Integer-Order Optimal Control Problems (IOOCP) A Tutorial and A Matlab Toolbox for Solving General IOOCP (RIOTS_95)

YangQuan Chen, *Senior Member, IEEE*

MESA LAB (Mechatronics, Embedded Systems and Automation Lab)
ME/EECS/SNRI/UCSolar, School of Engineering
5200 North Lake Road, University of California at Merced,
Merced, CA95343, USA

9:00-10:15AM, Thursday, November 14, 2013. @ KL 217

What will we study?

- **Part-1**: (Integer-Order) Optimal Control Theory:

    – The (Integer-Order) calculus of variations (a little bit)
    – Solution of general (Integer-Order) optimization problems
    – Optimal closed-loop control (LQR problem)
    – Pontryagin's minimum principle

What will we study? (cont.)

- **Part-2**: RIOTS_95 - General IOOCP Solver in the form of Matlab toolbox.

    - Introduction to (Integer-Order) numerical optimal control
    - Introduction to RIOTS_95
        * Background
        * Usage and demo

1. Part-1: Basic Optimal Control Theory

Warming-up: A Static Optimization Problem

$$\min : L(x, u), \ x \in R^n, \ u \in R^m$$

$$\text{subject to: } f(x, u) = 0, \ f \in R^n$$

Define *Hamiltonian* function

$$H(x, u, \lambda) = L(x, u) + \lambda^T f(x, u), \ \lambda \in R^n$$

Necessary conditions:

$$\frac{\partial H}{\partial \lambda} = f = 0$$

$$\frac{\partial H}{\partial x} = L_x + f_x^T \lambda = 0$$

$$\frac{\partial H}{\partial u} = L_u + f_u^T \lambda = 0$$

What is an (Integer-Order) optimal control problem (dynamic optimization)?

System model:

$$\dot{x}(t) = f(x, u, t), \ x(t) \in R^n, \ u(t) \in R^m$$

Performance index (cost function):

$$J(u, T) = \phi(x(T), T) + \int_{t_0}^{T} L(x(t), u(t), t) \mathrm{d}t$$

Final-state constraint:

$$\psi(x(T), T) = 0, \ \psi \in R^p$$

Some cost function examples

- Minimum-fuel problem

$$J = \int_{t_0}^{T} u^2(t)\mathrm{d}t$$

- Minimum-time problem

$$J = \int_{t_0}^{T} 1\mathrm{d}t$$

- Minimum-energy problem

$$J = x^T(T)Rx(T) + \int_{t_0}^{T} \left\{ x^T(t)Qx(t) + u^T(t)Ru(t) \right\}\mathrm{d}t$$
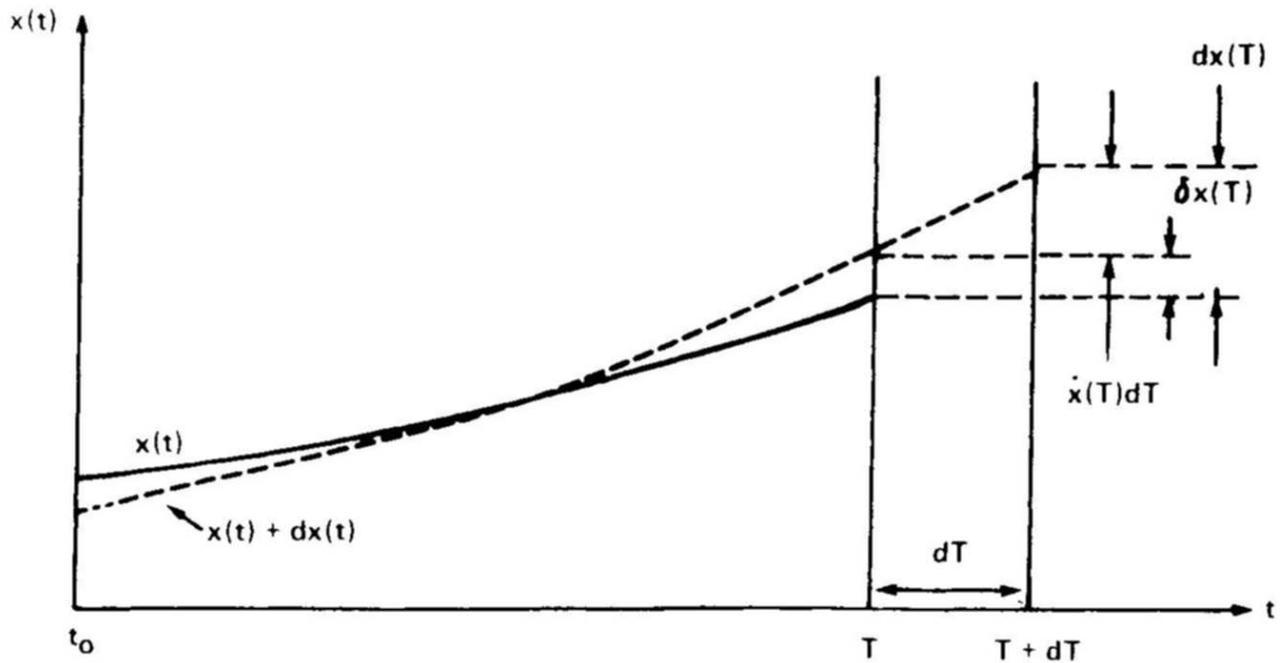
A little bit of calculus of variations

- Why calculus of variation?

