# 计算机仿真与Matlab大作业

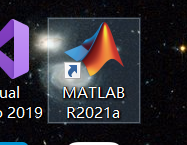
目录

[计算机仿真与Matlab大作业 1](#_Toc85661216)

[数学模型整理及Simulink仿真 1](#_Toc85661217)

[Matlab Function模块形式替换 3](#_Toc85661218)

[S函数模块形式替换 5](#_Toc85661219)

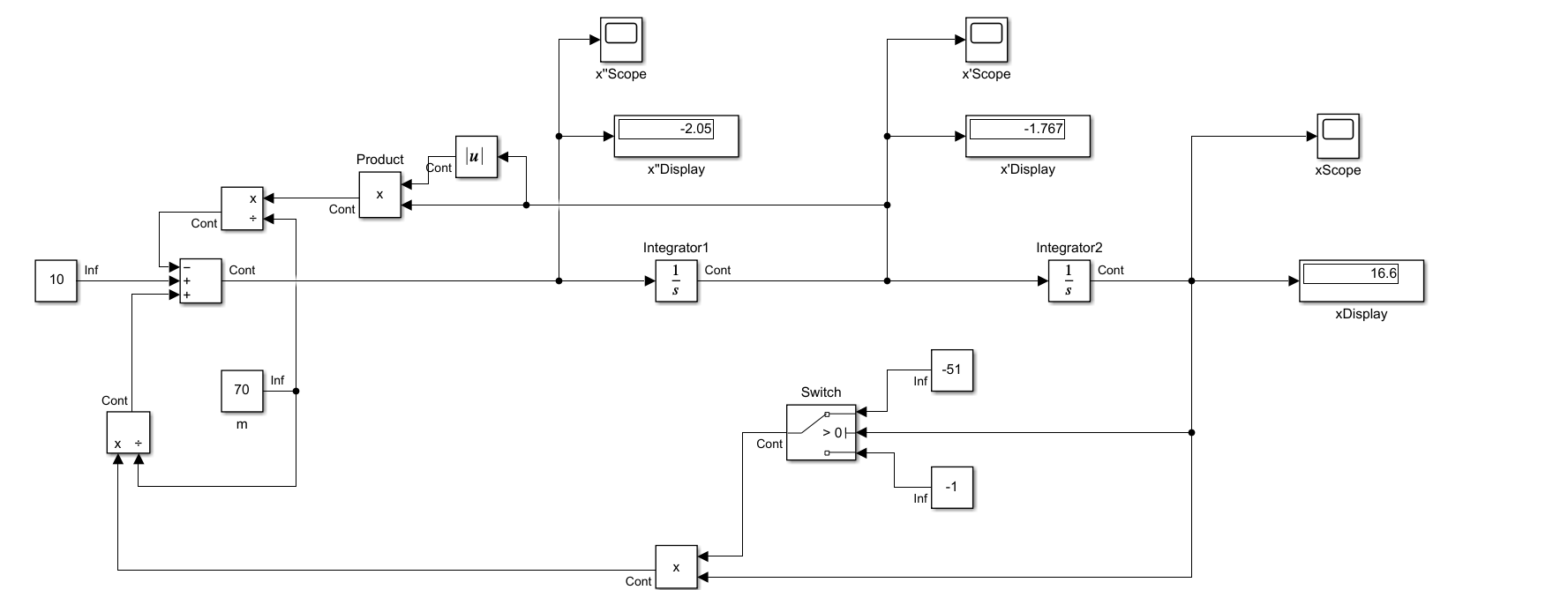


