 \end{aligned}$$

sys_init.m

```
function neq = sys_init(params)
% Here is a list of the different system information paramters.
% neq = 1 : number of state variables.
% neq = 2 : number of inputs.
% neq = 3 : number of parameters.
% neq = 4 : reserved.
% neq = 5 : reserved.
% neq = 6 : number of objective functions.
% neq = 7 : number of nonlinear trajectory constraints.
% neq = 8 : number of linear trajectory constraints.
% neq = 9 : number of nonlinear endpoint inequality constraints.
% neq = 10 : number of linear endpoint inequality constraints.
% neq = 11 : number of nonlinear endpoint equality constraints.
% neq = 12 : number of linear endpoint equality constraints.
% neq = 13 : 0 => nonlinear, 1 => linear, 2 => LTI, 3 => LQR, 4 => LQR and LTI.
% The default value is 0 for all except neq = 6 which defaults to 1.
neq = [1, 2; 2, 1]; % nstates = 2 ; ninputs = 1
```

sys_acti.m

```
function message = sys_acti
```

```
% This is a good time to allocate and set global variabels.
```

```
message = 'Rayleigh OCP for demo';
```

sys_h.m

```
function xdot = sys_h(neq,t,x,u)
```

```
% xdot must be a column vectore with n rows.
```

```
xdot = [x(2) ; -x(1)+(1.4-.14*x(2)^2)*x(2) + 4*u];
```

sys_l.m

```
function z = l(neq,t,x,u)
```

```
% z is a scalar.
```

```
z = x(1)^2 + u^2;
```

sys_g.m

```
function J = sys_g(neq,t,x0,xf)
```

```
% J is a scalar.
```

```
J = 0;
```

sys_Dh.m

```
function [h_x,h_u] = sys_dh(neq,t,x,u)
h_x = [0, 1; -1, 1.4-0.42*x(2)^2];
h_u = [0; 4.0];
```

sys_Dl.m

```
function [l_x,l_u,l_t] = sys_Dl(neq,t,x,u)
%l_t is not used currently
l_x = [2*x(1) 0];
l_u = 2*u;
l_t = 0;
```