  We are dealing with a function of functions $L(u(t), T)$, called *functional*, rather than a function of scalar variables $L(x, u)$. We need a new mathematical tool.

- What is $\delta x(t)$?

- Relationship between $dx(T)$, $\delta x(T)$, and dT

Relationship between $dx(T)$, $\delta x(T)$, **and** dT:

$$dx(T) = \delta x(T) + \dot{x}(T)dT$$

Leibniz's rule:

$$J(x(t)) = \int_{x_0}^{T} h(x(t), t)\mathrm{d}t$$

$$dJ = h(x(T), T)dT + \int_{t_0}^{T} [h_x^T(x(t), t)\delta x]\mathrm{d}t$$

where $h_x \triangleq \frac{\partial h}{\partial x}$.

Solution of the general optimization problem:

$$J(u, T) = \phi(x(T), T) + \int_{t_0}^{T} L(x(t), u(t), t)\mathrm{d}t \qquad (1)$$

$$\dot{x}(t) = f(x, u, t), \ x(t) \in R^n, \ u(t) \in R^m \qquad (2)$$

$$\psi(x(T), T) = 0 \qquad (3)$$

Using Lagrange multipliers $\lambda(t)$ and ν to join the constraints (2) and (3) to the performance index (1):

$$J' = \phi(x(T), T) + \nu^T \psi(x(T), T) + \int_{t_0}^{T} [L(x, u, t) + \lambda^T(t)(f(x, u, t) - \dot{x})]\mathrm{d}t$$

Note:
ν: constant, $\lambda(t)$: function

Define the *Hamiltonian function*:

$$H(x, u, t) = L(x, u, t) + \lambda^T f(x, u, t),$$

then

$$J' = \phi(x(T), T) + \nu^T \psi(x(T), T) + \int_{t_0}^{T} [H(x, u, t) - \lambda^T \dot{x}] \mathrm{d}t.$$

Using Leibniz's rule, the increment in J' as a function of increments in $x, \lambda, \nu, u,$ and t is

$$
\begin{aligned}
dJ' = \ & (\phi_x + \psi_x^T \nu)^T dx|_T + (\phi_t + \psi_t^T \nu) dt|_T + \psi^T|_T d\nu \\
& + (H - \lambda^T \dot{x}) dt|_T \\
& + \int_{t_0}^{T} [H_x^T \delta x + H_u^T \delta u - \lambda^T \delta \dot{x} + (H_\lambda - \dot{x})^T \delta \lambda] \mathrm{d}t
\end{aligned}
\tag{4}
$$

To eliminate the variation in $\dot{x}$, integrate by parts:

$$-\int_{t_0}^{T} \lambda^T \delta \dot{x} \mathrm{d}t = -\lambda^T \delta x|_T + \int_{t_0}^{T} \dot{\lambda}^T \delta x \mathrm{d}t$$

Remember

$$dx(T) = \delta x(T) + \dot{x}(T)dT,$$

so

$$
\begin{aligned}
dJ' = {} & (\phi_x + \psi_x^T \nu - \lambda)^T dx|_T + (\phi_t + \psi_t^T \nu + H)dt|_T + \psi^T|_T d\nu \\
& + \int_{t_0}^{T} [(H_x + \dot{\lambda})^T \delta x + H_u^T \delta_u + (H_\lambda - \dot{x})^T \delta\lambda]\mathrm{d}t.
\end{aligned}
$$

According to the Lagrange theory, the constrained minimum of J is attained at the unconstrained minimum of J'. This is achieved when $dJ' = 0$ for all independent increments in its arguments. Setting to zero the coefficients of the independent increments $d\nu$, δx, δu, and $\delta\lambda$ yields following necessary conditions for a minimum.

System model:

$$\dot{x} = f(x, u, t), \ t \geq t_0, \ t_0 \text{ fixed}$$

Cost function:

$$J(u, T) = \phi(x(T), T) + \int_{t_0}^{T} L(x, u, t)\mathrm{d}t$$

Final-state constraint:

$$\psi(x(T), T) = 0$$

State equation:

$$\dot{x} = \frac{\partial H}{\partial \lambda} = f$$

Costate (adjoint) equation:

$$-\dot{\lambda} = \frac{\partial H}{\partial x} = \frac{\partial f^T}{\partial x}\lambda + \frac{\partial L}{\partial x}$$

Stationarity condition:

$$0 = \frac{\partial H}{\partial u} = \frac{\partial L}{\partial u} + \frac{\partial f^T}{\partial u}\lambda$$

Boundary (transversality) condition:

$$(\phi_x + \psi_x^T \nu - \lambda)^T|_T dx(T) + (\phi_t + \psi_t^T \nu + H)|_T dT = 0$$

Note that in the boundary condition, since $dx(T)$ and dT are not independent, we cannot simply set the coefficients of $dx(T)$ and dT equal to zero. If $dx(T) = 0$ (fixed final state) or $dT = 0$ (fixed final time), the boundary condition is simplified. What if neither is equal to zero?