**由于本人安装的Matlab版本为2021a，fcn模块已经被移除，于是本人将6个模块通过matlab function模块的形式替换，将4个模块通过S函数模块的形式替换。**

## 数学模型整理及Simulink仿真

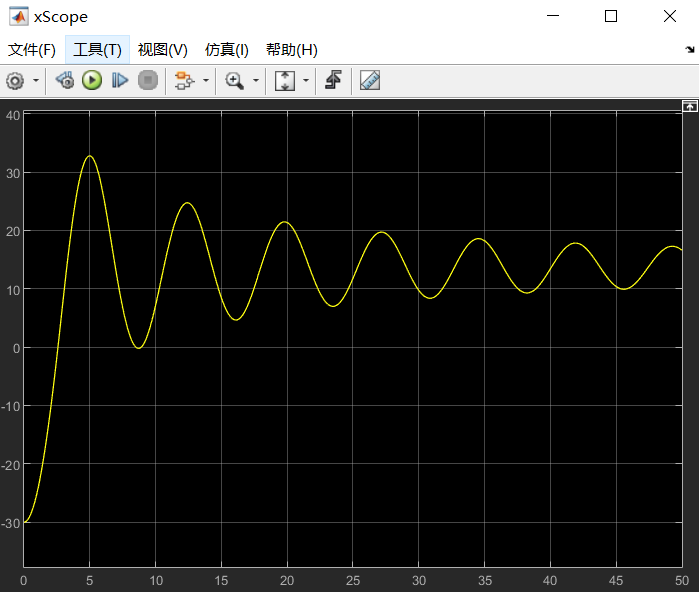
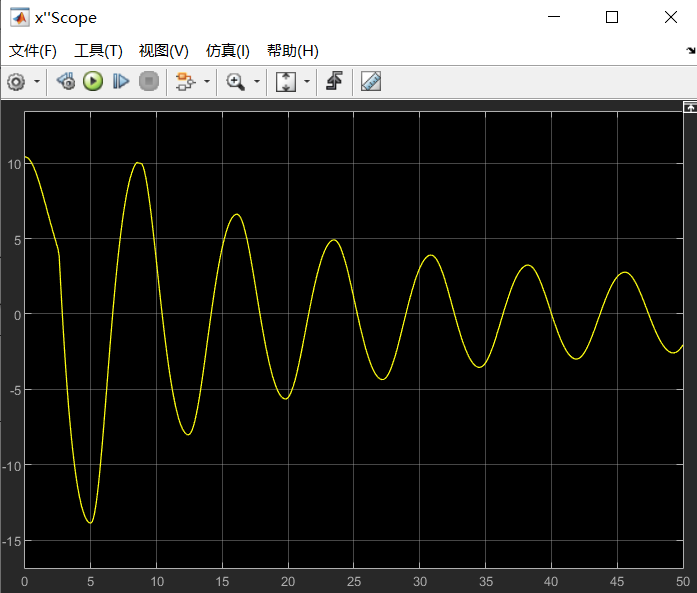
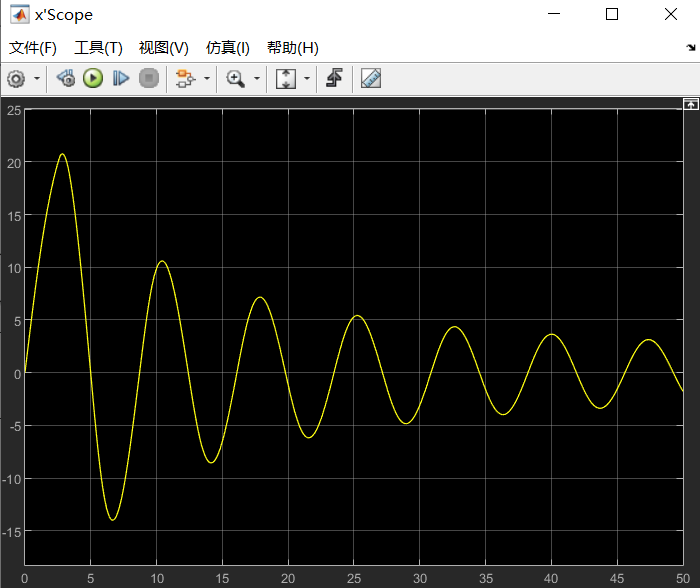
代入参数，蹦极跳的数学模型化为：

