第十章　用S－函数扩展Simulink

通过第八、第九章的学习，用户对用Simulink建模的基本思想已经有了清晰的认识，Simulink为用户提供了许许多多的内置库模块，用户只需使用这些库模块构建系统即可。但在实际应用中，用户通常会发现有些过程用Simulink的库模块不容易建模。这时，可以使用S－函数来扩展Simulink。S－函数结合了Simulink框图图形化的特点和MATLAB编程灵活方便的优点，从而给用户提供增强和扩展Simulink的强大机制，同时它也是使RTW（Real Time Workshop）实现实时仿真的关键。

10.1 S－函数概述

10.1.1　S－函数的基本概念

S－函数是System function系统函数的简称，是指采用非图形化（即计算机语言，而非Simulink系统模块）的方式描述的功能模块。在MATLAB中，用户除了可以使用MATLAB代码编写S－函数以外，还可以使用C、C＋＋、FORTRAN或Ada语言编写S－函数，只不过用这些语言编写程序时需要用编译器生成动态连接库（DLL）文件，然后在Simulink中直接调用。

S－函数是由一种特殊的语法构成的，用来描述并实现动态系统的。它采用一种特殊的调用语法，使函数和Simulink求解器进行交互。这种交互与求解器和Simulink仿真模型间的交互相类似：S－函数接受来自Simulink求解器的相关信息，并对求解器发出的命令做出适当的响应。

S－函数作为与其它语言结合的接口，可以使用这个语言所提供的强大功能。例如，使用MATLAB语言编写的S－函数称为M文件S－函数，它可以充分利用MATLAB所提供的丰富资源，方便地调用各种工具箱函数和图形函数；而使用C语言编写的S－函数被称为C－MEX文件S－函数，则可以实现对操作系统和外部设备等的访问，也可以提供与操作系统的接口。另外，S－函数可以使用其他多种语言编写，因此可以实现代码的移植，即将已有的代码结合进来，而不需在Simulink中重新实现算法。

S－函数中采用非图形化的方式描述系统，其内部采用文本方式输入描述系统的公式、方程，这种方式非常适合复杂动态系统的数学描述，且可以在仿真过程中对仿真进行精确的控制。

10.1.2　如何使用S－函数

在动态系统仿真中，要想将S－函数加入Simulink仿真模型中，用户需要将User Defined Functions模型库中的S－Function模块拖进该模型中。S－Function模块是一个单输入单输出模块，如果有多个输入与输出信号，用户需要使用Mux模块和Demux模块对信号进行组合或分离。S－Function模块仅仅是以图形的方式提供给用户一个S－函数的使用接口，在它的参数设置对话框中仅包含S－函数的名称及函数所需的参数列表（见图10.1），而S－函数所实现的功能则由S－函数源文件描述，S－函数源文件必须由用户自行编写。

使用S－函数的步骤如下：

一、在系统的Simulink仿真框图中添加S－function模块，并进行正确的设置；

二、创建S－函数源文件。创建S－函数源文件的方法有多种。用户可以按照S－函数的语法格式自行编写代码，但是这样做很麻烦，且容易出错。Simulink在S－function Examples模型库中为用户提供了针对不同语言的很多S－函数模板和例子，用户可以根据自己的需要修改相应的模板或例子即可完成S－函数源文件的编写工作；

三、在系统的Simulink仿真框图中按照定义好的功能连接输入输出端口。

这里需要说明的是，S－function模块中S－函数名称必须和用户建立的S－函数源文件的名称完全相同，S－function模块中的S－函数参数列表必须按照S－函数源文件中的参数顺序赋值，且参数之间需要用逗号隔开。另外，用户也可以使用子系统封装技术对S－函数进行封装，这样做的好处是可以增强系统模型的可读性。

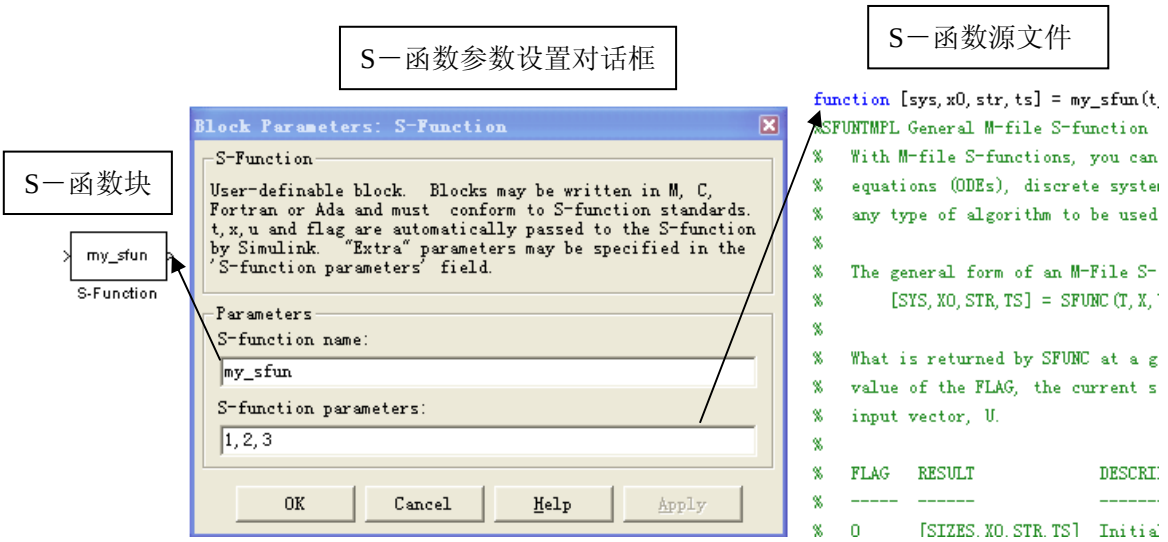


图 10.1　S－函数模块、参数设置对话框及其源文件的关系

为了方便用户编写 C MEX S 函数，Simulink 的 User Defined Functions 模型库中为用户编写 C MEX S 函数提供了一个图形化的集成的图形开发环境 S－function builder。用户只需在 S－function builder 中相应的位置写入相应的名称和代码即可编译成相应的 MEX 文件。

为了使读者尽快掌握 S－函数的使用步骤，先举一个简单的例子。

例 10.1 使用 S－函数实现系统：$y=2*u$。

解：一、在 Simulink 模型框图中添加 S－function 模块，打开 S－funfction 模块的参数设置对话框，参数 S－function name 需设置为 timestwo。

二、创建 S－函数源文件。

1、打开 M 文件 S－函数模板文件 Sfuntmpl.m，并在指定目录下另存为 timestwo.m。打开 M 文件 S－函数模板文件 Sfuntmpl.m 的方法由两种：

方法一，在 MATLAB 命令窗口键入 edit Sfuntmpl 即可；

方法二，在 Simulink 浏览器中寻找 S－function demos 模型库，其中包含各种语言编写的 S－函数例子和模板。对于本例，用户只需点击 M－file S－functions 就可以看到 M 文件 S－函数模板文件 Sfuntmpl.m 和几个编程例子，双击模板文件 Sfuntmpl.m 即可。

2、修改模板。找到函数 mdlInitializeSizes，修改以下代码：

size.NumOutputs=1;

size.NumIutputs=1;

找到函数 mdOutputs，将代码 sys＝[ ]改为 sys＝2*u

至此，已经写好了该 S－函数的源文件，保存修改即可。

三、在 Simulink 模型框图中按要求添加并连接各个模块。系统 Simulink 仿真模型如图 10.2 所示。

四、运行仿真。仿真结果可以从 Scope 模块中观察。结果如图 10.2 所示。

10.1.3　与 S-函数相关的术语

用户对 S－函数相关的术语的理解对于了解 S－函数的工作原理、编写 S－函数源文件是非常有用的。

一、仿真例程（Routines）

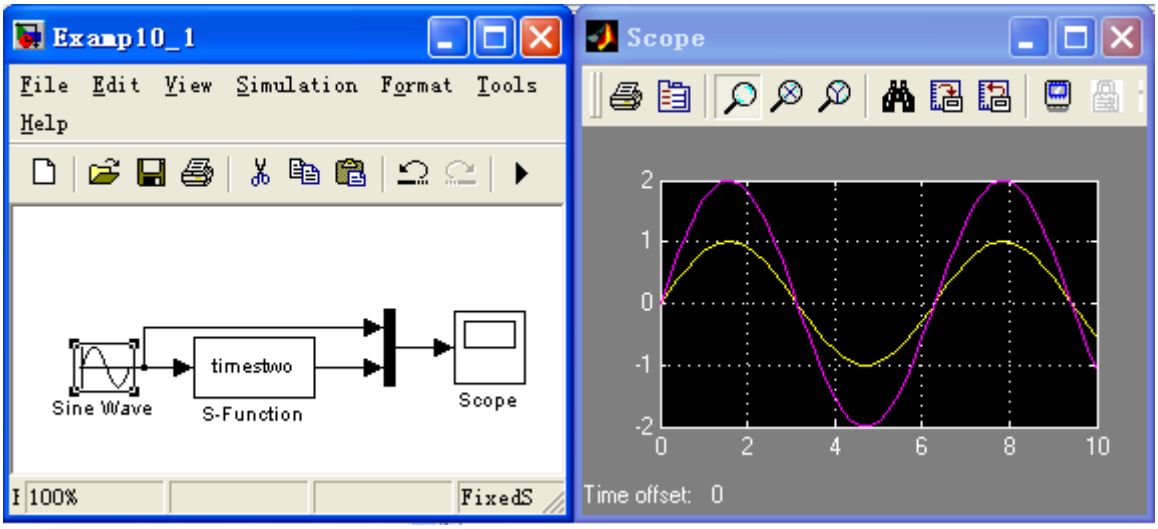


图 10.2　例 10.1 系统 Simulink 仿真模型及其结果

Simulink 在仿真的不同阶段调用 S－函数的不同的功能函数以完成不同的任务。对于 M 文件 S－函数，Simulink 通过传递一个 flag 参量给 S－函数，通知 S－函数当前所处的仿真阶段，以便执行相应的功能函数。S－函数的功能函数包括初始化、计算输出、更新离散状态、计算导数、结束仿真等。这些功能函数称为仿真例程或回调函数（call back function）。在写 M 文件 S－函数时，用户只需用 MATLAB 语言为每个 flag 对应的功能函数编写代码即可。表 10.1 列出了各仿真阶段的功能函数及其对应的 flag 值。

