1 实验目的

本次实验主要是对网络化系统的仿真方法进行探究，基于matlab中的TrueTime工具箱进行仿真，对网络控制系统的稳定性和性能进行评估。实验目的为：

- 了解网络化控制系统的仿真方法
- 学习使用TrueTime仿真工具，进行仿真分析

2 仿真平台搭建

本实验使用的控制系统仿真工具为Matlab/Simulink，利用TrueTime工具箱进行仿真，Truetime是由瑞典Lund 工学院Henriksson等人开发的一个基于Matlab/Simulink的实时网络控制系统的仿真工具箱，为NCS理论的仿真研究提供了简易可行、功能齐全的手段，拜托了软件编程实现特定的网络通讯协议、通信延迟所带来的困难，支持控制与实时调度同时仿真可以方便地仿真实时系统中的资源调度问题。TrueTime仿真软件主要包括两个基本模块：内核模块（TrueTime Kernel）和网络模块（TrueTime Network）。

内核模块可以是时间驱动也可以是事件驱动的，它主要包含了一个实时内核，A/D，D/A转换端口，与网络模块连接的信号端口（信号接收（Rcv），信号发送（Snd）），实时调度（schedule）显示端口等，调度器与监视器的输出用于显示仿真过程中公共资源（CPU、监控器、网络）的分配，此外，它还有一个外部中断通道（Interrupts）可以处理外部中断。任务和中断处理器的执行需要通过用户自定义函数来实现。调度策略使用一个优先权函数来决定任务的属性。

网络模块是事件驱动的，当有消息进入或离开网络时它便执行。一条消息包含的信息有发送和接收节点号，用户数据（如测量信号和控制信号），消息的长度和其他可选的实时属性（如优先级或最终时限等）。网络模块包含两个信号端口（信号接收（Rcv），信号发送（Snd）），以及一个实时调度（schedule）显示端口。其中收发信号端口可以通过Matlab模块扩充至多个接口，TrueTime中预定义了多种调度策略，包括固定优先级（Fixed Priority），单调速率（RM，Rate Monotonic），截止期单调（DM，Dead line Monotonic），最小截止期优先（EDF，Earliest Dead line First），同时，它还有多种介质访问控制协议（CSMA/CD,CSMA/CA,Round Robin,FDMA或TDMA）和相应的参数可以选择。

利用TrueTime仿真软件，网络控制系统中的各个处理单元（包括传感器、控制器和执行器）都可以由计算机模块构建，而网络控制系统的实时网络可以由所需协议的网络模块来构建，另外，再结合Matlab/Simulink 的其他控制模块，就可以简便而又快速的构建一个实时的网络控制系统。利用TrueTime仿真软件包的优点在于：

1. 由于该仿真软件中两个基本模块具有通用性，在构建各个处理单元时只需选用其相应的接口功能进行连接即可，因此大大加快模型构建的速度。
2. 该仿真软件可以比较方便模拟各种实时调度策略，并通过Scope可以很方便地观察各个任务的调度情况和对象的输出情况。
3. 在网络模块中，可以很方便的模拟数据传输率、数据包的大小和丢包率等网络参数，有利于分析各类参数对网络控制系统的性能影响。

使用TrueTime进行仿真时，首先要对网络控制系统中的内核模块TrueTime Kernel和网络模块TrueTime Network以及各个节点进行初始化，在初始化中需要完成以下工作：

1. 初始化功能块内核，设置功能块输入、输出端口的数目和调度策略。

2. 定义消息函数，并根据节点采用的驱动方式，设置不同的消息调度策略。对于时钟驱动的节点，调用ttCreatPeriodicTask函数，设置周期性的任务调度策略，以实现定时采样功能。对于事件驱动的节点，调用ttCreateInterruptHandler函数，设置中断式消息调度策略，使节点在接受到网络数据后触发相应的消息。

3. 初始化网络端口，设置节点对应的网络端口代号。控制网络功能由TrueTime Network功能块实现。网络类型、节点数、传输速率以及丢包率等参数可以通过TrueTime Network功能块的设置窗口进行设置。具体的参数选项根据网络类型的不同而不同。