sys_Dg.m

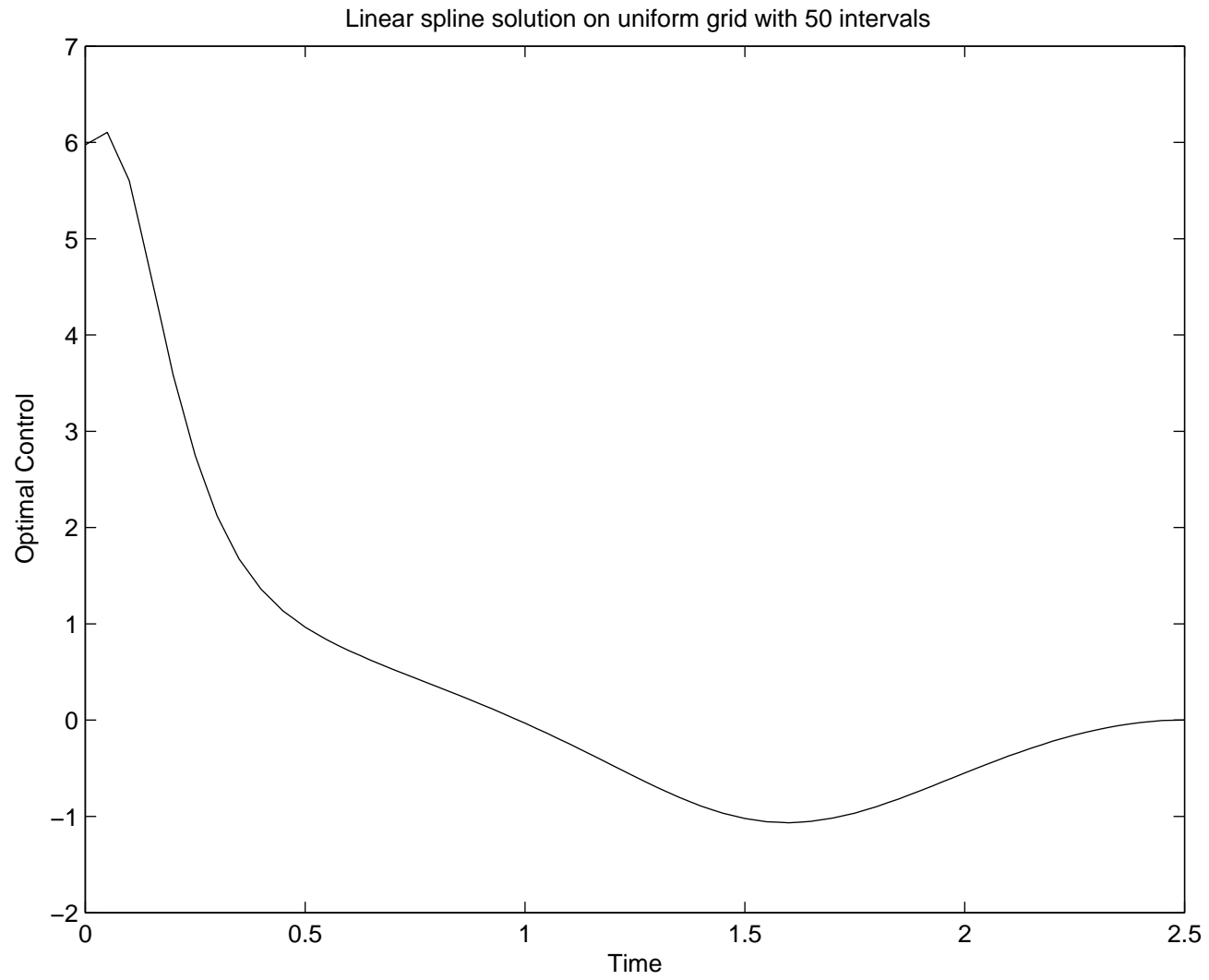
```
function [J_x0,J_xf,J_t] = sys_dg(neq,t,x0,xf)
% J_x0 and J_xf are row vectors of length n.
% J_t is not used.
J_x0 = [0 0];
J_xf = [0 0];
J_t = 0;
```