An optimal control problem example: temperature control in a room
It is desired to heat a room using the least possible energy. If $\theta(t)$ is the temperature in the room, θ_a the ambient air temperature outside (a constant), and $u(t)$ the rate of heat supply to the room, then the dynamics are

$$\dot{\theta} = -a(\theta - \theta_a) + bu$$

for some constants a and b, which depend on the room insulation and so on. By defining the state as

$$x(t) \triangleq \theta(t) - \theta_a,$$

we can write the state equation

$$\dot{x} = -ax + bu.$$

In order to control the temperature on the fixed time interval $[0, T]$ with the least supplied energy, define the cost function as

$$J(u) = \frac{1}{2}s(x(T)) + \frac{1}{2}\int_0^T u^2(t)dt,$$

for some weighting s.
The Hamiltonian is

$$H = \frac{u^2}{2} + \lambda(-ax + bu)$$

The optimal control $u(t)$ is determined by solving:

$$\dot{x} = H_\lambda = -ax + bu, \tag{5}$$

$$\dot{\lambda} = -H_x = a\lambda, \tag{6}$$

$$0 = H_u = u + b\lambda \tag{7}$$

From the stationarity condition (7), the optimal control is given by

$$u(t) = -b\lambda(t), \tag{8}$$

so to determine $u^*(t)$ we need to only find the optimal costate $\lambda^*(t)$.
Substitute (8) into (5) yields the state-costate equations

$$\dot{x} = -ax - b^2\lambda \tag{9}$$
$$\dot{\lambda} = a\lambda \tag{10}$$

Sounds trivial? Think about the boundary condition!

From the boundary condition:

$$(\phi_x + \psi_x^T \nu - \lambda)^T|_T dx(T) + (\phi_t + \psi_t^T \nu + H)|_T dT = 0,$$

dT is zero, $dx(T)$ is free, and there is no final-state constraint. So

$$\lambda(T) = \frac{\partial \phi}{\partial x}\Big|_T = s(x(T) - 10).$$

So the boundary condition of x and λ are specified at t_0 and T, respectively. This is called two-point boundary-value (TPBV) problem.

Let's assume $\lambda(T)$ is known. From (10), we have

$$\lambda(t) = e^{-a(T-t)} \lambda(T).$$

So

$$\dot{x} = -ax - b^2\lambda(T)e^{-a(T-t)}.$$

Solving the above ODE, we have

$$x(t) = x(0)e^{-at} - \frac{b^2}{a}\lambda(T)e^{-aT}sinh(at).$$

Now we have the second equation about $x(T)$ and $\lambda(T)$

$$x(T) = x(0)e^{-aT} - \frac{b^2}{2a}\lambda(T)(1 - e^{-2aT})$$

Assuming $x(0) = 0°$, $\lambda(T)$ can now be solved:

$$\lambda(T) = \frac{-20as}{2a + b^2s(1 - e^{-2aT})}$$

Now the costate equation becomes

$$\lambda^*(t) = \frac{-10ase^{at}}{ae^{aT} + sb^2 sinh(aT)}$$

Finally we obtain the optimal control

$$u^*(t) = \frac{10abse^{at}}{ae^{aT} + sb^2 sinh(aT)}$$

Conclusion:
TPBV makes it hard to solve even for simple OCP problems. In most cases, we have to rely on numerical methods and dedicated OCP software package, such as RIOTS.

Closed-loop optimal control: LQR problem
Problems with the optimal controller obtained so far:

- solutions are hard to compute.

- open-loop

For Linear Quadratic Regulation (LQR) problems, a closed-loop controller exists.

System model:

$$\dot{x} = A(t)x + B(t)u$$

Objective function:

$$J(u) = \frac{1}{2}x^T(T)S(T)x(T) + \frac{1}{2}\int_{t_0}^{T}(x^TQ(t)x + u^TR(t)u)\mathrm{d}t$$

where $S(T)$ and $Q(t)$ are symmetric and positive semidefinite weighting matrices, $R(t)$ is symmetric and positive definite, for all $t \in [t_0, T]$. We are assuming T is fixed and the final state $x(T)$ is free.

State and costate equations:

$$
\begin{aligned}
\dot{x} &= Ax - BR^{-1}B^T\lambda, \\
-\dot{\lambda} &= Qx + A^T\lambda
\end{aligned}
$$

Control input:

$$u(t) = -R^{-1}B^T\lambda$$

Terminal condition:

$$\lambda(T) = S(T)x(T)$$

Considering the terminal condition, let's assume that $x(t)$ and $\lambda(t)$ satisfy a linear relation for all $t \in [t_0, T]$ for some unknown matrix $S(t)$:

$$\lambda(t) = S(t)x(t)$$

To find $S(t)$, differentiate the costate to get

$$\dot{\lambda} = \dot{S}x + S\dot{x} = \dot{S}x + S(Ax - BR^{-1}B^T Sx).$$

Taking into account the costate equation, we have

$$-\dot{S}x = (A^T S + SA - SBR^{-1}B^T S + Q)x.$$

Since the above equation holds for all $x(t)$, we have the *Riccati equation*:

$$-\dot{S} = A^T S + SA - SBR^{-1}B^T S + Q, \ t \leq T.$$

Now the optimal controller is given by

$$u(t) = -R^{-1}B^T S(t)x(t),$$

and $K(t) = R^{-1}B^T S(t)x(t)$ is called *Kalman gain*.
Note that solution of $S(t)$ does not require $x(t)$, so $K(t)$ can be computed off-line and stored.