其中依据题意，，，使用分段函数表示，表达式为

在Simulink中建立仿真模型如下，仿真模型如下所示。

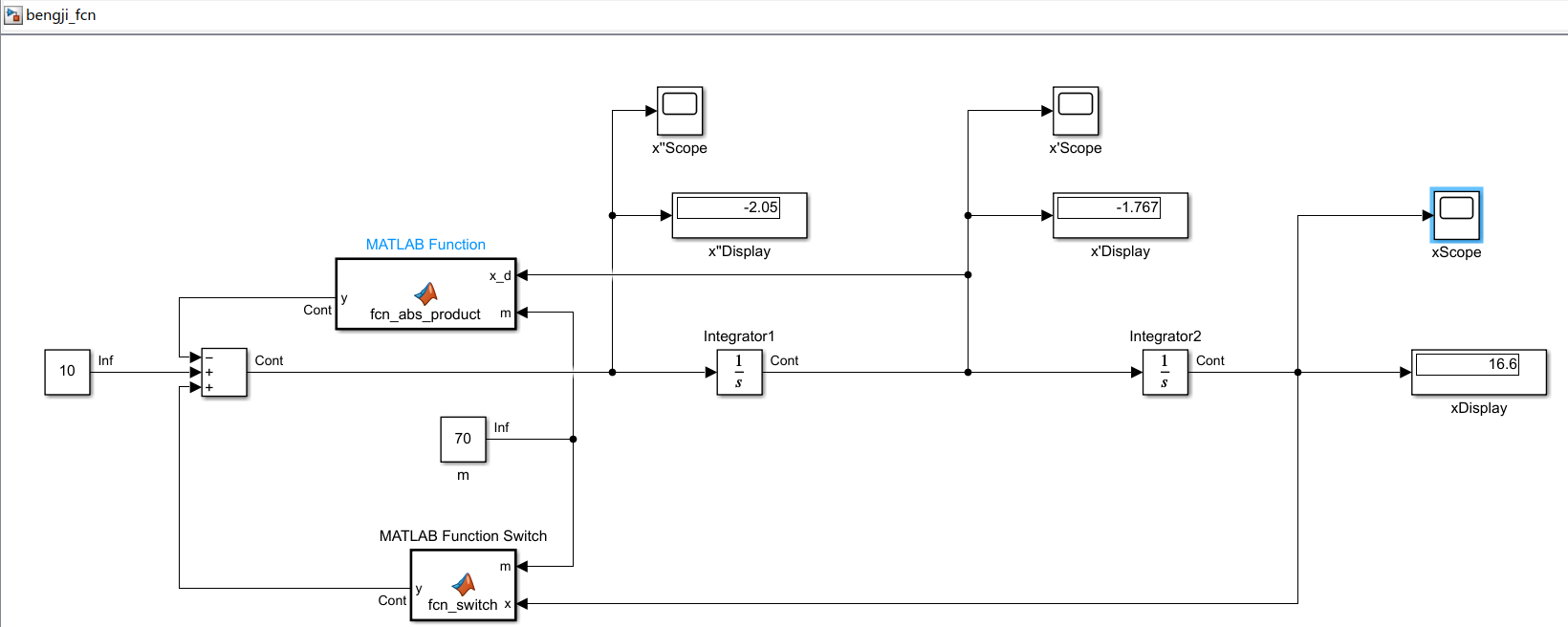
其中，使用Switch模块实现了与相关的分段函数，用abs模块实现了对取绝对值的功能。积分器Integrator1与积分器Integrator2分别求得与。乘法使用product模块实现，除法使用divide模块实现，再使用count模块将各项相加起来。使用display组件和scope组件输出仿真结果和图像。

设置好、的初值，即Integrator1的initial condition设置为0，Integrator2的initial condition设置为-30。仿真时间设置为50s，开始仿真，仿真结果如下所示：

从物理意义上理解，x是人的位置，x‘是人的速度，x’‘是人的加速度。从仿真图像上可以看出，人的位置x在0-20秒间屡次超过20，这是人与地面的安全位置，如果x超过20证明蹦极选手已经碰到地面，有生命风险。