3 实验步骤

本次实验控制对象为直流电机，其传递函数为：

$$G(s) = \frac{1000}{s^2 + s} \tag{1}$$

该电机通过网络IEEE 802.11b/g（WLAN）的方式进行远程控制，系统的控制结构如图1所示：

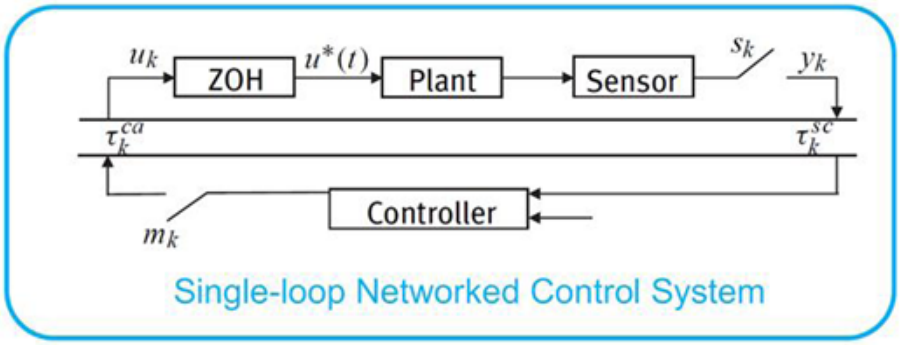

Figure 1: 系统的控制结构图

假设传感器采用时钟驱动的方式进行周期性采样，控制器和执行器采用事件驱动方式。存在网络丢包现象，其状态空间模型可以描述成:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} + (1-p) \begin{bmatrix} 0 \\ 1000 \end{bmatrix} u(t) + p \begin{bmatrix} 0 \\ 1000 \end{bmatrix} u(t-1) \tag{2}$$

$$y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(t) \tag{3}$$

其中初始状态为$x(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$，$p$为丢包概率，$p = 0$ 表示无数据丢包，$p = 1$ 表示有数据丢包，假设不会发生连续丢包。给定控制器为$u(t) = Kx(t) = [-0.005 \quad -0.005]$。

对于本仿真实验，按照TrueTime的仿真流程，可以把它分为三步，即仿真系统模块图、模块初始化、编写任务代码，接下来我们按照实验流程介绍具体的步骤。

3.1 仿真系统模块图

基于系统的流程图，通过阅读network和wirelss两个示例文件，我们在wireless的基础上搭建了系统模块图如图 2：

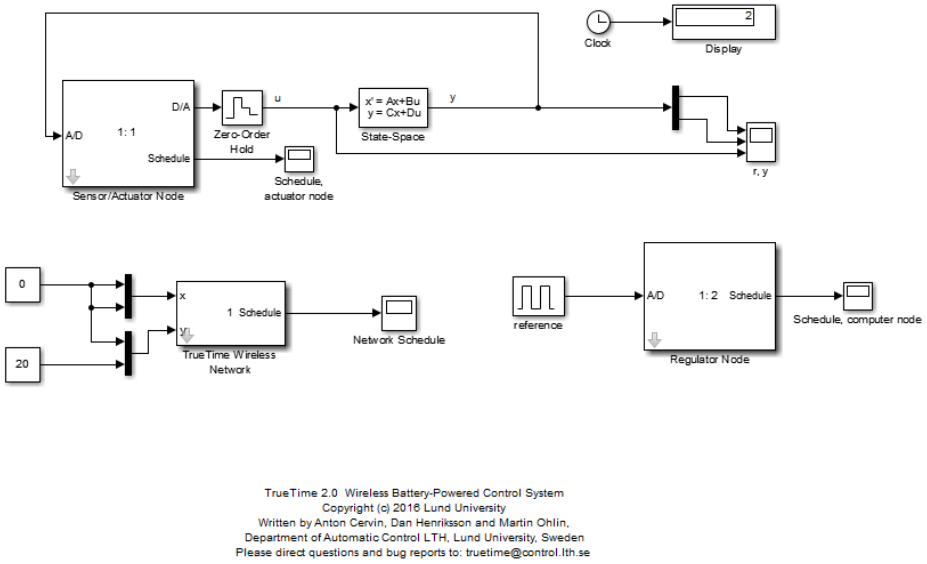

Figure 2: 系统模块图

从图中我们可以得到，系统的反馈回路为执行器节点输出$u(t)$经过零阶保持器传入控制对象，即本题中的状态空间方程，输出$y(t)$，进入传感器节点，在Sensor/Actuator Node的TrueTime Kernel中，模拟了传感器节点到控制器再到执行器节点的过程。在其中，TrueTime Wireless Network模块模拟了网络的部分，在其中我们可以设置丢包率的问题。仿真流程的过程可以描述成图 3。