表 10.1　各仿真阶段的仿真例程及其 flag 值

仿真阶段	S－函数仿真例程（回调函数）	Flag 值（M 文件 S－函数）
初始化	mdlInitializeSizes	0
计算下一个采样点	mdlGetTimeofNextVarHit	4
计算输出值	mdlOutputs	3
更新离散状态	mdlUpdate	2
计算导数	mdlDerivatives	1
结束仿真	mdlTerminate	9

二、直接馈入（Direct Feedthrough）

直接馈入是指模块的输出或采样时间（变速率模型）直接由其某个输入端口控制。判断一个 S－函数是否具有直接馈入的标准是：

某时刻系统的输出 y 包含该时刻系统的输入 u，即计算系统输出的方程中包含输入变量 u；

若系统是一个变采样时间系统，且下一个采样点的计算与输入 u 有关。

馈入标志的设置不仅关系到系统模型中的系统模块的执行顺序，而且关系到对代数环的检测和处理。因此正确设置馈入标志是非常重要的。

三、采样时间和偏移量（Sample time & offsets）

M 文件和 C MEX 文件 S－函数都允许用户十分方便地设定 S－函数被调用的时间。

采样时间在离散系统中控制采样点的间隔，偏移量则用于延迟采样点。一个采样点对应的时间值由下列公式计算：

$$TimeHit=n\times period+offset$$

其中，n 表示当前仿真步，是整数。

如果用户定义了一个离散采样时间，Simulink 就会在所定义的每个采样点调用 S－函数的 mdlOutputs 和 mdlUpdate 例程。

对于连续时间系统，采样时间和偏移量的值均应设置为零。采样时间还可以继承来自驱动模块、目

标模块或系统最小的采样时间，这种情况下，采样时间值设置为-1。

四、动态输入（Dynamically sized inputs）

S－函数支持动态可变维数的输入。S－函数的输入变量的维数取决于驱动 S－函数模块的输入信号维数。仿真开始时，通过 size 或 length 函数确定输入信号的维数。然后就可以利用这个维数来估计连续状态数目、离散状态数目和输出向量的维数。

在 M 文件 S－函数中动态设置输入维数时，应该把 sizes 数据结构的对应成员设置为－1。

比如在例 10.1 中，如果将函数 mdlInitializeSizes 中的代码设置成 size.NumOutputs=－1; size.NumIutputs=－1; 则当输入是两维信号，分别为幅值为 1 和 3 的正弦信号，系统的输出也是两维信号，结果如图 10.3 所示。

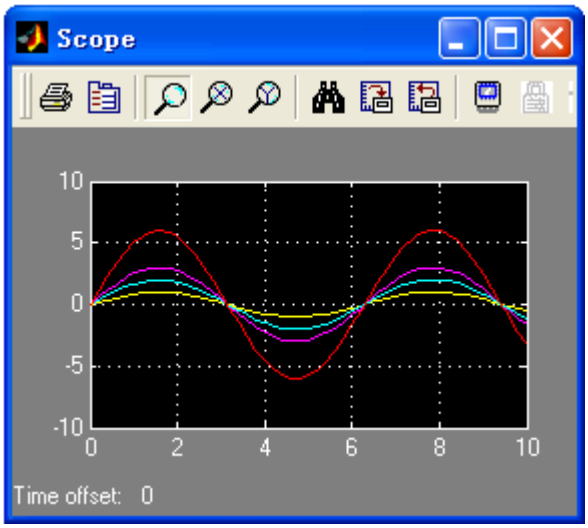


图 10.3　例 10.1 进行动态输入设置后的仿真结果

10.2　S-函数工作原理

了解 S－函数的工作原理对于用户掌握 S－函数的编写方法是非常有用的，对用户对于 Simulink 的仿真原理的理解也是很有帮助的。本节介绍 S－函数的工作原理。

在具体介绍 S－函数的工作原理之前，首先需要回顾一下 Simulink 模块的工作原理。

Simulink 中的每个模块都有三个基本元素：输入向量、状态向量和输出向量，分别表示为 u，x 和 y。图 10.4 反映了它们之间的关系。在 Simulink 模块的三个元素中，状态向量是最重要的，也是最灵活的概念。在 Simulink 中状态向量可以分为连续状态、离散状态或两者的结合。输入、输出及状态的关系可以用状态方程描述：

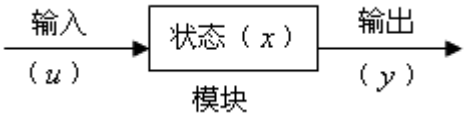


图 10.4　Simulink 模块的基本模型

输出方程：$y = f_o(t, \boldsymbol{x}, \boldsymbol{u})$

连续状态方程：$d\boldsymbol{x} = f_d(t, \boldsymbol{x}, \boldsymbol{u})$

离散状态方程：$\boldsymbol{x}_{k+1} = f_u(t, \boldsymbol{x}, \boldsymbol{u})$

其中 $\boldsymbol{x} = \left[d\boldsymbol{x} \quad \boldsymbol{x}_{k+1} \right]$。

Simulink 在仿真时将上述方程对应不同的仿真阶段，它们分别是计算模块的输出、更新离散状态、计算连续状态的微分。在仿真开始和结束，还包括初始化和结束仿真两个阶段。在每个阶段，Simulink 都反复地调用模块。

至此，读者已经接触到了几个关于仿真的概念：仿真步长（Simulation step）、仿真阶段（Simulation stage）。为了深入了解 S－函数的工作原理，还需了解一个概念：仿真循环（Simulation loop）。一个仿真循环就是由仿真阶段按一定顺序组成的执行序列。对于每个模块，经过一次仿真循环就是一个仿真步长，而在同一个仿真步长中，模型中各模块的仿真按照事先排好的顺序依次执行。这个过程可以用图 10.5 表示。

从图中可以看出，在仿真开始时，Simulink 首先对模型进行初始化，此阶段不属于仿真循环。在所有模块都初始化后，模块进入仿真循环，在仿真循环的每个阶段，Simulink 都要调用模块或者 S－函数。

由于在积分时，对仿真步长有要求，所以此时需要将仿真步长细化。完成一个仿真循环就进入下一个仿真步长，如此循环直至仿真结束。

在调用模型中的 S－函数时，Simulink 会调用用户定义的 S－函数的例程来实现每个仿真阶段要完成的任务。这些任务包括：

一、初始化：仿真开始前，Simulink 在这个阶段初始化 S－函数，完成的主要工作包括：

1、初始化包含 S－函数所有信息的结构体 SimStruct；

2、确定输入输出端口的数目和大小；

3、确定模块的采样时间；

4、分配内存和 Sizes 数组。

二、计算下一个采样时刻。如果模型使用变步长求解器，那么就需要在当前仿真步长内确定下一个采样点的时间，也即下一个仿真步长的大小；

三、计算输出：计算所有输出端口的输出值。

四、更新离散状态：此例程在每个仿真步长处都要执行一次，为当前时间的仿真循环更新离散状态；

五、数值积分：这个阶段只有模块具有连续状态和非采样过零点时才会存在。如果 S－函数存在连续状态，Simulink 就在细化的小时间步长中调用 S－函数的输出(mdlOutputs)和微分(mdlDerivatives)例程。如果存在非采样过零点，Simulink 将调用 S－函数中的输出（mdlOutputs）和过零检测（mdlZeroCrossngs）例程，以定位过零点。

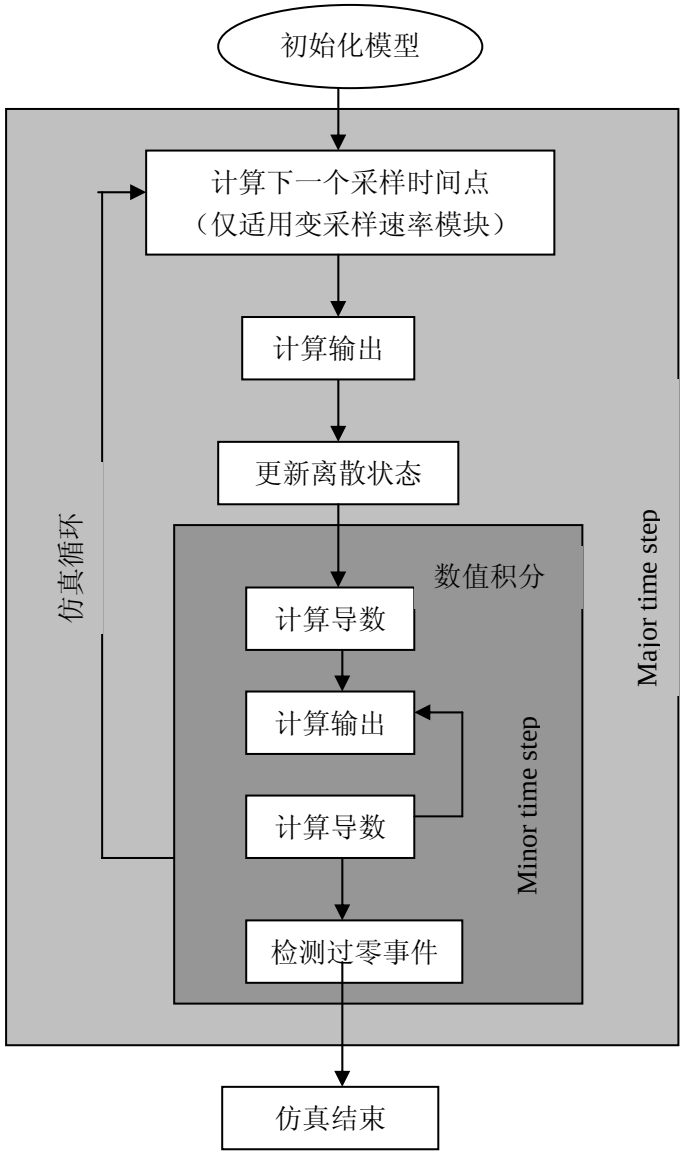


图 10.5 S－函数仿真流程

10.3　编写 M 文件 S－函数

由前面的介绍可以知道，S－函数是由一系列仿真例程组成的。这些仿真例程就是 S－函数特有的语法结构，用户编写 S－函数的任务就是在相应的例程中填写适当的代码，供 Simulink 及 MATLAB 求解器调用。M 文件 S－函数结构明晰，易于理解、书写方便，可以调用丰富 MATLAB 函数，所以在实际工作中得到了广泛的应用。