Pontryagin's Minimum Principle: a bang-bang control case study

So far, the solution to an optimal control problem depends on the stationarity condition $\frac{\partial H}{\partial u} = 0$. What if the control $u(t)$ is constrained to lie in an admissible region, which is usually defined by a requirement that its magnitude be less than a given value?

Pontryagin's Minimum Principle: **the Hamiltonian must be minimized over all admissible u for optimal values of the state and costate**

$$H(x^*, u^*, \lambda^*, t) \leq H(x^*, u, \lambda^*, t), \text{ for all admissible } u$$

Example: bang-bang control of systems obeying Newton's laws:
System model:

$$\dot{x}_1 = x_2,$$
$$\dot{x}_2 = u,$$

Objective function (time-optimal:

$$J(u) = T = \int_0^T 1 \mathrm{d}t$$

Input constraints:

$$|u(t)| \leq 1$$

End-point constraint:

$$\psi(x(T), T) = \begin{pmatrix} x_1(T) \\ x_2(T) \end{pmatrix} = 0.$$

The Hamiltonian is:

$$H = 1 + \lambda_1 x_2 + \lambda_2 u,$$

where $\lambda = [\lambda_1, \lambda_2]^T$ is the costate.
Costate equation:

$$\begin{aligned} \dot{\lambda}_1 &= 0 & \Rightarrow \lambda_1 = \text{constant} \\ \dot{\lambda}_2 &= -\lambda_1 & \Rightarrow \lambda_2(t) \text{ is a linear function of } t \text{ (remember it!)} \end{aligned}$$

Boundary condition:

$$\lambda_2(T)u(T) = -1$$

Pontryagin's minimum principle requires that

$$\lambda_2^*(t)u^*(t) \leq \lambda_2^*(t)u(t)$$

How to make sure $\lambda_2^*(t)u^*(t)$ is less or equal than $\lambda_2^*(t)u(t)$ for any admissible $u(t)$?
Answer:

$$u^*(t) = -sgn(\lambda_2^*(t)) = \begin{cases} 1, & \lambda_2^*(t) < 0 \\ -1, & \lambda_2^*(t) > 0 \end{cases}$$

What if $\lambda_2^*(t) = 0$? Then u^* is undetermined.
Since $\lambda_2^*(t)$ is linear, it changes sign at most once. So does $u^*(t)$!

Since rigorous derivation of $u^*(t)$ is still a little bit complicated, an intuitive method using phase plane will be shown below.