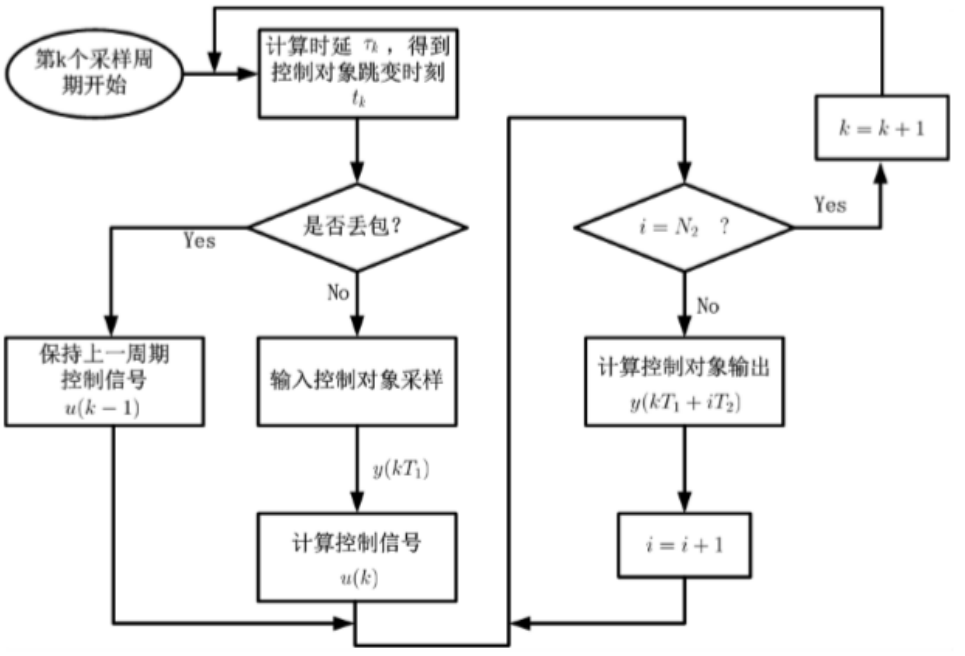

Figure 3: NCS离散仿真流程图

3.2 模块初始化

在搭建好仿真系统模块图后，我们就要对其中的执行器控制器编写相应的初始代码。对于actuator_init和regulator_init初始化，直接可以参照系统给出的wirelss示例不需要改动，代码段如下：

```
function actuator_init

% Distributed control system: actuator node
%
% Receives messages from the controller and actuates
% the plant.

% Initialize TrueTime kernel
ttInitKernel( prioFP ); % fixed priority
ttSetKernelParameter( energyconsumption , 0.010); % 10 mW

% Create mailboxes
ttCreateMailbox( control_signal , 10)
ttCreateMailbox( power_ping , 10)
ttCreateMailbox( power_response , 10)

% Create sensor task
data.y = 0;
offset = 0.0;
period = 0.010;
prio = 1;
ttCreatePeriodicTask( sens_task , offset, period,  senscode , data);
ttSetPriority(prio,  sens_task );

% Create actuator task
deadline = 100;
prio = 2;
ttCreateTask( act_task , deadline,  actcode );
ttSetPriority(prio,  act_task );

% Create power controller task
offset = 2.07;
period = 0.025;
prio = 3;
power_data.transmitPower = 20;
power_data.name = 1;      % We are node number 1 in the network
power_data.receiver = 2; % We are communicating with node 2
power_data.haverun = 0;  % We have not run yet
ttCreatePeriodicTask( power_controller_task , offset, period, ...
```

```
      powctrlcode , power_data);
ttSetPriority(prio,  power_controller_task );

% Create power response task
deadline = 100;
prio = 4;
ttCreateTask( power_response_task , deadline,  powrespcode );
ttSetPriority(prio,  power_response_task );

% Initialize network
ttCreateHandler( nw_handler , 1,  msgRcvActuator );
ttAttachNetworkHandler( nw_handler );
```

```
function regulator_init

% Distributed control system: regulator node
%
% Receives messages from the sensor node, computes control signal
% and sends it back to the actuator node.

% Initialize TrueTime kernel
ttInitKernel( prioFP ); % fixed priority
ttSetKernelParameter( energyconsumption , 0.010); % 10 mW

% Create mailboxes
ttCreateMailbox( sensor_signal , 10)
ttCreateMailbox( power_ping , 10)
ttCreateMailbox( power_response , 10)

% Controller parameters
h = 0.010;
N = 100000;
Td = 0.035;
K = 1.5;

% Create task data (local memory)
data.u = 0.0;
data.K = K;
data.ad = Td/(N*h+Td);
data.bd = N*K*Td/(N*h+Td);
data.Dold = 0.0;
data.yold = 0.0;