M 文件 S－函数利用 Flag 标志控制调用例程函数的顺序。各仿真阶段的仿真例程及对应的标志值如表 10.1 所示。

M 文件 S－函数的仿真流程同图 10.5 介绍的 S－函数的仿真流程。在初始化阶段，通过标志 0 调用 S－函数，并请求提供输入输出个数、初始状态和采样时间等信息。然后，仿真开始。下一个标志为 4，请求 S－函数提供下一步的采样时间（这个例程在单采样速率系统下不被调用）。接着 flag＝3 计算模块的输出，flag＝2 更新离散状态，当需要计算连续状态导数时 flag＝1。然后求解器使用积分例程计算状态的值。计算状态导数和更新离散状态之后通过标志 3 计算模块的输出。这样就完成了一个仿真步长的工作。当到达结束时间时，采用标志 9 完成结束前的处理工作。

10.3.1 M 文件 S－函数模板

Simulink 为用户提供了各种语言编写 S－函数的模板文件。这些 S－函数的模板文件中定义了 S－函数的框架结构，用户可以根据自己的需要修改。

编写 M 文件 S－函数时，需要使用 M 文件 S－函数模板文件 sfuntmpl.m 文件。该文件包含了所有的 S－函数的例程，及包含 1 个主函数和 6 个子函数。在主函数中，程序使用一个多分支语句（Switch－case）根据标志将执行流程转移到相应的例程函数。主函数的参数 Flag 标志值是由系统（Simulink 引擎）调用时给出的。读者可以打开并阅读该 M 文件 S－函数模板文件。

一、打开模板文件的方法由两种，用户可以在 MATLAB 命令窗口中键入：

```
>> edit sfuntmpl
```

或者双击 User-defined Function \S-function Examples\M-file S-functions\Leveal-1 M-file S-functions1\ Leveal-1 M-file template 模块。

二、M 文件 S－函数模板文件代码

M 文件 S－函数模板文件的代码如下：

```
%主函数
function [sys,x0,str,ts] = sfuntmpl(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,
    sys=mdlOutputs(t,x,u);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end                                  % 主函数结束，下面是各个子函数，即各个仿真例程

%  初始化例程子函数：提供状态、输入、输出、采样时间数目和初始状态的值。
function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;                    % 生成 sizes 数据结构
sizes.NumContStates  = 0;            %  连续状态数，缺省为 0
sizes.NumDiscStates  = 0;            %  离散状态数，缺省为 0
sizes.NumOutputs     = 0;            %  输出量个数，缺省为 0
sizes.NumInputs      = 0;            %  输入量个数，缺省为 0
sizes.DirFeedthrough = 1;            %  有无直接馈入，有取 1，无取 0，缺省为 1
sizes.NumSampleTimes = 1;            %  采样时间个数，至少取 1

sys = simsizes(sizes);               % 返回 sizes 数据结构所包含的信息
```

```
x0  = [];                          % 设置初始状态
str = [];                           % 保留变量，置为空矩阵
ts  = [0 0];                        % 采样时间：[采样周期  偏移量]，采样时间取 0 表示为连续系统

% 计算导数例程子函数：计算连续状态的导数，用户需在此例程输入连续状态方程。
% 该子函数可以不存在。
function sys=mdlDerivatives(t,x,u)

sys = [];                %  sys 表示连续状态导数


% 状态更新例程子函数：计算离散状态的更新。
% 用户除了需在此输入离散状态方程外，还可以输入其它每个仿真步长都有必要执行的代码。
% 该子函数可以不存在。
function sys=mdlUpdate(t,x,u)

sys = [];                %  sys 表示下一个离散状态，即 x（k＋1）


% 计算输出例程子函数：  计算模块输出。该子函数必须存在，用户在此输入系统的输出方程。
function sys=mdlOutputs(t,x,u)

sys = [];                 %  sys 表示系统输出 y


%  计算下一个采样时间，  只有变采样时间系统才调用此仿真例程。
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;                %  设置下一次的采样时间是 1s 以后
sys = t + sampleTime;            %   sys 表示下一个采样时间点


% 仿真结束调用的例程函数：用户需在此输入结束仿真所需要的必要工作。
function sys=mdlTerminate(t,x,u)

sys = [];
```

三、M 文件 S－函数模板文件的几点说明

主函数包含四个输出参数：sys 数组返回某个子函数，它的含义随着调用子函数的不同而不同；x0 为所有状态的初始化向量；str 是保留参数，总是一个空矩阵；Ts 返回系统采样时间。

主函数的四个输入参数分别是采样时间 t，状态 x，输入 u 和仿真流程控制标志变量 flag。

输入参数后面还可以附加一系列用户仿真需要的参数。

编写用户自己的 S－函数时，应将函数名改为 sfuntmpl 改为 S－function 模块中设置的函数名。

读者可能已经发现一个令人困惑的问题：不论在哪个仿真阶段，例程子函数的返回变量都是 sys。要

搞清楚这个问题，还要回到 Simulink 如何调用 S－函数上来。前面讲过，Simulink 在每个仿真步长的仿真循环中的每个仿真阶段都要调用 S－函数。在调用时，Simulink 不但根据所处的仿真阶段为 flag 传入不同的值，还会为返回变量 sys 指定不同的角色。即是说尽管是相同的 sys 变量，但在不同的仿真阶段其意义是不相同的，这种变化由 Simulink 自动完成。

10.3.2　M 文件 S－函数的应用举例

了解了 M 文件 S－函数模板文件的代码、代码中各个部分完成的功能及各参数的含义后，用户可以着手利用 S－函数进行系统仿真了。下面我们使用 M 文件 S－函数实现几种不同的系统。

一、含用户参数的简单系统

M 文件 S 函数除了模板文件中要求的几个必需的参数，还可以加入用户自定义的参数，自定义参数需要在 S－函数的输入参数中列出。在含用户自定义参数的 S 函数中，主函数要做适当的修改以便将自定义参数传递到子函数中，子函数也需要相应的修改以便接受自定义参数。在编写 S－函数时，应能区分哪些参数会影响哪一个子函数的执行，要针对这些参数做相应的修改。还需注意的一点是，S－function 模块中的参数设置对话框中的参数输入顺序应与 S－函数中自定义参数的顺序相同。

例 10.2　用 S－函数实现 gain 模块：增益值作为 S－函数用户自定义参数输入。

解：（1）编写 S－函数的源文件

修改 M 文件 S－函数的主函数：增加自定义参数，采用新的函数名：

```
function [sys,x0,str,ts] = sfun_var_gain(t,x,u,flag,gain)
```

由于增益参数只是用来计算输出值的，因而对初始化例程和计算输出例程子函数做修改，其他例程均不需调用，不用做修改

```
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(gain);
case 3,
    sys=mdlOutputs(t,x,u,gain);
```

修改初始化例程子函数：

```
function [sys,x0,str,ts]=mdlInitializeSizes(gain)
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
```

定义计算输出例程子函数

```
function sys=mdlOutputs(t,x,u,gain)
sys = gain*u;        % 输出＝增益×输入
```