Going backward ($x_1 = 0$ and $x_2 = 0$) in time from T, with $u(t) = +1$ or $u(t) = -1$, we obtain a trajectories, or *switching curve*, because switching of control (if any) must occur on this curve. Why?
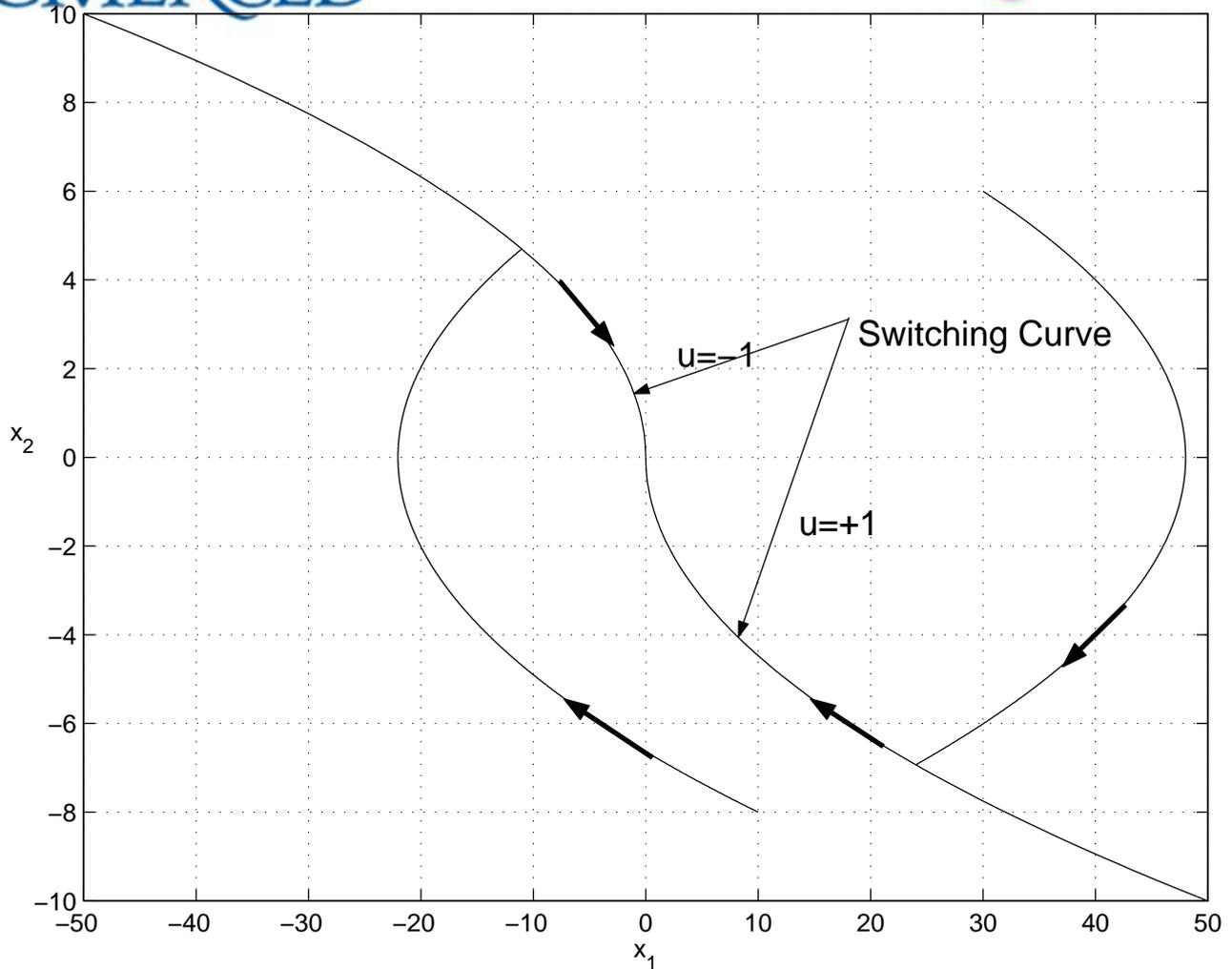
References:
F. L. Lewis and V. L. Syrmos, *Optimal Control*, John Wiley & Sons, Inc, 1997
A. E. Bryson and Y. C. Ho, *Applied Optimal Control*, New York: Hemisphere, 1975
J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, 2001
I. M. Gelfand and S. V. Fomin, *Calculus of Variations*, New York: Dover Publications, 1991

2. Part-2: RIOTS_95 - General OCP Solver in the form of Matlab toolbox

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 an 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 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 fro different levels of generality of the optimal control problem.

- There are programs that provides 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:

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

Subject to:

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

$$u^j_{min}(t) \le u^j(t) \le u^j_{max}(t), \quad j = 1, \ldots, m, \quad t \le [a, b]$$

$$\xi^j_{min} \le \xi^j \le \xi^j_{max}, \quad j = 1, \ldots, n,$$

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

$$g^\nu_{ei}(\xi, x(b)) \le 0, \quad \nu \in \mathbf{q}_{ei},$$

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

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 a
% 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;
```
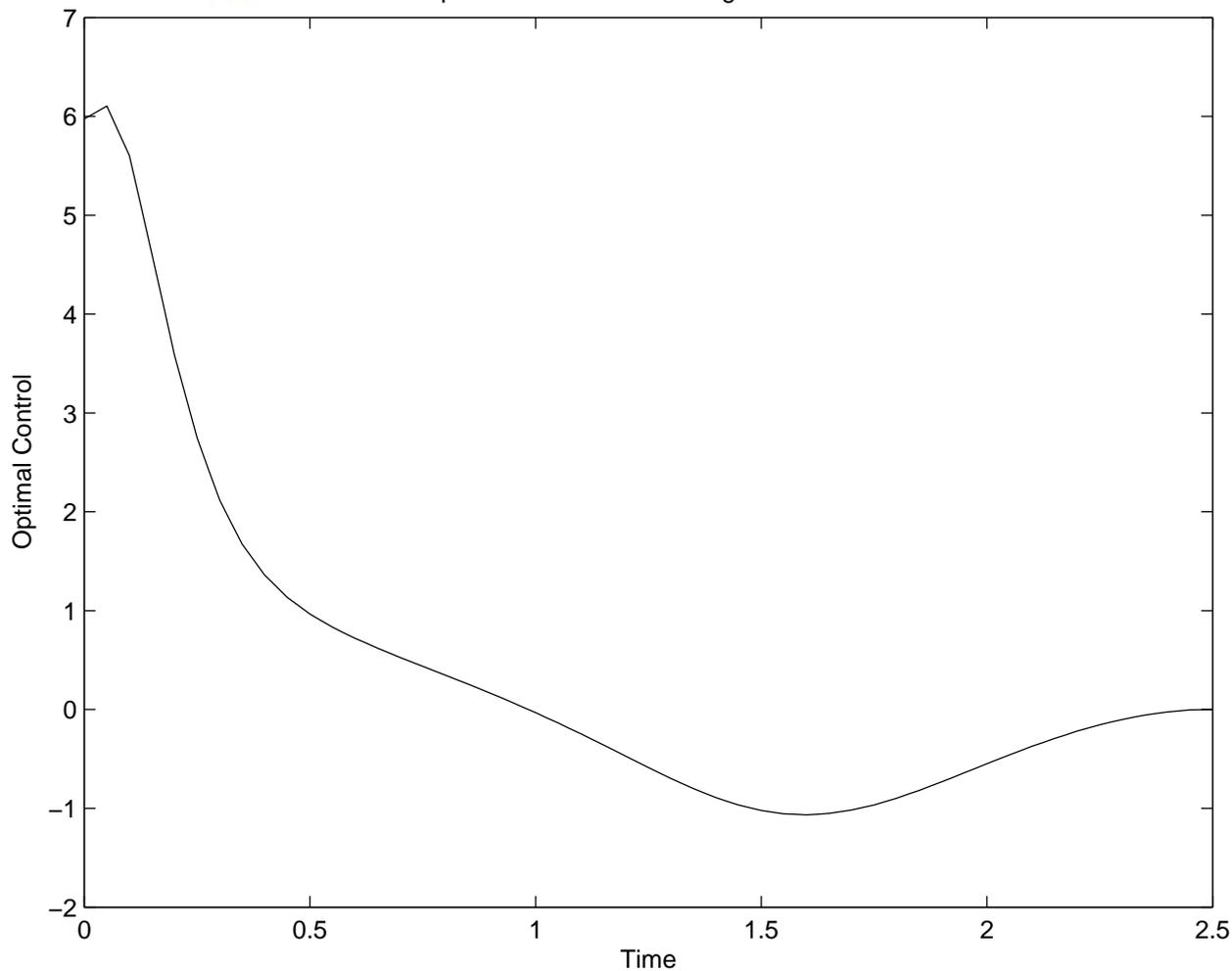
sys_Dg.m