因此得出结论，70kg的人在此条件下蹦极可能会有危险。

## Matlab Function模块形式替换

 将第一部分中的两个乘法模块、两个除法模块、一个分段函数模块和一个绝对值模块共6个模块替换为matlab function函数模块的形式，并进行建模优化，最终建模图示如下：

其中，fun\_switch模块主要是整合了与x相关的分段函数、乘法、除法模块。输入参数为蹦极人员的质量m（70kg）和蹦极人员的位置x，计算，并输出y。具体代码如下：

function y = fcn\_switch(m,x)

if x>0

b = -50\*x;

else

b = 0;

end

y = (b-x)/m;

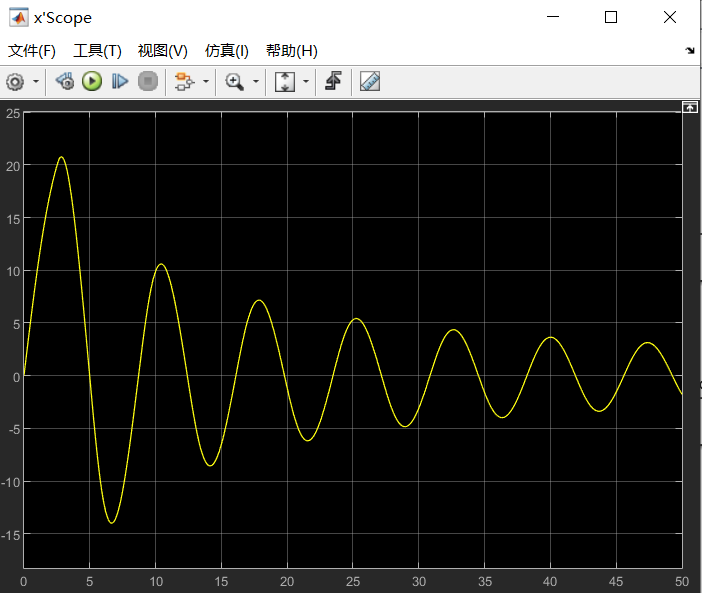
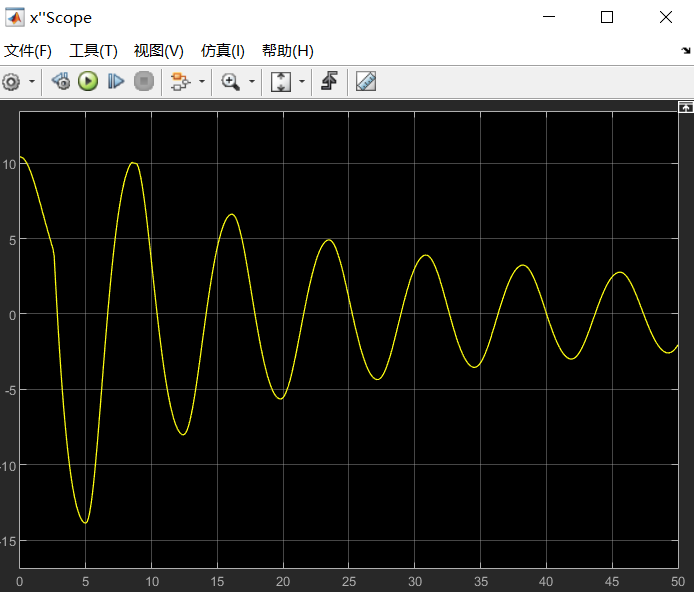
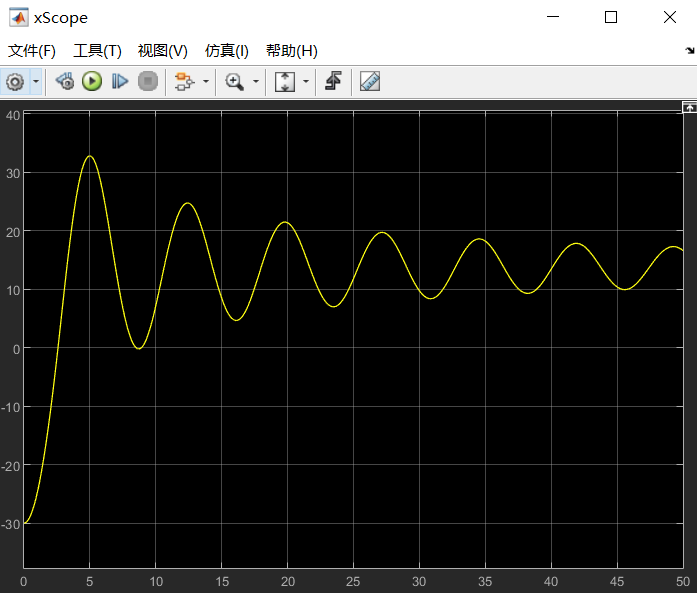
fcn\_abs\_product模块主要是整合了与x’相关的绝对值函数、乘法、除法模块。输入参数为蹦极人员的质量m（70kg）和蹦极人员的速度x’，计算并输出y。具体代码如下：