（2）建立如图 10.6 所示的系统仿真模型，将自定义参数设置为 3，运行仿真，仿真结果如图 10.6 所示，验证了 S－函数的正确性。

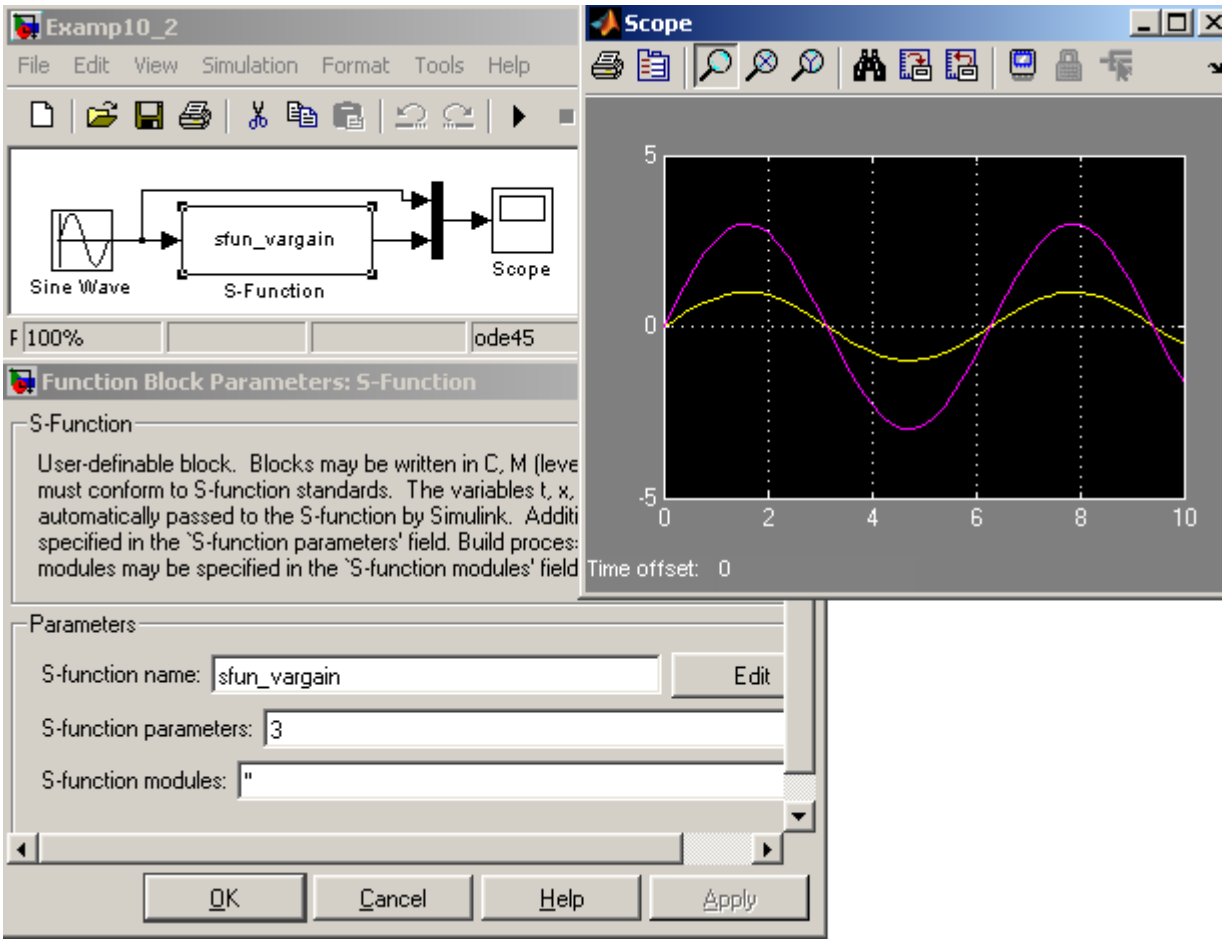


图 10.6 含用户参数的简单系统仿真

二、连续系统的 S－函数描述

用 S－函数实现一个连续系统时，首先需要修改初始化例程子函数 mdlInitialSizes，包括正确设置连续状态个数、状态初始值和采样时间。其次，需要编写计算导数例程子函数，将状态的导数向量通过 sys 变量返回。如果存在多个系统状态，可以通过索引 x(1),x(2)等得到各个状态，此时，变量 sys 即为一个向量，包含所有各个连续状态的导数。最后，也需在计算输出例程子函数中编写系统的输出方程。下面举例说明。

例 10.3　试用 S－函数对蹦极跳系统做仿真分析。

蹦极跳是一种挑战身体极限的运动，蹦极者系着一根弹力绳从高处的桥梁或山崖向下跳。如果蹦极者系在一个弹性系数为 k 的弹力绳索上。定义绳索下端的初始位置为 0，则蹦极者受到的弹性力是

$b(x)=\begin{cases}-kx & x>0\\ 0 & x\le 0\end{cases}$，整个蹦极跳系统的数学模型为

$$m\ddot{x}=mg+b(x)-a_1\dot{x}-a_2|\dot{x}|\dot{x}$$

其中 m 为物体的质量，g 为重力加速度，x 为物体的位置，第二项是物体受到的弹性力，第三与第四项表示空气的阻力。

设桥梁距离地面高度为 50 米，绳索长度 30 米，

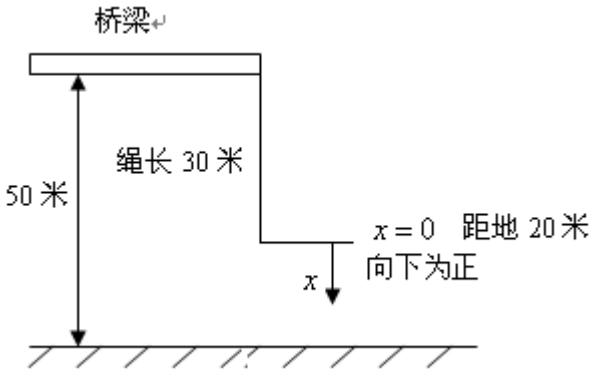


图 10.7 蹦极跳系统坐标系

由此可知蹦极者初始站在桥上时的初始位置 $x(0)=-30$，初始速度 $\dot{x}(0)=0$；根据对蹦极跳系统的描述，我们可以分析出该系统的坐标系如图 10.7 所示。其余参数分别取为 $k=50$， $a_1=a_2=1$， $m=70kg$，$g=10m/s^2$。下面我们分析此蹦极跳系统对于 70kg 重的蹦极者是否安全。

解：为了使用 S－函数对蹦极跳系统进行仿真计算，需将其数学模型转变为如下状态方程形式

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = g + b(x_1)/m - a_1 x_2/m - a_2 |x_2| x_2/m \end{cases}$$，其中 $x_1 = x, x_2 = \dot{x}$

（1）编写 S－函数的源文件

① 修改 M 文件 S－函数的主函数：增加自定义参数，采用新的函数名：

```
function [sys,x0,str,ts] = jumping(t,x,u,flag,l,m,d,k)
```

由于增添了自定义参数来计算导数和输出值的，因而对初始化例程子函数、计算导数和输出例程的子函数做修改

```
case 0,
[sys,x0,str,ts]=mdlInitializeSizes(l,m,d,k);
case 1,
    sys=mdlDerivatives(t,x,u,l,m,d,k);
case 3,
    sys=mdlOutputs(t,x,u,l,m,d,k);
```

修改初始化例程子函数：

```
function [sys,x0,str,ts]=mdlInitializeSizes(l,m,d,k)
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
x0  = [-l;0];
```

定义计算导数例程子函数

```
function sys=mdlDerivatives(t,x,u,l,m,d,k)
if x(1)>=0
    b=-k*x(1);
else
    b=0;
end
sys = [x(2);10+b/m-1/m*x(2)-1/m*abs(x(2))*x(2)];
```

定义计算输出例程子函数

```
function sys=mdlOutputs(t,x,u,l,m,d,k)
sys = [d-l-x(1)];
```

（2）建立如图 10.8 所示的系统仿真模型，并按正确的顺序输入自定义参数 l,m,d,k 的值，运行仿真，仿真结果也如图 10.8 所示。由仿真曲线可知，本蹦极跳系统对于 70kg 重的蹦极者来说是非常危险的，因为仿真结果显示蹦极者有触地的危险，为了满足大体重的蹦极爱好者的要求，必须对系统参数做适当的调整。具体如何调整，我们下章再做介绍。

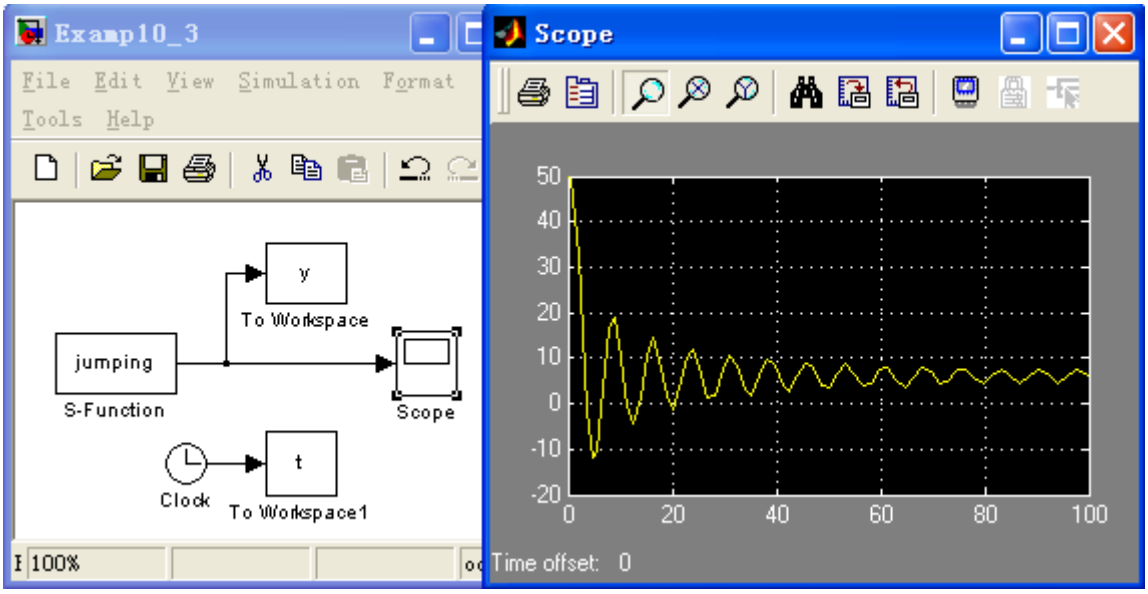


图 10.8　蹦极跳系统的仿真模型及仿真结果

三、离散系统的 S－函数描述

用 S－函数实现离散系统时，首先要对初始化例程子函数 mdlInitializeSizes 做适当修改，包括声明离散状态的个数、对状态进行初始化、确定采样时间等，其次需要定义状态更新和计算输出例程子函数 mdlUpdate 和 mdlOutputs，分别需要输入表示离散系统的离散状态方程和输出方程。

例 10.4 编写 S－函数实现输出对输入的单位延迟，即 $y(k+1)=u(k)$。

解：单位延迟系统的状态方程可以表示为 $x(k+1)=u(k)$， $y(k)=x(k)$。

（1）编写 S－函数的源文件

① 修改 M 文件 S－函数的主函数：采用新的函数名：

```
function [sys,x0,str,ts] = sfun_unitdelay(t,x,u,flag)
```

② 修改初始化例程子函数：

```
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 1;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

x0  = 0;
ts  = [0.1 0];
```