```
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;
```
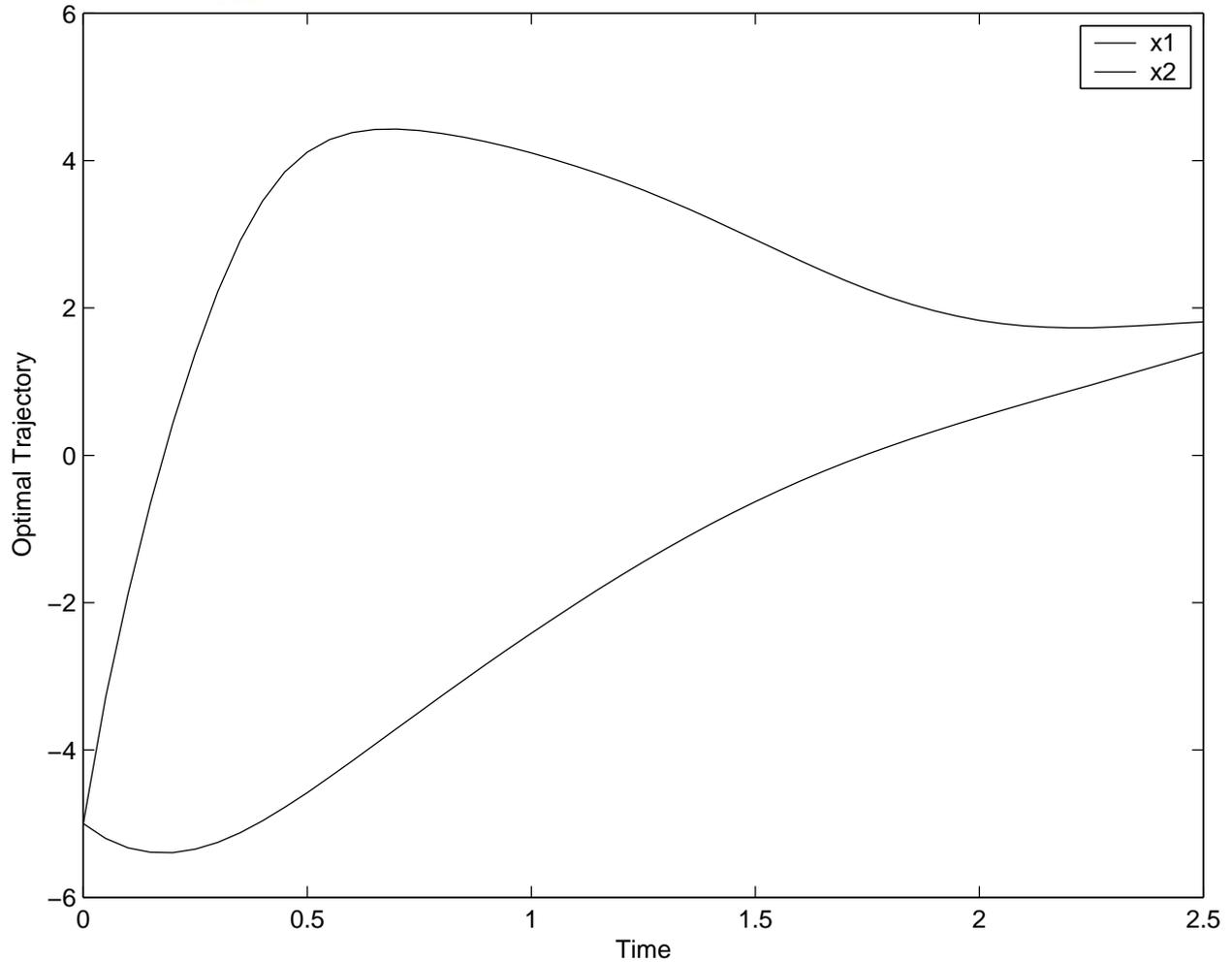
mainfun.m

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

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

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

Linear spline solution on uniform grid with 50 intervals

Unconstrained Rayleigh Problem (N=50)

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

Example 2: Bang problem

$$J(u) = T$$

Subject to:

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

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

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.

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

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 a
% 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 const
```