```

```
% Create controller task
deadline = h;
prio = 1;
ttCreateTask( pid_task , deadline,  ctrlcode , data);
ttSetPriority(prio,  pid_task );

% Create power controller task
offset = 2;
period = 0.025;
prio = 2;
power_data.transmitPower = 20;
power_data.name = 2;      % We are node number 2 in the network
power_data.receiver = 1; % We are communicating with node 1
power_data.haverun = 0;  % We have not run yet
ttCreatePeriodicTask( power_controller_task , offset, period, ...
    powctrlcode , power_data);
ttSetPriority(prio,  power_controller_task );

% Create power response task
deadline = 100;
prio = 3;
ttCreateTask( power_response_task , deadline,  powrespcode );
ttSetPriority(prio,  power_response_task );

% Initialize network
ttCreateHandler( nw_handler , 1,  msgRcvCtrl );
ttAttachNetworkHandler( nw_handler );
```

3.3 编写任务代码

最后一节中，将控制器任务(task)代码的编写，任务(task)用来模拟用户代码的执行任务，分为两类：

1. 任务的周期性时钟触发：ttCreatePeriodicTask(name,starttime, period, codeFcn, data)
2. 任务的非周期性时钟、事件、中断触发：ttCreateTask(name, deadline, codeFcn, data)，ttCreateJob (taskname,time)，ttKillJob(taskname)

参照示例wirelss中的代码，我们可以对任务代码进行编写。本实验中的任务较为简单，示例中给出了较为复杂的pid控制的实现方法，而本实验中的控制器直接可以表示为输入量的倍数关系，故代码更为简化且不需要再读入参考值。seg 中一共有两个case，case 1 产生控制信号$u(t)$，case 2 将控制信号发送出去。实现代码如下：

```
function [exectime, data] = ctrlcode(seg, data)

```

```
switch seg,
 case 1,
  % Read all buffered packets
  temp = ttTryFetch( sensor_signal );
  while ¬isempty(temp),
    y = temp;
    temp = ttTryFetch( sensor_signal );
  end

  data.u = [-0.005 -0.005] * y ;
  exectime = 0.0005;
 case 2,
  msg.msg = data.u;
  msg.type =  control_signal ;
  ttSendMsg(1, msg, 80);     % Send 80 bits to node 1 (actuator)
  exectime = -1; % finished
end
```

4 实验结果分析

4.1 实验曲线图

对于仿真系统，我们假设周期$h = 0.01s$，并且采样周期τ_k也为$0.01s$，分别给出丢包率分别为$px = 1 = 0.3, px = 1 = 0.6$时的系统状态曲线、输出曲线和控制曲线如图 4 5

4.2 不同丢包率的稳定性分析

4.2.1 基于LMI的稳定性分析

对于具有Bernoulli过程的线性跳变系统，我们可以建立期系统的模型如下：

$$x_{k+1} = \left[\ e^{Ah} - \int_0^h e^{As} B\bar{K} ds \ \right] x_k \qquad p = 0 \tag{4}$$

$$x_{k+1} = e^{Ah} x_k - \left[\ \int_0^h e^{As} B\bar{K} ds \ \right] x_{k-1} \qquad p = 1 \tag{5}$$

写成拓展方程：

$$\xi_{k+1} = \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} \tag{6}$$

则有：

$$\xi_{k+1} = \begin{bmatrix} e^{Ah} - \int_0^h e^{As} B\bar{K} ds & 0 \\ -\bar{K} & 0 \end{bmatrix} \xi_k \qquad p = 0 \tag{7}$$

$$\xi_{k+1} = \begin{bmatrix} e^{Ah} & \int_0^h e^{As} B ds \\ 0 & I \end{bmatrix} \xi_k \qquad p = 1 \tag{8}$$

从上式可得：

$$A_0 = \begin{bmatrix} e^{Ah} - \int_0^h e^{As} B\bar{K} ds & 0 \\ -\bar{K} & 0 \end{bmatrix} \tag{9}$$