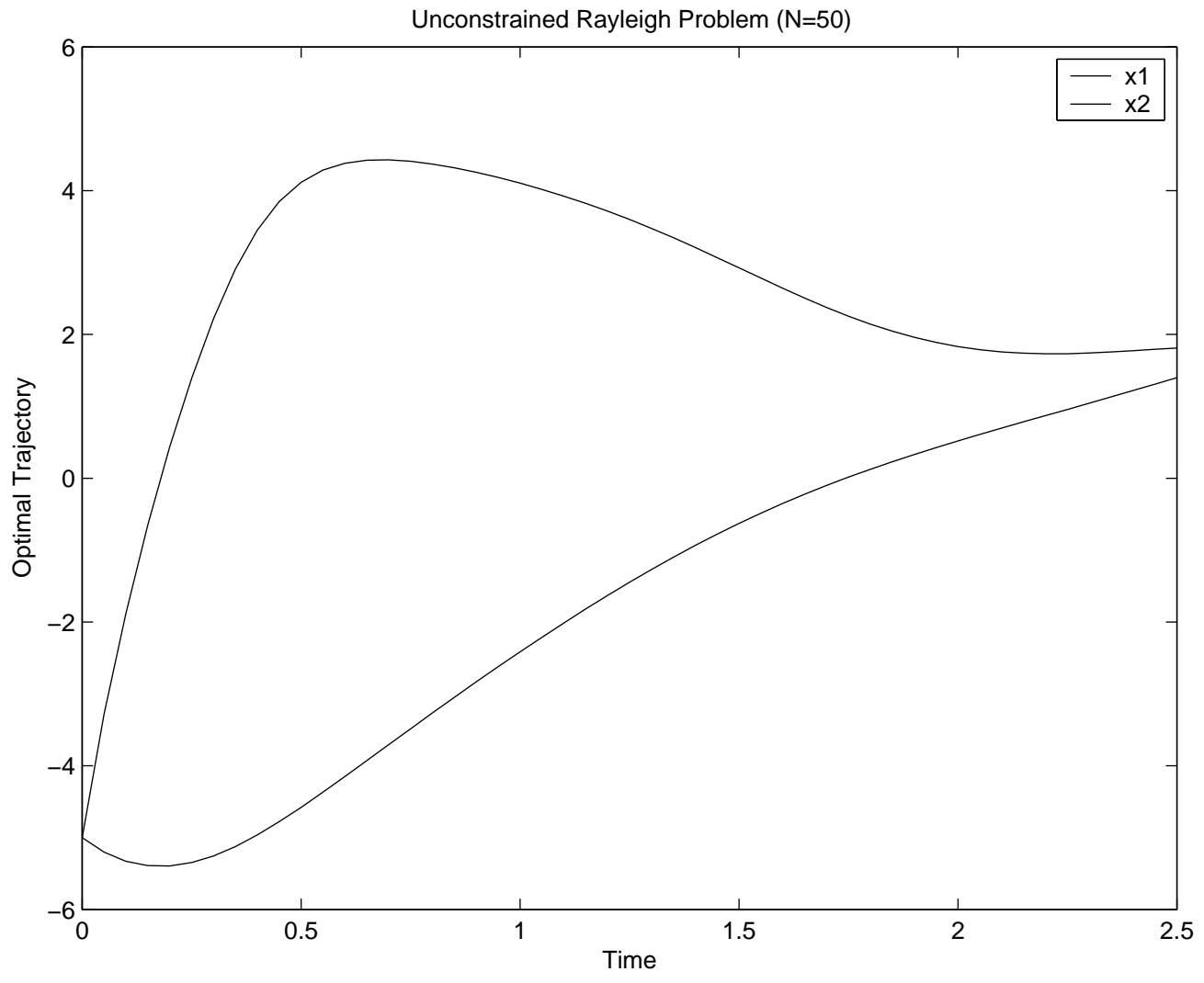
mainfun.m

```
N = 50;
x0 = [-5;-5];
t = [0:2.5/N:2.5];
u0 = zeros(1,N+2-1);          % Second order spline---initial guess

[u,x,f] = riots(x0,u0,t,[],[],[],100, 2);

sp_plot(t,u);
figure;
plot(t,x);
```





Rayleigh Problem Demo: (please be patient . . .)

Example 2: Bang problem

$$J(u) = T$$

Subject to:

$$\begin{aligned} \dot{x}_1 &= x_2, & x_1(0) &= 0, & x_1(T) &= 300, \\ \dot{x}_2 &= u, & x_2(0) &= 0, & x_2(T) &= 0 \end{aligned}$$

Transcription for free final time problem

Free final time problems can be transcribed into fixed final time problems by augmenting the system dynamics with two additional states (one additional state for autonomous problems). The idea is to specify a nominal time interval, $[a, b]$, for the problem and to use a scale factor, adjustable by the optimization procedure, to scale the system dynamics and hence, in effect, scale the duration of the time interval. This scale factor, and the scaled time, are represented by the extra states. Then RIOTS can minimize over the initial value of the extra states to adjust the scaling.

$$\min_{u, T} \quad g(T, x(T)) + \int_a^{a+T} l(t, x, u) dt$$

Subject to:

$$\dot{x} = h(t, x, u) \quad x(a) = \zeta \quad , t \in [a, a + T],$$

where $x = [x_0, x_1, \dots, x_{n-1}]^T$.