function y = fcn\_abs\_product(x\_d,m)

a\_x\_d = abs(x\_d);

y = a\_x\_d\*x\_d/m;

其余模块与第一部分相同。Integrator1的initial condition为0，Integrator2的initial condition为-30。仿真时间设置为50s，开始仿真，仿真结果如下所示：

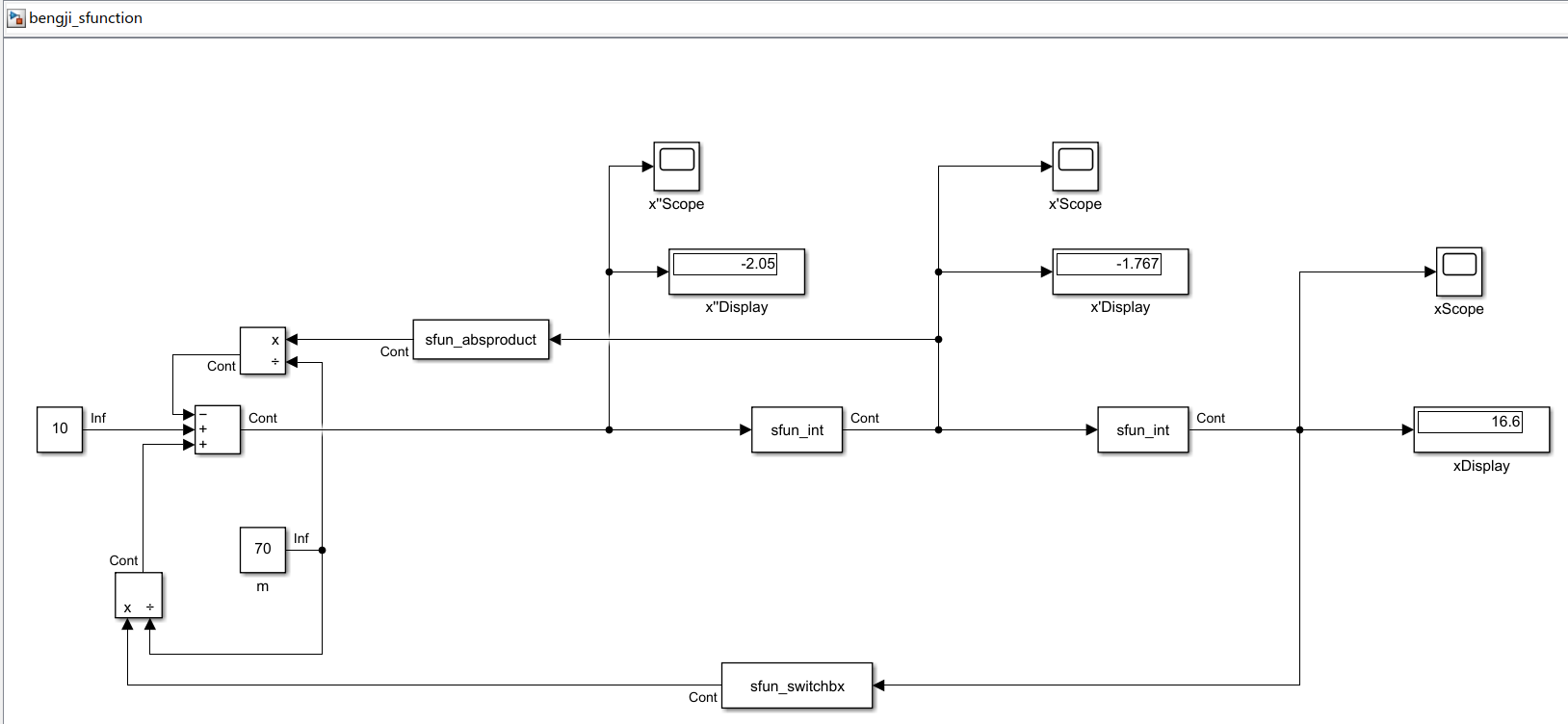


可以看出，将第一部分中的6个模块替换为matlab function函数模块的形式之后，运行结果和第一部分是相同的。

使用matlab function函数模块替换原有模块相当于使用传统的Matlab代码方式简化了建模步骤，而且简化了仿真流程图，使其变得清晰易读。

但每个matlab function模块都需要结合代码阅读才能了解其用途，仿真流程图的理解变得复杂。并且就运行效果而言，这种方法在进行模拟仿真时运行时间更长，运行速度较慢。阅读Matlab官方的指导手册后发现，matlab function模块实际上是旧版本中嵌入式函数功能的延续，每一次仿真它都需要调用Matlab编译器进行编译将Matlab语言翻译成C代码，生成cmex之后再执行，因此效率较低。Matlab官方实际上并不推荐用户在Simulink仿真，尤其是在实时仿真中时大量使用matlab function函数模块。

## S函数模块形式替换

将第一部分中的分段函数模块、绝对值模块、两个积分模块共4个模块替换为S函数模块的形式，并进行建模优化。所有的S函数模块都是基于sfuntmpl这个模板m文件编写而成的。最终建模图示如下：

其中，sfun\_int模块实现的是积分器的功能。系统有一个初始输入initial\_state，它就是积分变量的初始状态变量。这个变量在x’’第一次求积分的积分器设置为0，在x’’第二次求积分的积分器设置为-30。具体代码如下：

function [sys,x0,str,ts] = sfun\_int(t,x,u,flag, initial\_state)

switch flag,

case 0, %初始化例程

[sys,x0,str,ts]=mdlInitializeSizes(initial\_state);