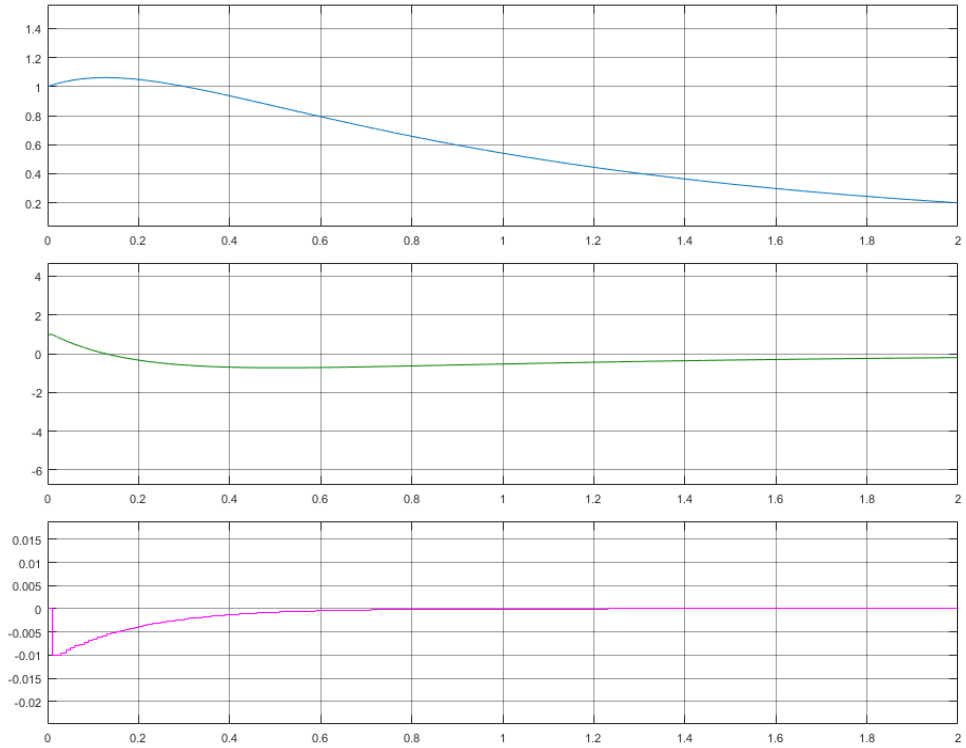

Figure 4: p(x=1)=0.3时的曲线图

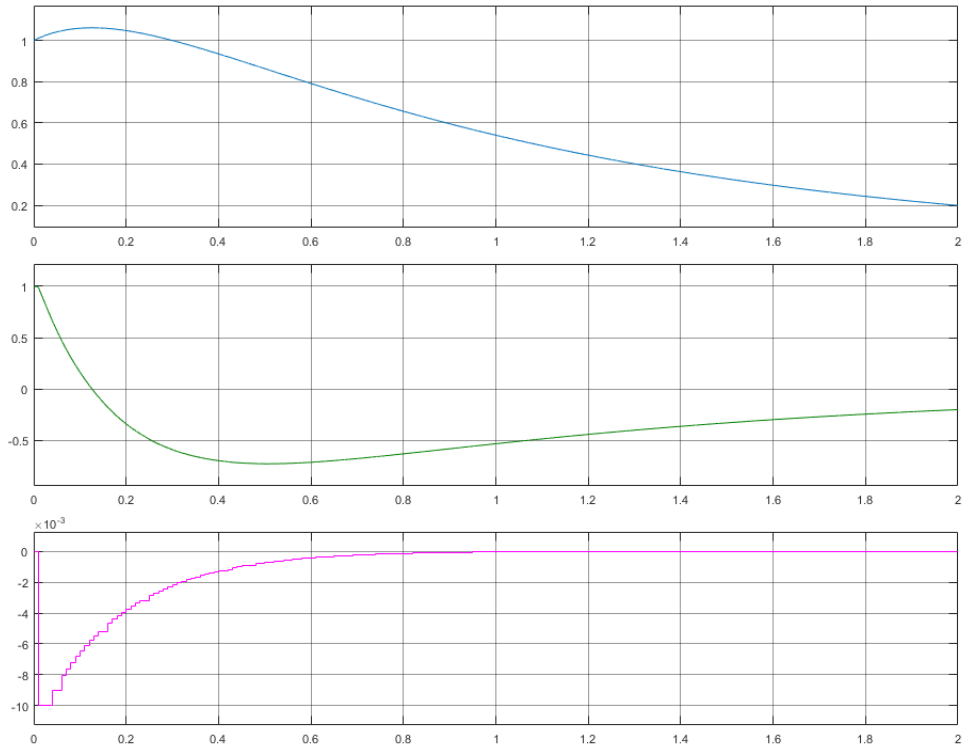

Figure 5: p(x=1)=0.6时的曲线图

$$A_1 = \begin{bmatrix} e^{Ah} & \int_0^h e^{As}Bds \\ 0 & I \end{bmatrix} \tag{10}$$