定义状态更新例程子函数

```
function sys=mdlUpdate(t,x,u)
sys = u;
```

定义计算输出例程子函数

```
function sys=mdlOutputs(t,x,u)
sys = x;
```

（2）建立如图 10.9 所示的系统仿真模型，运行仿真，仿真结果如图 10.9 所示，验证了 S－函数的正确性。

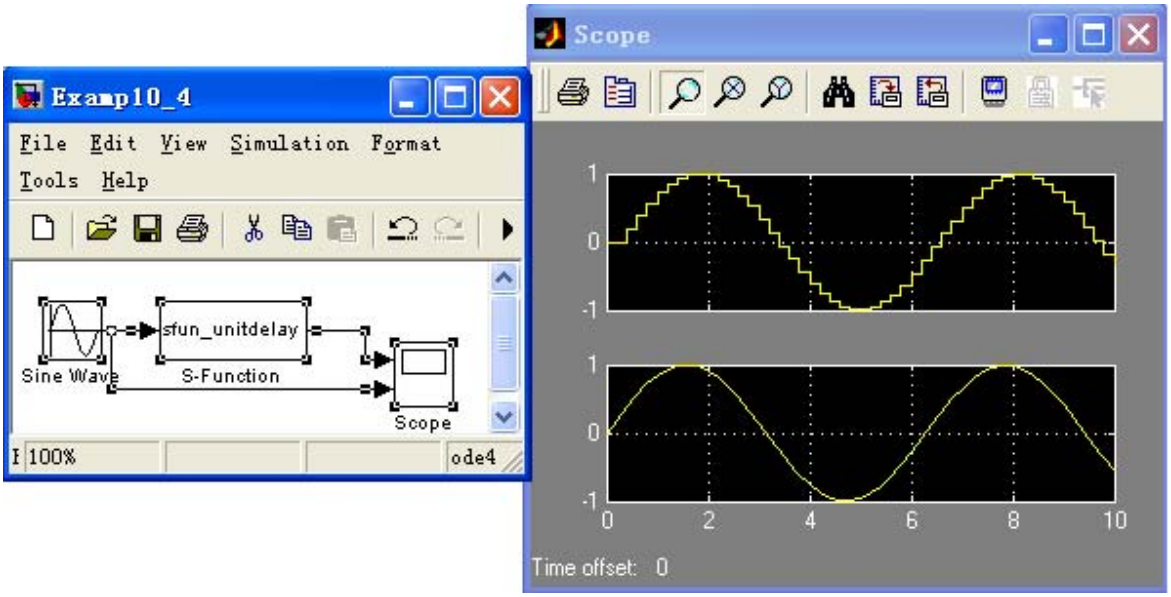


图 10.9　单位延迟系统的仿真模型及结果

四、混合系统的 S－函数描述

既包含离散状态，又包含连续状态的系统称为混合系统。这里我们介绍如何用 S－函数描述一个连续积分器后接一个离散单位延迟的混合系统。

例 10.5　编写 S－函数实现一个连续积分器后接一个离散单位延迟的混合系统。

解：首先，在初始化例程要定义两个采样时间：连续采样时间和离散采样周期，于是在这两个采样时间决定的采样点，Simulink 都会调用 S－函数源文件，并依次用 flag 来指定要调用的例程。

在混合系统中，与离散状态和连续状态有关的 S－函数例程都有可能执行，但各自的执行时间不同。连续状态的导数计算是在每个微时间步都要进行的，但离散状态的更新以及系统的输出计算，只在离散采样点到达时才完成。但这一点 Simulink 是无法区分什么时候该更新什么时候不该更新。因为 Simulink 从初始化获得的信息只是该系统既有离散状态又有连续状态，据此只能决定该函数的仿真循环应该包含连续状态的微分计算和离散状态的更新例程，没有足够的信息来判断当前时刻是否是离散采样点。

这时常用的技巧是，在 mdlUpdate 和 mdlOutputs 中由程序判断当前时刻是否是离散采样点。mdlUpdate 例程中完成此功能的语句是：

```
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);      % 是离散采样点，将离散状态更新为当前连续状态
else
  sys = [];        % 不是离散采样点，离散状态保持不变
end
```

mdlOutputs 例程中完成此功能的语句类似，这里就不赘述了。

本系统的S－函数源文件的代码如下：

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
dperiod = 1;          % 设置离散采样周期和偏移量
doffset = 0;

switch flag
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset);
  case 1
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u,dperiod,doffset);
  case 3
    sys=mdlOutputs(t,x,u,doffset,dperiod);
   case 9
    sys = [];       % do nothing
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
  end

function [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset)

sizes = simsizes;
sizes.NumContStates  = 1;           % 一个连续状态
sizes.NumDiscStates  = 1;           % 一个离散状态
sizes.NumOutputs     = 1;           % 一个输出
sizes.NumInputs      = 1;              % 一个输入
sizes.DirFeedthrough = 0;                % 没有直接馈入
sizes.NumSampleTimes = 2;              % 两个采样时间

sys = simsizes(sizes);
x0  = ones(2,1);                      % 初值均设置为1
str = [];
ts  = [0 0;                           % 采样时间是[0 0]表示连续系统
       dperiod doffset];              % 离散采样时间及偏移量，主程序开始时已设置

function sys=mdlDerivatives(t,x,u)
sys = u;                              % 连续系统是积分环节

function sys=mdlUpdate(t,x,u,dperiod,doffset)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);                         % 离散系统是单位延迟
else
  sys = [];
```

```
end

function sys=mdlOutputs(t,x,u,doffset,dperiod)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(2);                          % 输出采样延迟
else
  sys = [];
end
```

10.4 编写 C MEX S－函数

10.3 节介绍的 M 文件 S－函数由于具有易于编写和理解的特点，在仿真计算中得到了广泛的应用。但是它有一些缺点：首先，M 文件 S－函数使得每个仿真步都必须激活 MATLAB 解释器，以致仿真速度变慢；其次，当需要利用 RTW 从 Simulink 框图生成实时代码时，框图中不能含有 M 文件 S－函数。而 C MEX S－函数不仅运算速度快，而且可以用来生成独立的仿真程序。现已的 C 语言编写的程序还可以方便地通过包装程序结合至 C MEX S－函数中。C MEX S－函数结合了 C 语言的优势，可以实现对操作系统和硬件的访问，实现与串口或网络的通信，编写设备驱动程序等。

本节介绍 C MEX S－函数的概念及编程使用方法。

10.4.1 MEX 文件

M 文件 S－函数在 MATLAB 环境下可以通过解释器直接执行，而 C 文件或其它语言编写的 S－函数，则需要先编译成可以在 MATLAB 内运行的二进制代码：动态连接库或静态连接库，然后才可以使用，这些经过编译的二进制文件就称作 MEX 文件。在 Windows 系统下 MEX 文件后缀是 dll。要将 C 文件 S－函数编译成动态连接库，需 MATLAB 命令窗口键入：

```
>> mex my_sfunction.c      % my_sfunction.c 是用户自己编写的 C 文件 S－函数文件名
```

要使用 mex 命令，需要先在系统中安装 C 编译器。如果系统中还没有设置编译器，则要在命令窗口键入：

```
>> mex -setup
```

然后按照提示选取 VC、BC 或其它 C 编译器。建议使用 VC 编译器。生成的文件 my_sfunction.dll 即是所需的动态连接库。当 C 文件中使用了其它库文件时，编译时应该在其后面加上引用的库文件名。如

```
>> mex my_sfunction.c kernel32.lib
```

其实 M 文件 S－函数也可以编译成 MEX 文件再使用，此时 MATLAB 调用的是动态连接库而不是 M 文件本身。M 文件 S－函数编译成 MEX 文件实际上是先用特定的命令将其转换成 C 文件 S－函数，再调用 C 编译器编译链接的。这样编译的代码一般比 M 文件的 S－函数执行速度快，且其代码是二进制的，具有保密性。要将 M 文件编译成 MEX 文件，可在 MATLAB 命令窗口键入：

```
>> mcc  -S mfilename
```

其中，－S 选项表示编译的是 S－函数。在 Windows 系统下生成文件 mfilename.c、mfilename.h、mfilename_simulink.c 和 mfilename.dll。在这个过程中，mcc 自动调用了 mex 命令将生成的 C 代码编译成动态连接库。

10.4.2 C MEX S－函数模板

与 M 文件 S－函数类似，Simulink 也为用户提供了编写 C MEX S－函数的模板。通常使用模板文件 sfuntmpl_basic.c，它为用户提供了 C MEX S－函数的框架结构。该文件包含了常用的几个例程，这对于一般的应用已经足够了。文件 sfuntmpl_doc.c 则包含了所有的例程，并附有详细的说明。

每个 C MEX S－函数的开头应包含下列语句：

```
#define S_FUNCTION_NAME  your_sfunction_name_here
```

```
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
```

其中 your_sfunction_name_here 是用户要编写的 S－函数的名字，即 S－function 模块要输入的 S－函数名。S－函数的格式随之 Simulink 版本的不同而略有不同，这里 S_FUNCTION_LEVEL 说明了该 S－函数适用的 Simulink 版本。头文件 simstruc.h 定义了 S－函数的一个重要数据结构。S－函数的有关信息都保存在此数据结构中。另外，头文件 simstruc.h 还包含着其它重要的头文件，如 tmwtypes.h，头文件 tmwtypes.h 定义了各种数据类型。

另外，和编写普通的 C 文件相同，在文件的顶部还应包含适当的头文件或定义其它的宏或者变量。在它的尾部必然包含下面代码：

```
#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

宏 MATLAB_MEX_FILE 用于告诉编译器该 S－函数正被编译成 MEX 文件。这组代码的含义是如果该文件正被编译成 MEX 文件（Windows 下为 dll 文件），包含 simulink.c；如果正在使用 RTW 将整个 Simulink 框图编译成实时的独立程序（Windows 下为 exe 文件），包含头文件 cg_sfun.h。模板中其它的代码是几个和 M 文件 S－函数中功能类似的 S－函数例程。和 M 文件 S－函数不同，Simulink 不是通过显式的 flag 参数来指定调用 C MEX S－函数例程的。这是因为 Simulink 在交互时会自动地在合适的时候调用每个 S－函数的例程。下面分别介绍 C MEX S 函数在不同仿真阶段的例程。