case 1, %连续系统的状态方程

sys=mdlDerivatives(t,x,u);

case 2,

sys=mdlUpdate(t,x,u);

case 3, %输出方程

sys=mdlOutputs(t,x,u);

case 4,

sys=mdlGetTimeOfNextVarHit(t,x,u);

case 9,

sys=mdlTerminate(t,x,u);

otherwise

DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end

function [sys,x0,str,ts]=mdlInitializeSizes(initial\_state)

sizes = simsizes;

sizes.NumContStates = 1; %一个连续状态量

sizes.NumDiscStates = 0;

sizes.NumOutputs = 1;%一个输出，输出积分结果

sizes.NumInputs = 1;

%一个输入，一开始是initial\_state,后来是Simulink仿真中确定的实时值

sizes.DirFeedthrough = 0;

%输出或可变采样时间都不是和输入直接相关，不存在直接馈通

sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0 = initial\_state; %初始化状态变量

str = [];

ts = [0 0];

function sys=mdlDerivatives(t,x,u)

sys = u; %这个积分器的状态方程就是简单的对单变量（上一个状态的值）求积分

function sys=mdlUpdate(t,x,u)

sys = [];

function sys=mdlOutputs(t,x,u)

sys = x; %系统输出就是系统状态

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;

sys = t + sampleTime;

function sys=mdlTerminate(t,x,u)

sys = [];