With two extra augmented states $[x_n, x_{n+1}]$, we have the new state variable $y = [x^T, x_n, x_{n+1}]^T$. Now the original problem can be converted into the equivalent fixed final time optimal control problem.

$$\min_{u, x_{n+1}} g((b-a)x_{n+1}, x(b)) + \int_a^b l(x_n, x, u) dt$$

Subject to:

$$\dot{y} = \begin{pmatrix} x_{n+1} h(x_n, x, u) \\ x_{n+1} \\ 0 \end{pmatrix}, \quad y(a) = \begin{pmatrix} x(a) \\ a \\ \xi \end{pmatrix}, \quad t \in [a, b],$$

where ξ is the initial value chosen by the user.

For autonomous systems, the extra variable x_n is not needed because it is not shown explicitly anywhere.

sys_init.m

```
function neq = sys_init(params)
% Here is a list of the different system information paramters.
% neq = 1   : number of state variables.
% neq = 2   : number of inputs.
% neq = 3   : number of parameters.
% neq = 4   : reserved.
% neq = 5   : reserved.
% neq = 6   : number of objective functions.
% neq = 7   : number of nonlinear trajectory constraints.
% neq = 8   : number of linear trajectory constraints.
% neq = 9   : number of nonlinear endpoint inequality constraints.
% neq = 10  : number of linear endpoint inequality constraints.
% neq = 11  : number of nonlinear endpoint equality constraints.
% neq = 12  : number of linear endpoint equality constraints.
% neq = 13  : 0 => nonlinear, 1 => linear, 2 => LTI, 3 => LQR, 4 => LQR and LTI.
% The default value is 0 for all except neq = 6 which defaults to 1.
neq = [1 3 ; 2 1 ; 12 2]; % nstates = 3 ; ninputs = 1; 3 endpoint constr.
```

sys_acti.m

```
function message = sys_activate
```

```
message = 'bang';
```

sys_h.m

```
function xdot = sys_h(neq,t,x,u)
```

```
global sys_params
```

```
tau = x(3);
```

```
xdot = [tau*x(2) ; tau*u(1) ; 0];
```

sys_l.m

```
function z = l(neq,t,x,u)
global sys_params
z = 0;
```

sys_g.m

```
function J = sys_g(neq,t,x0,xf)
global sys_params
F_NUM = neq(5);
if F_NUM == 1
    J = x0(3);
elseif F_NUM == 2
    J = xf(1)/300.0 - 1;
elseif F_NUM == 3
    J = xf(2);
end
```