一、初始化

C MEX S－函数的初始化部分包括下面三个不同的例程函数：

1、mdlInitializeSizes：在该函数中给出各种数量信息；

2、mdlInitializeSampleTimes：在该函数给出采样时间；

3、mdlInitializeConditions：在该函数给出初始状态。

mdlInitializeSizes 通过宏函数对状态、输入、输出等进行设置。工作向量的维数也是在此例程中设置。C MEX S－函数还需要在此例程声明期望的输入参数的个数。表 10.2 列出了初始化所用到的部分宏函数。初始化工作实际上都是通过宏函数访问 SimStruct 数据结构的。

表 10.2 S－函数初始化所用的宏函数

宏 函 数 定 义	功 能 描 述	
ssSetNumContStatus(S, numContStates)	设置连续状态个数	
ssSetNumDiscStatus(S, numDiscStates)	设置离散状态个数	
ssSetNumOutputs(S, numOutputs)	设置输出个数	
ssSetNumInputs(S, numInputs)	设置输入个数	
ssSetDirectFeedthrough (S,dirFeedThrou)	设置是否存在直接前馈	
ssSetNum SampleTimes (S, numSampleTimes)	设置采样时间数目	
ssSetNumInputArgs(S, numInputArgs)	设置输入参数个数	
ssSetNumIWork (S, numIWork)	设置整数型工作向量维数	实际上是为各个工作向量分配内存提供依据。
ssSetNumRWork (S, numRWork)	设置实数型工作向量维数	
ssSetNumPWork (S, numPWork)	设置指针型工作向量维数	

二、使用输入和输出

在 C MEX S－函数中，对输入输出的操作也是通过描述该 S－函数的 SimStruct 进行的。比如，在 C MEX S－函数中如果需要对输入进行操作，需要先使用宏函数：

```
input=ssGetInputPortRealSignalPtrs(S,index)
```

该宏函数的返回中含有指向输入向量的指针，其中每个元素通过*input[i]来访问。同样，如果需要对输出进行操作，需要先使用下列宏函数得到指向输出的指针：

output=ssGetOutputPortRealSignal (S,index)

有关输入与输出相关的宏函数见表 10.3。

表 10.3 有关输入与输出的宏函数

宏函数	功能描述
ssGetInputPortRealSignalPtrs	获得指向输入的指针（double 型）
ssGetInputPortSignalPtrs	获得指向输入的指针（其它数据类型）
ssGetInputPorWidth	获得输入信号的宽度
ssGetInputPortOffsetTime	获得输入端口的采样时间偏移量
ssGetInputPortSampleTime	获得输入端口的采样时间
ssGetOutputPortRealSignal	获得指向输出的指针
ssGettOutputPorWidth	获得输出信号的宽度
ssGetOutputPortOffsetTime	获得输出端口的采样时间偏移量
ssGetOutputPortSampleTime	获得输出端口的采样时间

三、使用参数

使用用户自定义参数时，在初始化时，必须说明参数的个数。为了得到指向存储参数的数据结构的指针，必须使用宏函数 ptr=ssGetSFcnParam(S,index)；为了得到存储在此数据结构中的指向参数值本身的指针，需要使用宏：mxGetPr(ptr)；使用参数值时要使用宏：param_value=* mxGetPr(ptr).

四、使用状态

若 S 函数中包含连续或离散状态，则需要编写 mdlDerivatives 或 mdlUpdate 子函数。使用宏：ssGetRealDiscStates(S)可以得到指向离散状态向量的指针；使用宏函数：ssGetContStates(S) 可以得到指向离散状态向量的指针。在 mdlDerivatives 函数中，连续状态的导数由状态和输入量计算而得，可以通过宏*dx=ssGetX(S)将 SimStruct 结构体中的连续状态导数指针指向得到的结果，然后修改所指向的值。在多状态情况下，需要通过索引得到 dx 中的每个元素。它们被返回给求解器，求解器再积分求出状态。需要指出的是，在离散系统中，没有对应于 dx 的变量，由于状态是由 S－函数更新的，不需要求解器再做工作。

10.4.3 C MEX S 函数应用举例

这里给出几个 C S 函数的示例，以帮助用户更快地掌握 C S 函数的编程方法。

例 10.6 S－函数 csfunc 描述了一个用状态方程表示的线性连续系统：

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

下面是其 C MEX S－函数的完整源代码和注释。

```
#define S_FUNCTION_NAME csfunc1              /*  S-函数名称  */
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])           /* 宏定义，方便对输入的索引 */
/* 定义状态方程中的参数  A B C D 阵  */
static real_T A[2][2]={ { -0.09, -0.01 } ,
                        {  1   ,  0    }   /* 在 C S 函数中有一套自己的数据*/
                      };                   /* 类型表示方法，real_T 表示双精 */
static real_T B[2][2]={ {  1   , -7    } ,
                        {  0   , -2    }
```

```
                        };
static real_T C[2][2]={ {  0   , 2     } ,
                         {  1   , -5    }
                        };
static real_T D[2][2]={ { -3   , 0     } ,          /* 存在直接馈入*/
                         {  1   , 0     }
                        };
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);                /* 不含用户参数，设置为零 */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* 若参数不匹配，Simulink 将发出警告 */
    }
    ssSetNumContStates(S, 2);                /* 系统有两个连续状态*/
    ssSetNumDiscStates(S, 0);               /* 系统无离散状态*/
    if (!ssSetNumInputPorts(S, 1)) return;  /* 如果输入端口数不为 1，则返回*/
    /* S-functions 块只有一个输入端口，当需要多个输入时，必须使用 mux 模块把需要输入的信号合
成一个向量*/
    ssSetInputPortWidth(S, 0, 2);   /* 输入信号宽度为 2 */
    ssSetInputPortDirectFeedThrough(S, 0, 1);   /* 设置馈通标志为 1 */
    if (!ssSetNumOutputPorts(S, 1)) return;  /* 如果输出端口数不为 1，则返回*/
    ssSetOutputPortWidth(S, 0, 2);   /* 输出信号宽度为 2 */
    ssSetNumSampleTimes(S, 1);    /* 1 个采样时间 */
    ssSetNumRWork(S, 0);           /* 不使用工作向量 */
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    /*连续系统的采样时间设置为 0，等同于 ssSetSampleTime(S, 0, 0)   */
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetContStates(S);   /* 获得指向连续状态的指针*/
    int_T  lp;
    for (lp=0;lp<2;lp++) {
        *x0++=0.0;                       /* 各状态初值设置为 0  */
    }
```

```
}
static void mdlOutputs(SimStruct *S, int_T tid)
{   /*获得指向输出向量、连续状态向量和输入端口的指针  */
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    real_T            *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
     UNUSED_ARG(tid); /* not used in single tasking mode */
    /* y=Cx+Du               输出方程 */
    y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
    y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}
  #define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
{
    real_T            *dx   = ssGetdX(S);  /* 获得指向状态导数向量的指针  */
    real_T            *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* xdot=Ax+Bu   状态方程  */
    dx[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*U(1);
    dx[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*U(1);
}
static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S); /* unused input argument */
}
#ifdef  MATLAB_MEX_FILE    /* 是否编译成 MEX 文件 ? */
#include "simulink.c"      /* 包含 MEX 文件的接口机制 */
#else
#include "cg_sfun.h"       /* 代码生成注册函数  */
#endif
```

例 10.7　由 C MEX S 函数实现对下列 van der pole 方程的求解。

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -m(x_1^2 - 1)x_2 - x_1 \end{cases}$$