其中，sfun\_switchbx模块实现的是分段函数功能。系统输入参数为蹦极人员的位置x，计算，并输出y。具体代码如下：

function [sys,x0,str,ts] = sfun\_switchbx(t,x,u,flag)

switch flag,

case 0,

[sys,x0,str,ts]=mdlInitializeSizes;

case 1,

sys=mdlDerivatives(t,x,u);

case 2,

sys=mdlUpdate(t,x,u);

case 3, %使用S函数进行数学计算，只更改输出方程就可以，不需要迭代

sys=mdlOutputs(t,x,u);

case 4,

sys=mdlGetTimeOfNextVarHit(t,x,u);

case 9,

sys=mdlTerminate(t,x,u);

otherwise

DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end

function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates = 0;

sizes.NumDiscStates = 0;

sizes.NumOutputs = 1; %一个输出

sizes.NumInputs = 1; %一个输入，输入蹦极人员的位置x

sizes.DirFeedthrough = 1;% 输出与输入直接相关，存在直接馈通

sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0 = [];

str = [];

ts = [0 0];

function sys=mdlDerivatives(t,x,u)

sys = [];

function sys=mdlUpdate(t,x,u)

sys = [];

function sys=mdlOutputs(t,x,u)

%输出方程，在位置>0的时候，输出-50x-x；否则输出-x

if u>0

sys = -51\*u;

else

sys = -1\*u;

end

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;

sys = t + sampleTime;

function sys=mdlTerminate(t,x,u)

sys = [];

sfcn\_absproduct模块实现的是与求x’绝对值相关的功能。系统输入参数为蹦极人员的速度x’，计算并输出y。具体代码如下：

function [sys,x0,str,ts] = sfun\_absproduct(t,x,u,flag)

switch flag,

case 0,

[sys,x0,str,ts]=mdlInitializeSizes;

case 1,

sys=mdlDerivatives(t,x,u);

case 2,

sys=mdlUpdate(t,x,u);

case 3, %使用S函数进行数学计算，只更改输出方程就可以，不需要迭代

sys=mdlOutputs(t,x,u);

case 4,

sys=mdlGetTimeOfNextVarHit(t,x,u);

case 9,

sys=mdlTerminate(t,x,u);

otherwise

DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end

function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates = 0;

sizes.NumDiscStates = 0;

sizes.NumOutputs = %一个输出

sizes.NumInputs = 1; %一个输入，输入蹦极人员的位置x

sizes.DirFeedthrough = 1;% 输出与输入直接相关，存在直接馈通

sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0 = [];

str = [];

ts = [0 0];

function sys=mdlDerivatives(t,x,u)

sys = [];

function sys=mdlUpdate(t,x,u)

sys = [];

function sys=mdlOutputs(t,x,u)

%输出方程，输出速度的绝对值与速度的乘积

sys = abs(u)\*u;