sys_Dh.m

```
function [h_x,h_u] = sys_Dh(neq,t,x,u)
global sys_params
% h_x must be an n by n matrix.
% h_u must be an n by m matrix.

tau = x(3);
h_x = zeros(3,3);
h_u = zeros(3,1);

h_x(1,2) = tau;
h_x(1,3) = x(2);
h_x(2,3) = u(1);

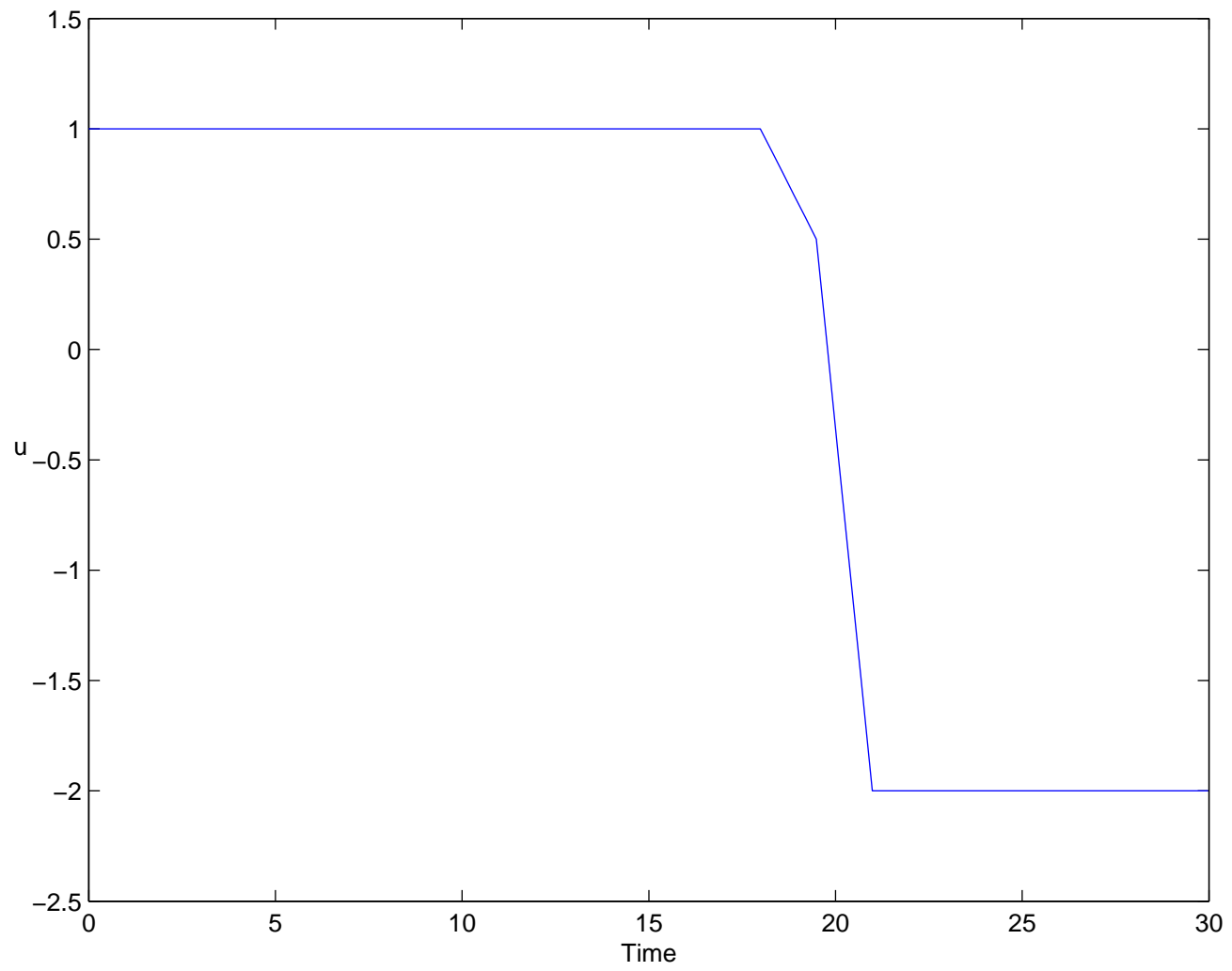
h_u(2,1) = tau;
```