对于这样的线性跳变系统，如果存在$P > 0$，满足：

$$(1-p)A_0^T P A_0 + pA_1^T P A_1 - P < 0 \tag{11}$$

则线性跳变系统是MSS的。

此时我们可以使用LMI工具包对P进行求解，代码如下，得到丢包率大于0.95时，系统不是均方稳定。

```
h=0.01;
eAh=[1 1-exp(-h);0 exp(-h)]
b = [h h+exp(-h)-1;0 1-exp(-h)]
A01=eAh-[h h+exp(-h)-1;0 1-exp(-h)]*[0;1000]*[0.005 0.005];
A0=[A01(1),A01(3),0;A01(2),A01(4),0;-0.005,-0.005,0]
tmp=[h h+exp(-h)-1;0 1-exp(-h)]*[0;1000];
A1=[eAh(1),eAh(3),tmp(1);eAh(2),eAh(4),tmp(2);0,0,1]
c=zeros(1,120);
i=1;
for p=0:0.01:1
    setlmis([]);
    P=lmivar(1,[3,1]);
    lmiterm([-1,1,1,P],1,1);%P>0
    lmiterm([2,1,1,P],(1-p)*A0 ,A0);
    lmiterm([2,1,1,P],p*A1 ,A1);
    lmiterm([2,1,1,P],-1,1);
    lmis=getlmis;
    [tmin,xfeas]=feasp(lmis);
    if tmin>0
        c(i)=p;
        i=i+1;
    end
end
c
```

4.2.2 基于TrueTime仿真的稳定性分析

同样，如果使用LMI工具包进行分析，我们也可以通过手工调节模型丢包率来确定系统的稳定性，在调节丢包率到0.95左右时，系统出现了不稳定情况如图 **??**。

故若要线性跳变系统稳定，则丢包率要小于0.95。

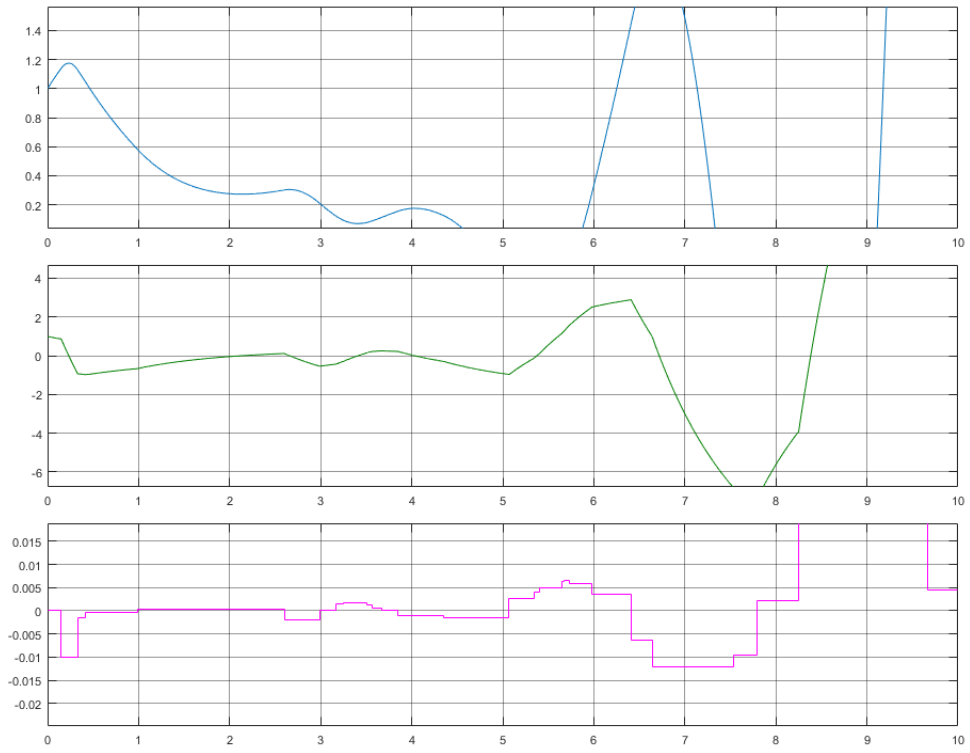

Figure 6: p(x=1)=0.95时的曲线图