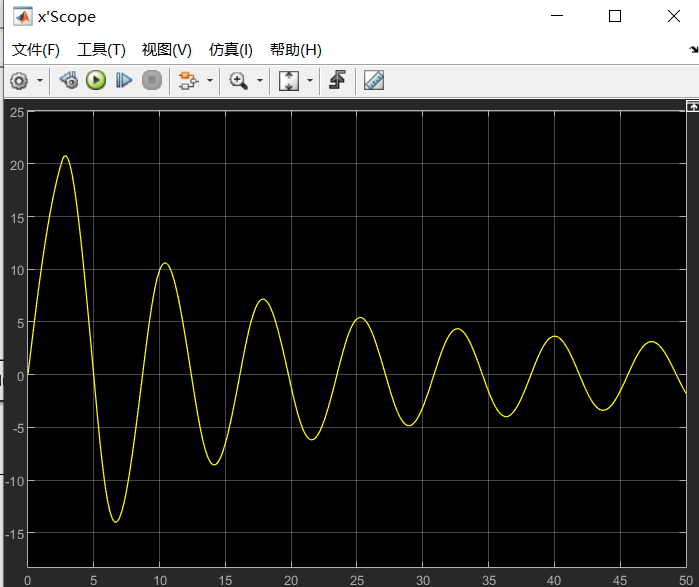
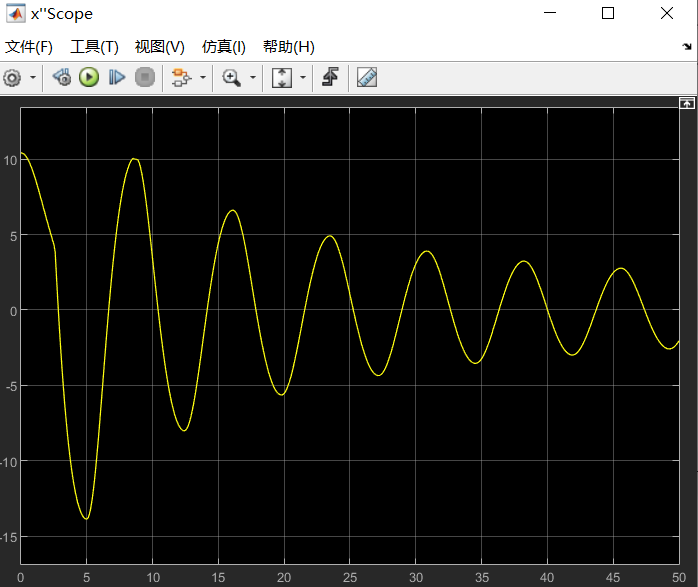
function sys=mdlGetTimeOfNextVarHit(t,x,u)

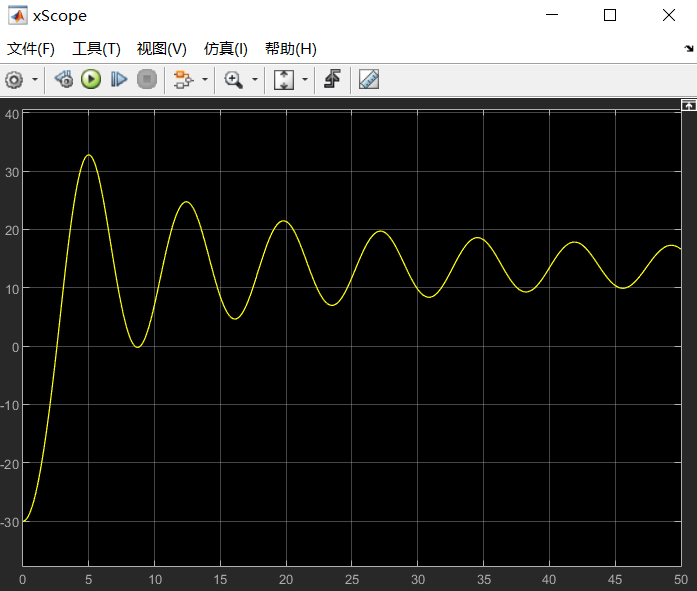
sampleTime = 1;

sys = t + sampleTime;

function sys=mdlTerminate(t,x,u)

sys = [];

 其余模块与第一部分相同。设置好两个S函数的initial\_state值（分别为0和-30）。仿真时间设置为50s，开始仿真，仿真结果如下所示：



可以看出，以S函数模块的形式替换部分模块后，运行结果和前两次建模仿真结果仍然是相同的。

S函数模块的使用更加复杂，它提供的系统仿真流程更加全面、准确，包括对状态变量、输入输出口个数的初始化设置，状态变量的更新计算，输出计算和终止时的动作定义等等，这一套系统仿真从定义上理解起来比较复杂，但是运行效果比matlab function模块更快。

在实际使用中，发现s函数模块默认是单输入单输出系统模块。如果想利用s函数模块实现多输入多输出，可以通过Mux跟Demux对信号进行组合和分离。