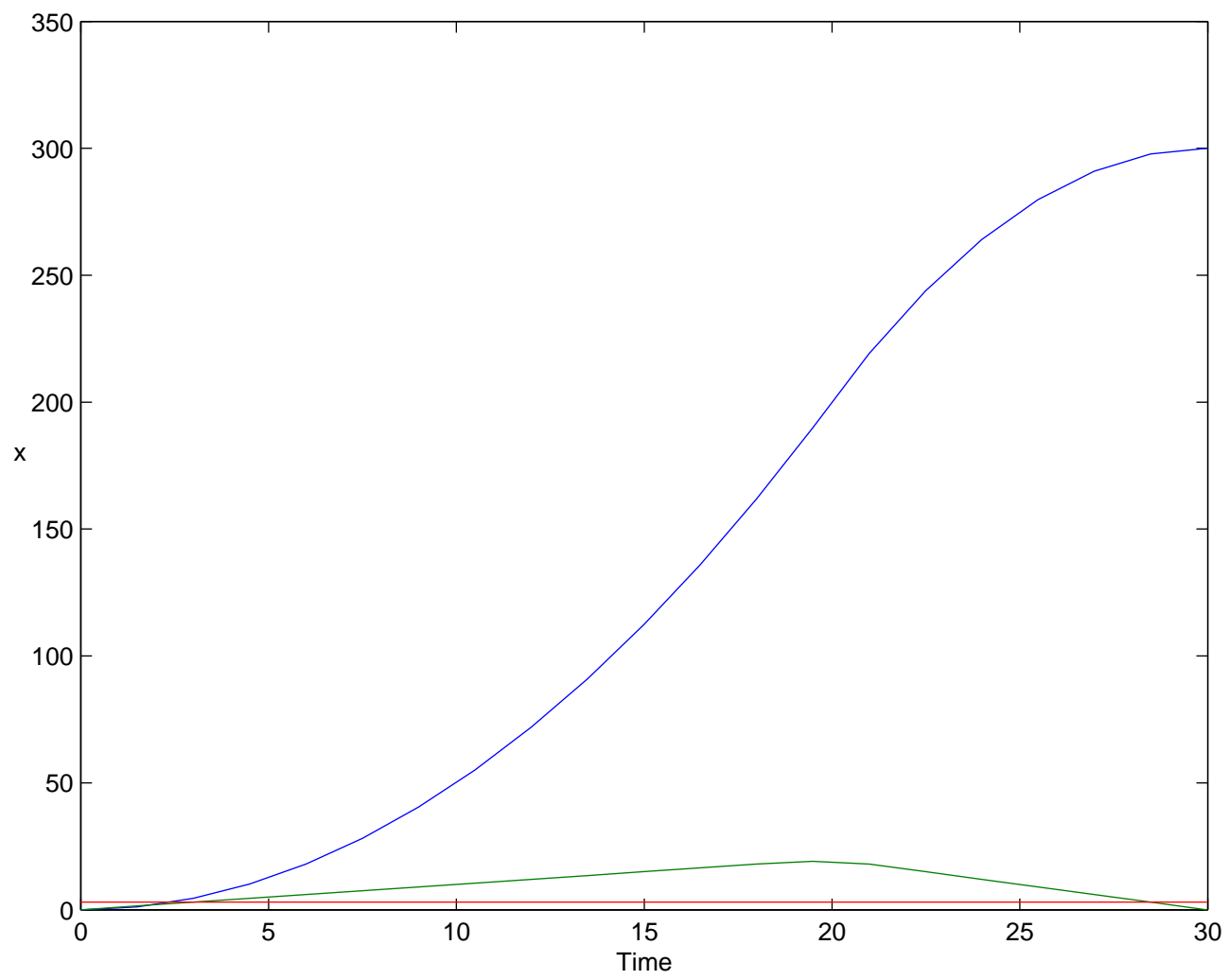
mainrun.m

```
x0 = [0 1 0 0; 0 1 0 0;1 0 .1 10]
% The first column is the initial condition with the optimal duration
% set to 1. The zero in the second column will be used to tell
% riots() that the third initial condition is a free variable. The
% third and fourth columns represent upper and lower bounds on the
% initial conditions that are free variables.

N = 20;
u0 = zeros(1,N+2-1);
t = [0:10/N:10]; % Set up the initial time vector so that
                % the intial time interval is [0,10]*1,

[u,x,f]=riots(x0,u0,t,-2,1,[],100,2);
Tf = x(3,1)
% The solution for the time interval is t*Tf = [0,10]*Tf. Thus, the
% solution for the final time is 10*Tf = 29.9813. The actual solution
% for the final time is 30.
sp_plot(t*Tf,u)
xlabel('Time'),ylabel('Optimal Control')
figure
plot(t*Tf,x)
```





Bang problem demo: (please be patient . . .)

References:

Adam L. Schwartz, *Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems*, Ph.D. Dissertation, University of California at Berkeley, 1996