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 duraction
% 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 t
                                         % the intial time interval is [0,10]
[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)
```
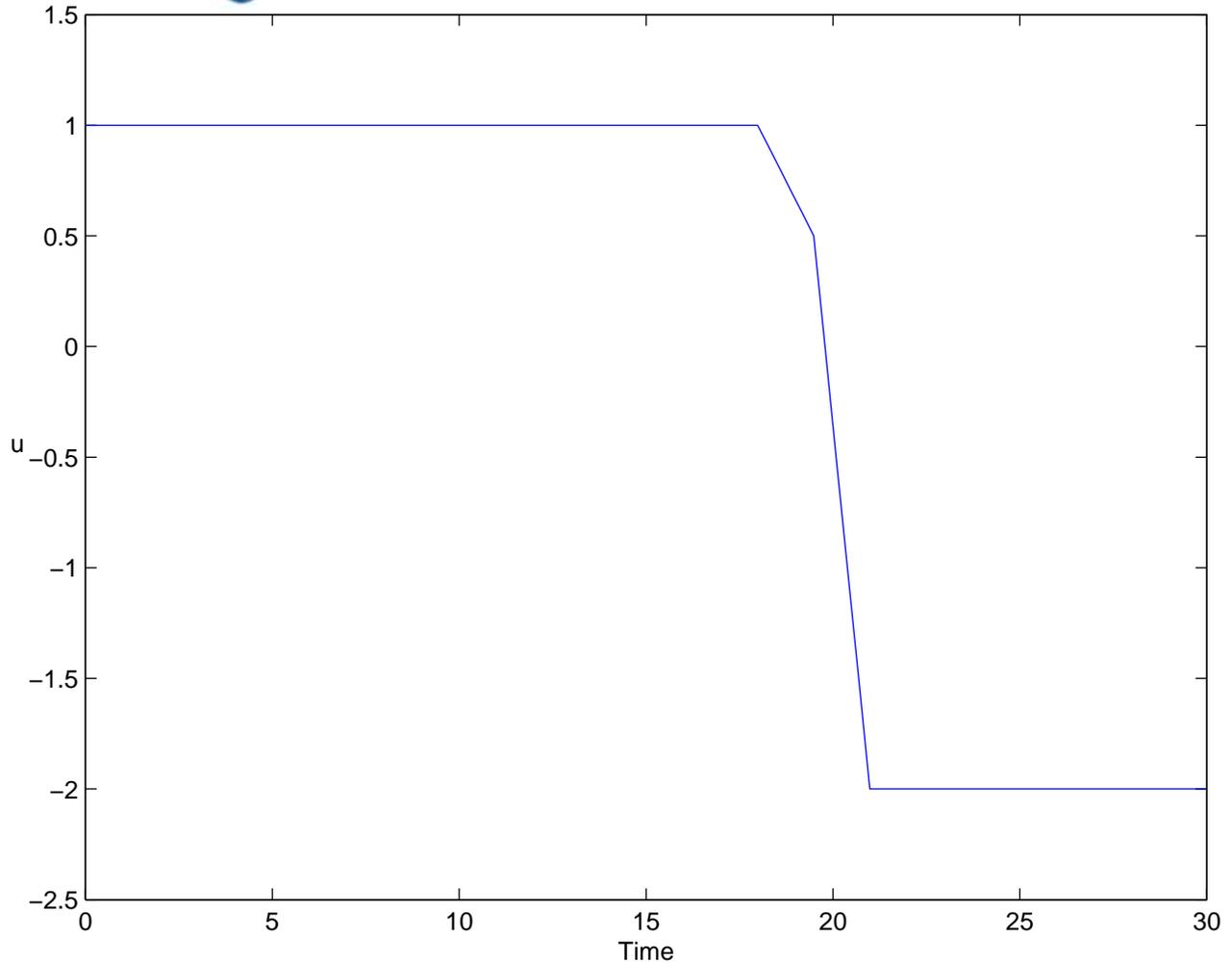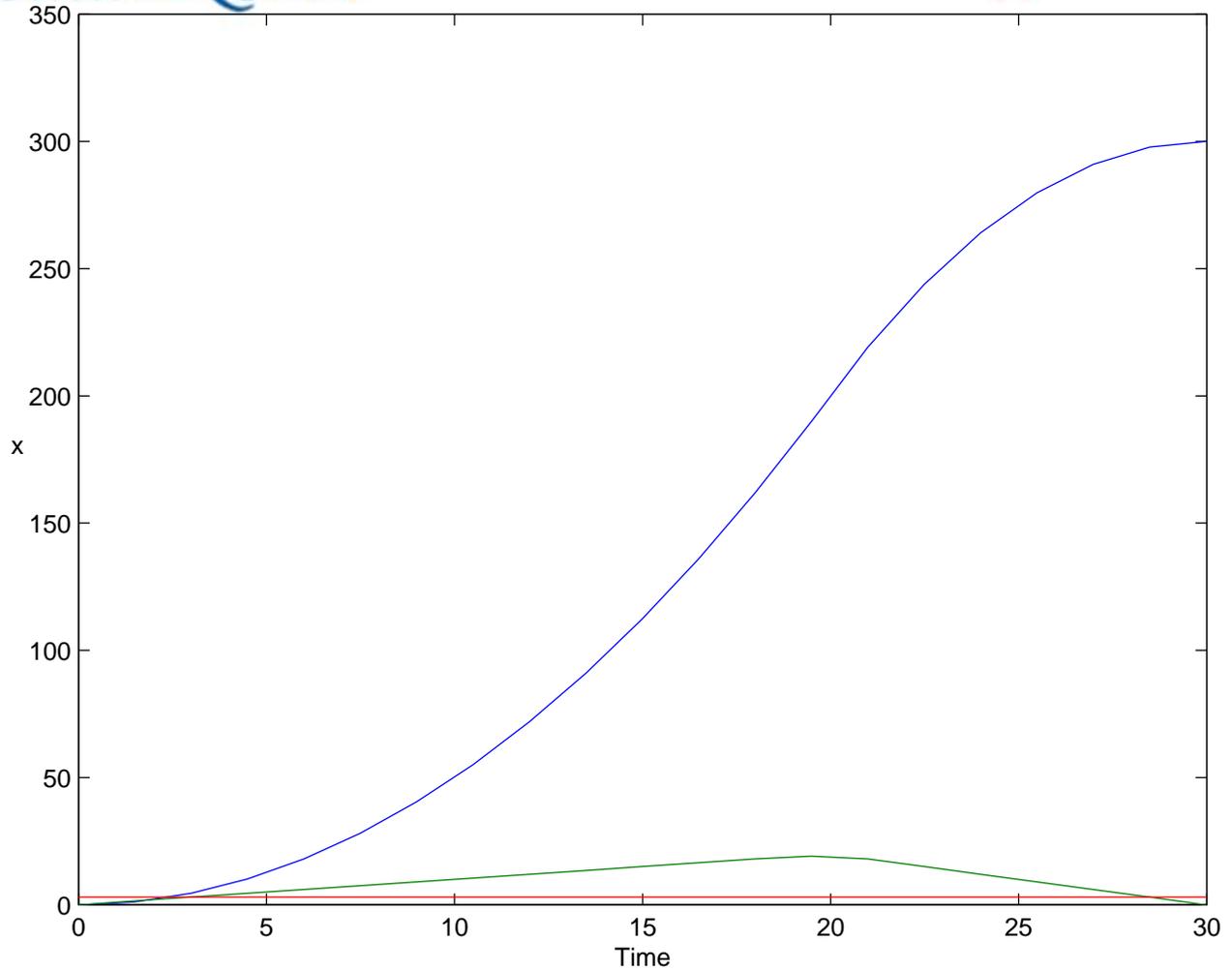
Bang problem demo: (please be patient ...)

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

YangQuan Chen and Adam L. Schwartz."*RIOTS_95 – a MATLAB Toolbox for Solving General Optimal Control Problems and Its Applications to Chemical Processes,*" Chapter in "Recent Developments in Optimization and Optimal Control in Chemical Engineering", Rein Luus Editor, Transworld Research Publishers, 2002. `http://www.csois.usu.edu/publications/pdf/pub077.pdf` [See also `http://www.optimization-online.org/DB_HTML/2002/11/567.html`]

Thank you for your attention!

Question/Answer Session.

Mobile Actuator and Sensor Networks (MAS-net):
`http://mechatronics.ece.usu.edu/mas-net/`

Task-Oriented Mobile Actuator and Sensor Networks (TOMAS-net)
IEEE IROS'05 Tutorial, Edmonton, Canada, August 2, 2005:
`http://www.csois.usu.edu/people/yqchen/tomasnet/`

RIOTS_95 web:
`http://www.csois.usu.edu/ilc/riots`

Credits:

Dr. Adam L. Schwartz, the father of RIOTS for creating the first Unix OS4 version of RIOTS. Dr. YangQuan Chen for making the Windows version of RIOTS_95. Dr. Jinsong Liang for making Windows-based RIOTS MEX-6 compatible and versions for Linux and Solaris, AIX.

RIOTS_95 users world-wide for feedback.

This set of slides were mostly prepared by Jinsong Liang under the supervision of Prof. YangQuan Chen as a module in Dr. Chen's ECE/MAE7360 "Robust and Optimal Control" course syllabus. This OCP/RIOTS module has been offered twice (2003, 2004).