并要求 m 值和状态初值 x_0 由用户输入。

解 建立如图 10.10 的系统仿真方程。Van der pole 方程由 C MEX S 函数实现，其 C 程序代码如下：

```
#define S_FUNCTION_NAME  van_der2
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define M ssGetSFcnParam(S,0)
#define X00 ssGetSFcnParam(S,1)
static void mdlInitializeSizes(SimStruct *S)     /* 得到指向存储参数的数据结构的指针  */
```

```
{
    ssSetNumSFcnParams(S, 2);              /* 两个用户参数 */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 2);              /* 两个连续状态 */
    ssSetNumDiscStates(S, 0);
    if (!ssSetNumInputPorts(S, 0)) return;     /* 系统无输入端口  */
    if (!ssSetNumOutputPorts(S, 1)) return;
       ssSetOutputPortWidth(S, 0, 2);        /* 输出信号为 2 维  */
    ssSetNumSampleTimes(S, 1);             /*一个采样时间 */
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetContStates(S);
 const real_T    *x00=mxGetPr(X00);     /* 得到存储在数据结构中的指向参数值本身的指针
x0[0] =x00[0];
    x0[1] = x00[1];
}
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    real_T            *x    = ssGetContStates(S);
    y[0]=x[0];
    y[1]=x[1];
    }
#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
{
    real_T *dx   = ssGetdX(S);
    real_T *x    = ssGetContStates(S);
          const real_T      m = *mxGetPr(M);
```

```
    dx[0] = -m*x[0] * (1.0 - x[1] * x[1]) - x[1];
    dx[1] = x[0];
}
static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S); /* unused input argument */
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

由例 10.6 和 10.7 给出的程序可以看出，C MEX S－函数是通过一套宏函数获得指向存储在 SimStruct 中的输入、输出、状态、状态导数及用户参数的指针来引用这些变量，从而完成对系统的描述。

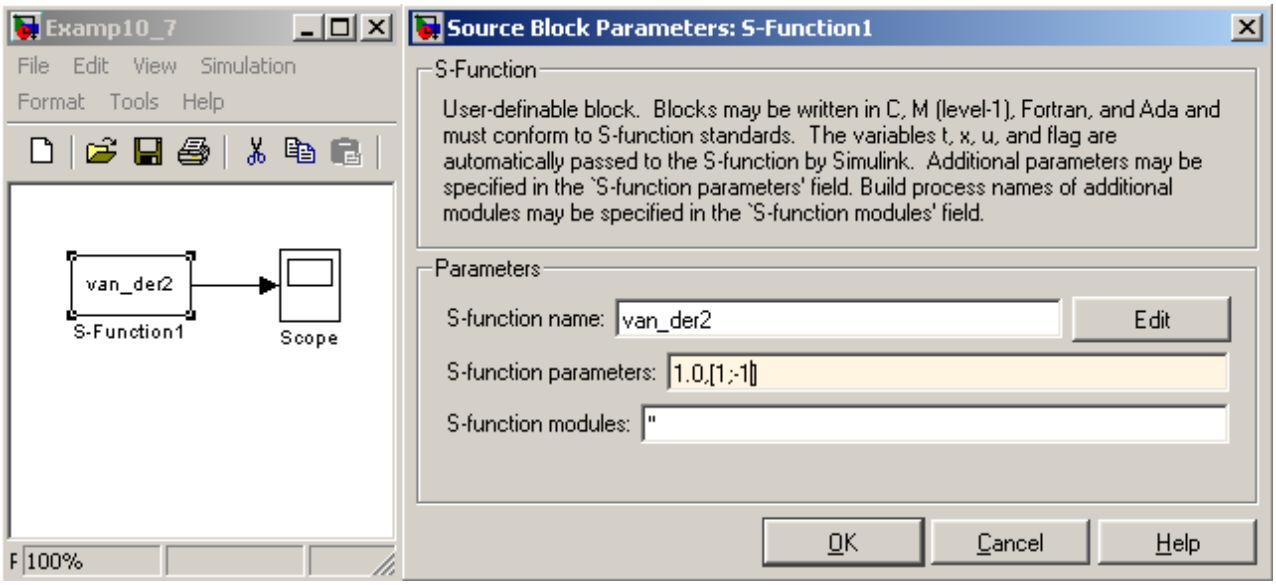


图 10.10　例 10.7 系统仿真模型及 S－函数参数对话框

10.4.4　S－function Builder

Simlink 为用户编写常用的 C MEX S－函数提供了一个非常方便的开发工具－－S－function Builder。S－function Builder 可以使读者不需了解任何宏函数就可以编写出自己的 C MEX S－函数。用户只需在 S－function Builder 界面中的相应位置写入所需的信息和代码即可。S－function Builder 会自动生成 C MEX S－函数源文件。用户只要单击 Build 按钮，S－function Builder 就会自动编译，自动生成用户所需的 MEX 文件。

下面通过例子介绍 S－function Builder 的应用。

例 10.8　利用 S－function Builder 实现对 Van der pole 方程的求解。

解　从 Simulink 游览器中将 S－function Builder 图标拖至新建的模型文件中。

双击 S－function Builder 图标，即可打开如图 10.11（a）所示的 S－function Builder 界面。用户可以看出 1、4、5、6 选项卡对应着 S－函数的四个常用例程。用户只需在相应的例程函数中填写所需的信息和代码即可。下面给出使用 S－function Builder 编写 S－函数的方法。

在 S－function name 编辑栏中填写 S－函数名称；

在图 10.11（a）所示的初始化（Initialization）选项卡中按照提示设置仿真的相关信息。本例所设置的信息见图 10.11（a）所示；

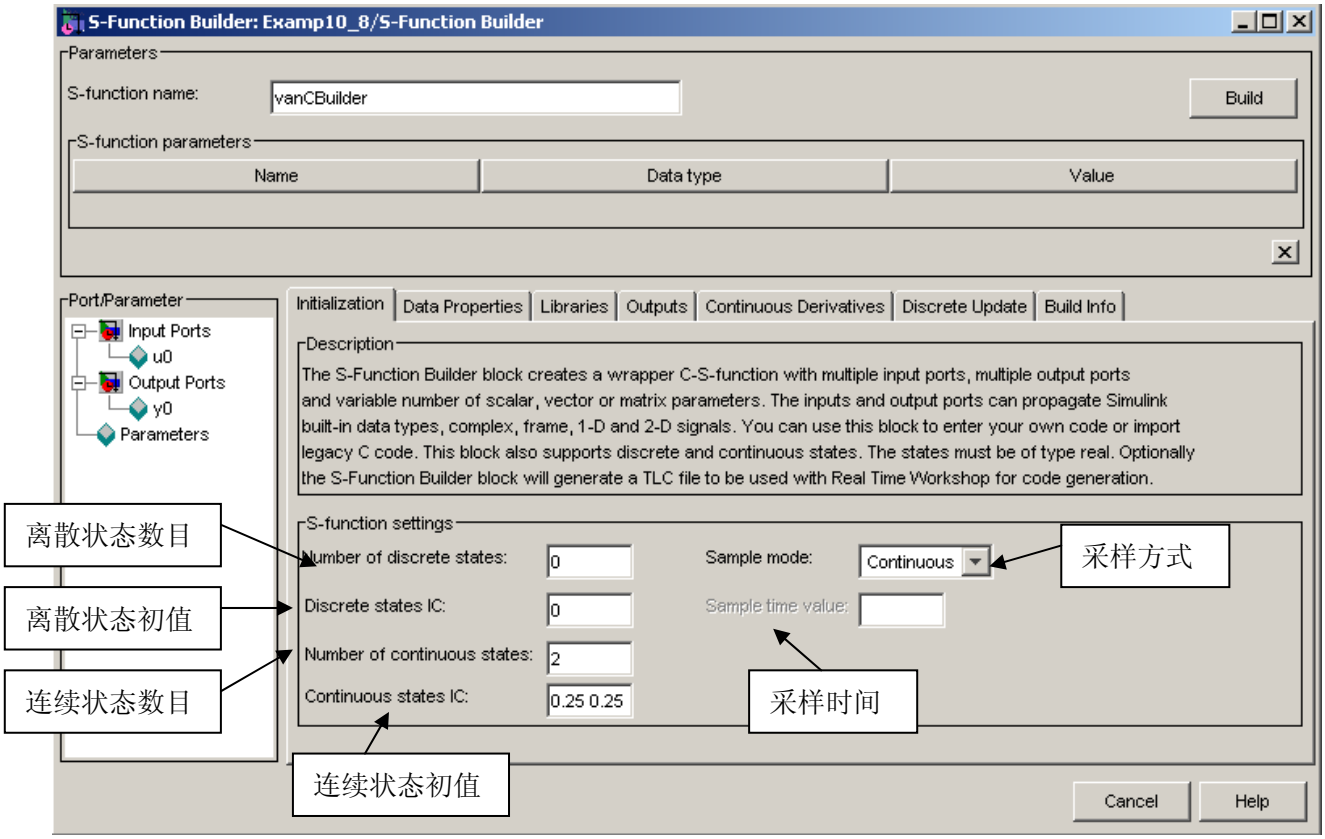
在图 10.11（b）所示的数据属性选项卡中设置输入、输出的数据类型、信号维数等信息，设置用户参数的名称、数据类型等。本例设置见图 10.11（b）；

在 Library 选项卡中需要填入所需的库文件（包括目录）、要包含的头文件以及外部函数声明等；

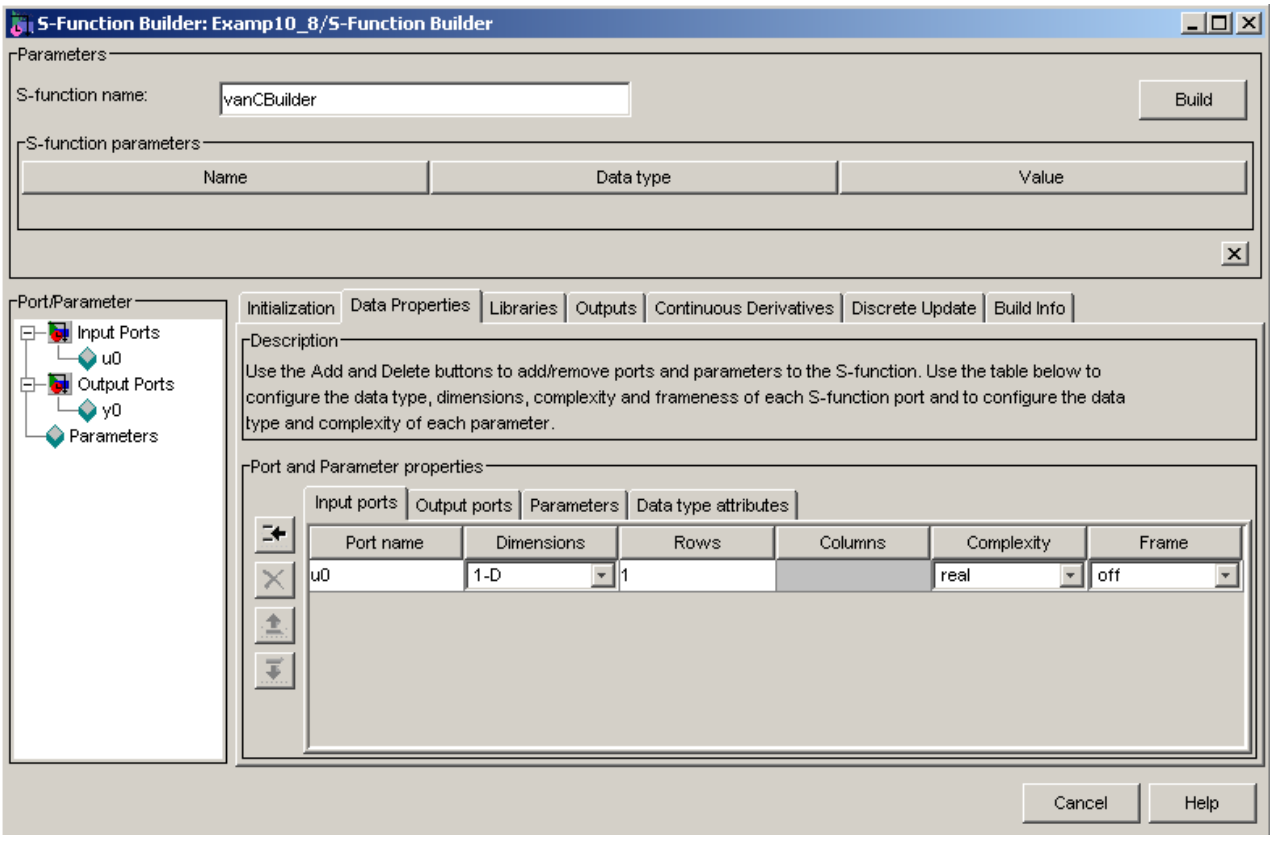
在 Outputs、Continues Derivatives 和 Discrete Update 选项卡中分别填入输出方程、连续状态方程、离散状态方程以及其它用户定制的代码。本例是连续系统，其 Outputs、Continues Derivatives 选项卡所需填写的代码见图 10.11（c）、（d）；

单击 Build 按钮，S－function Builder 会自动生成 C 代码、编译链接等工作，并给出编辑链接成功信息。

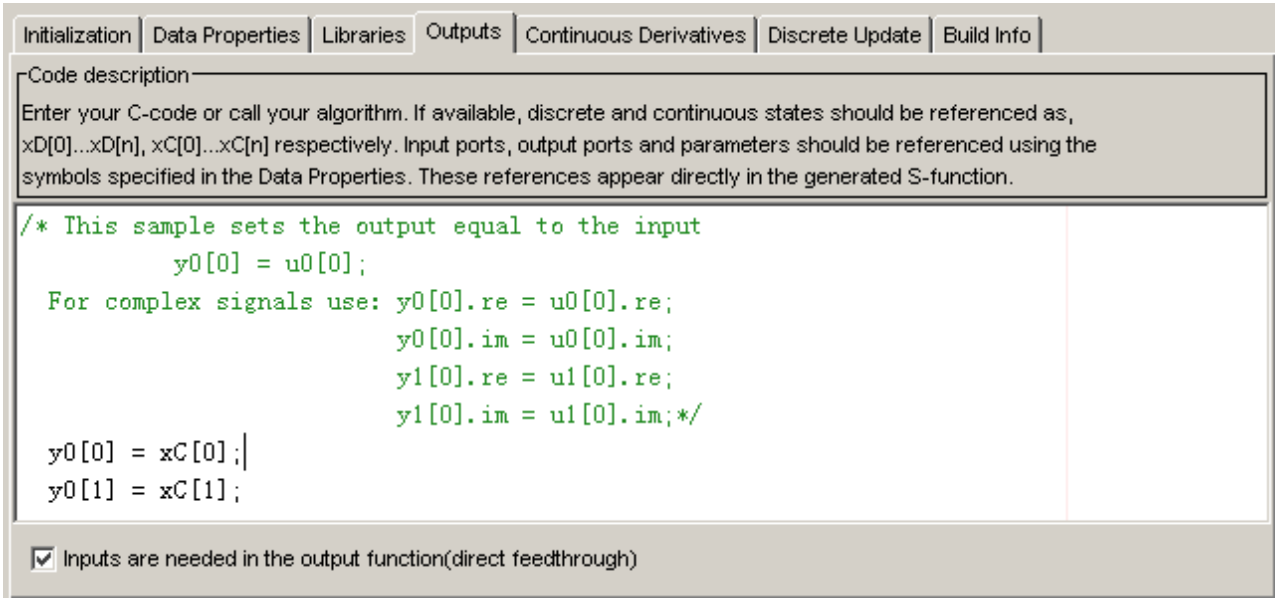
例 10.8 的仿真模型及仿真结果见图 10.11(e)。



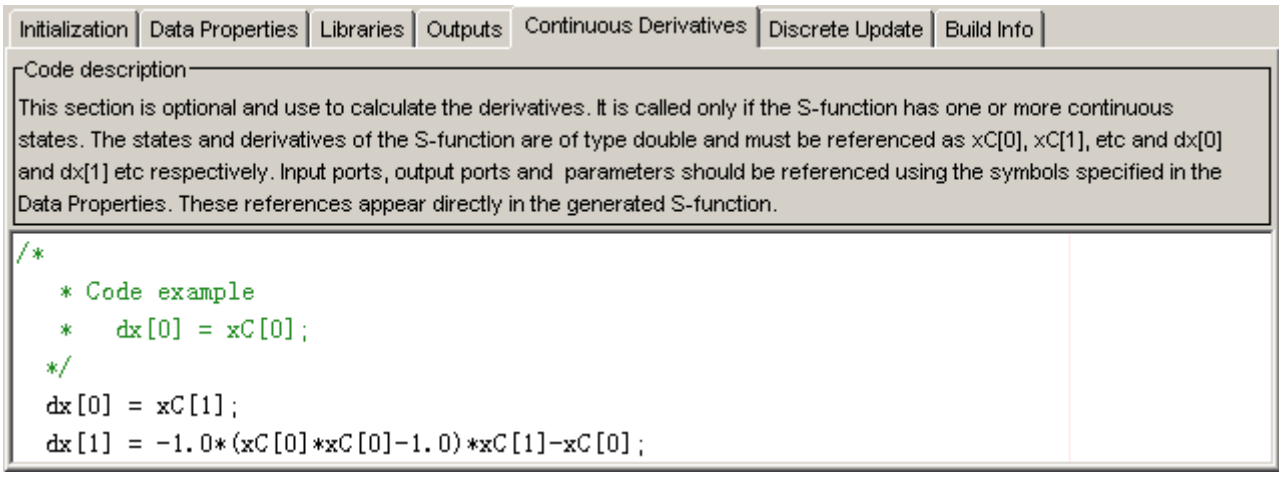
（a） S－function Builder 初始化界面



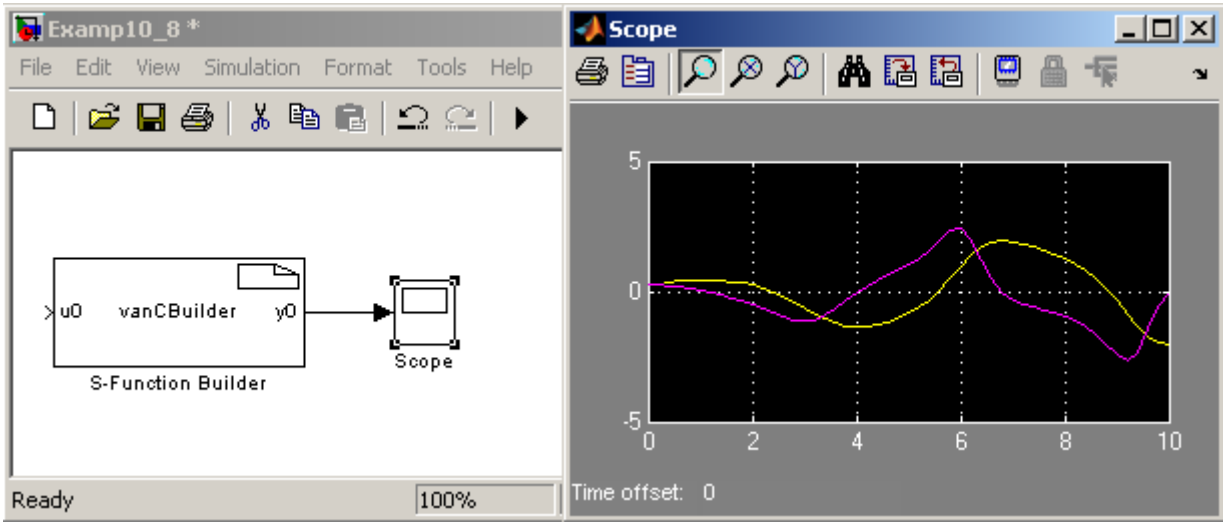
（b）S－function Builder 数据属性设置界面



（c）S－function Builder 输出选项卡



（d）S－function Builder 连续状态导数计算选项卡



（e） 例 10.8 系统仿真模型及结果

图 10.11　例 10.8 的 S－function Builder 设置及其仿真结果

习 题

10.1 已知图 10.12 所示为倒立摆系统。摆杆长度$l=1m$，摆杆质量$m=0.1kg$。摆杆底端用铰链安装在质量为$M=1kg$的小车上，在水平方向上施加控制力u，相对参考系产生位移z，忽略各种摩擦，设重力加速度为$g=9.81m/s^2$。试编写 M 文件 S 函数和 C MEX S 函数，建立倒立摆系统的模型并进行仿真。要求用户可以输入系统参数（摆杆长度、摆杆质量、小车质量）

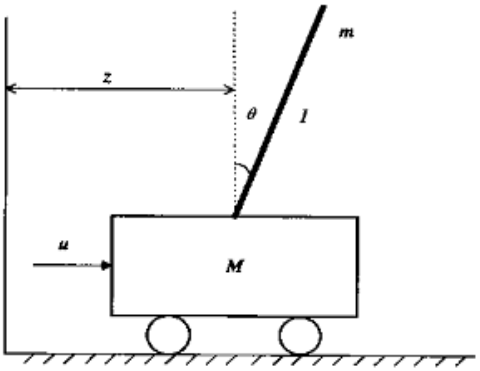


图 10.12 题 10.1 图

提示：建立数学模型：小车位置z，立摆偏离垂直位置的角度是θ，见图 10.12。

根据力学方程，其运动可用如下方程描述

$$\begin{cases} \ddot{z}=\dfrac{(u+ml\dot{\theta}^2\sin\theta)\cos\theta-(g\sin\theta+l\dot{\theta}^2\sin\theta\cos\theta)m}{M\cos\theta} \\ \ddot{\theta}=\dfrac{(M+m)(g\sin\theta+l\dot{\theta}^2\sin\theta\cos\theta)-\cos\theta(u+ml\sin\theta)}{Ml\cos^2\theta} \end{cases}$$