第5章 基于Simulink的S-Function建模

S-Function是一个Simulink模块。S-Function中的输出值是状态、输入和时间的函数。S-Function是Simulink的重要组成部分，Simulink为编写S-Function提供了各种模板文件，其中定义了S-Function完整的框架结构，用户可以根据自己的需要加以剪裁。本章主要引导用户从最简单的S-Function编写出发，逐步掌握S-Function进行控制系统设计。

学习目标：

(1) 熟练掌握MATLAB S-Function编写；

(2) 熟练掌握S-Function用于控制系统建模；

(3) 熟练掌握S-Function书写Simulink模块等。

5.1 Simulink S-Function仿真应用

S-Function(system function)是Simulink模块的计算机语言描述，可以用M、C/C++、Ada、Fortran语言以MEX文件的形式编写。

S-Function以特殊的方式与Simulink方程求解器交互。这种交互和Simulink内建模块的做法非常相似。S-Function模块可以是连续、离散或者混合系统。

通过S-Function，用户可以将自己的模块加入Simulink模型中，从而可以实现用户自定义的算法或者利用操作系统、硬件设备交互。

5.1.1 Simulink S-Function仿真过程

Simulink模型的执行按下述几个步骤。

首先是初始化阶段。在这一阶段Simulink将库模块集合到模型，传播宽度、数据类型和采样时间，评估模块参数，确定模块执行顺序，分配内存。然后是仿真阶段。此时Simulink进入一个仿真循环，循环的每次执行对应一个仿真步。

在每个仿真步，Simulink按初始化阶段确定的顺序执行各个模块。

对每个模块，Simulink 计算模块在当前采样时间的状态、微分和输出。这将持续到仿真结束。

5.1.2 S-Function 的回调方法

S-Function 包括一系列的回调(Callback)方法，用于执行每个仿真步骤所需的任务。在一个模型的仿真过程中，每个仿真步骤，Simulink 将调用各 S-Function 的适当方法。S-Function 执行的方法包括：

(1) 初始化。在首次仿真循环中执行。Simulink 初始化 S-Function。在这一步骤中 Simulink 将：

① 初始化 SimStruct，这是一种 Simulink 结构，包含了 S-Function 的信息。

② 设置输入/输出端口的个数和纬度。

③ 设置模块的采样次数。

④ 分配存储区域和数组长度。

(2) 计算下一采样点。如果定义了一个可变采样步长的模块，这一步将计算下一次采样点，也就是计算下一步长。

(3) 计算在主要时间步中的输出。这一步结束之后，模块的输出端口在当前时间步是有效的。

(4) 更新主要时间步中的离散状态。所有的模块在该回调方法中，必须执行一次每次时间步都要执行的活动，如为下一次仿真循环更新离散状态。

(5) 积分。这用于具有连续状态的或者(和)具有非采样过零的模型。如果用户的 S-Function 具有连续状态，Simulink 在最小采样步长调用 S-Function 的输出和微分部分。这也是 Simulink 之所以能计算 S-Function 的状态。如果用户 S-Function(仅针对 C MEX)具有非采样过零，Simulink 在最小采样步长调用 S-Function 的输出和微分部分，这样可以确定过零点。

5.2 M-File S-Functions 应用

用 M 语言编写的 S-Function 称为 M-File S-Functions，根据 API 版本不同，分为 Level-2 和 Level-1 类型。

Level-1 的 M-File S-Function 支持采用 M 语言实现具有全部功能的 Simulink 模块。

Level-2 的 M-File S-Function APIs 非常接近于 C MEX-File S-Functions，许多特性可以相同使用。

在 MATLAB 主界面中直接输入 sfundemos，即可调出 S-Function 的许多编程例子。

```
>> sfundemos
```

弹出如图 5-1 所示的菜单。

单击图 5-1 中的 MATLAB file S-Functions，弹出 MATLAB file S-Function Level-2

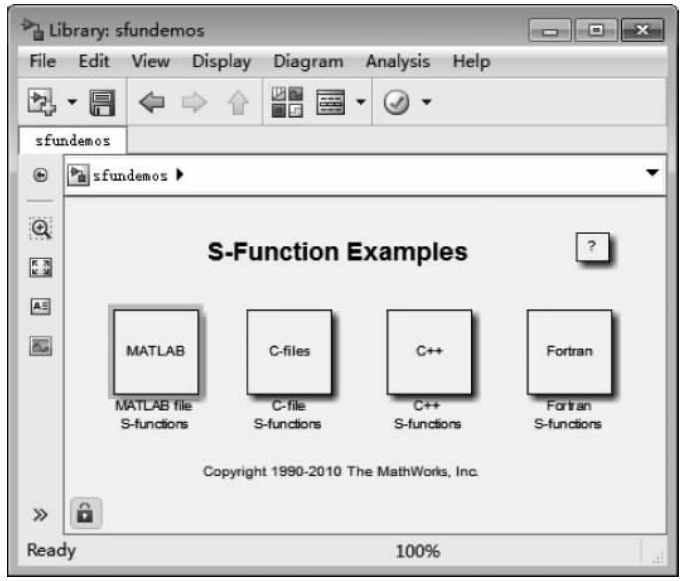

图 5-1　S-Function 例子

和 Level-1 两种类型，如图 5-2 所示。

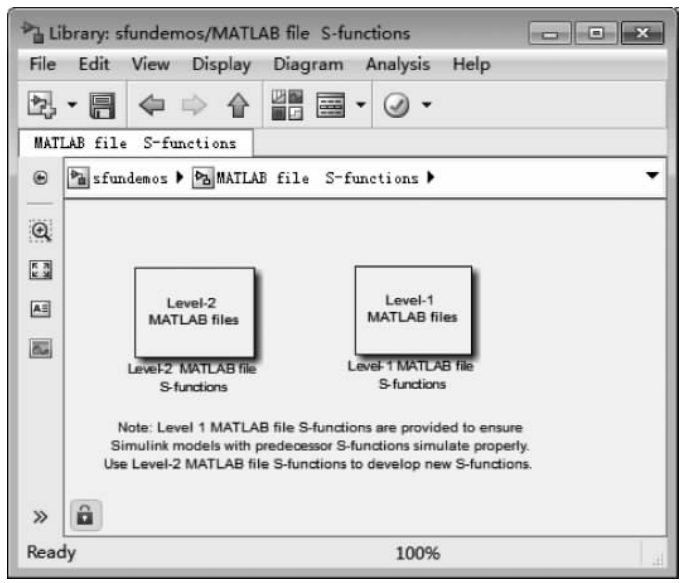

图 5-2　Level-2 和 Level-1 S 函数

分别单击 Level-2 和 Level-1，弹出相应的案例分析界面，如图 5-3 和图 5-4 所示。

单击 Level-1 中的 Continuous Time Variable Step MATLAB file 模块，弹出图 5-5 所示的仿真文件模型。

对图 5-5 所示仿真模型进行仿真操作，得到图 5-6 所示的图形。

单击 Continuous Time Variable Step by S-Function 模块，弹出图 5-7 所示的界面。

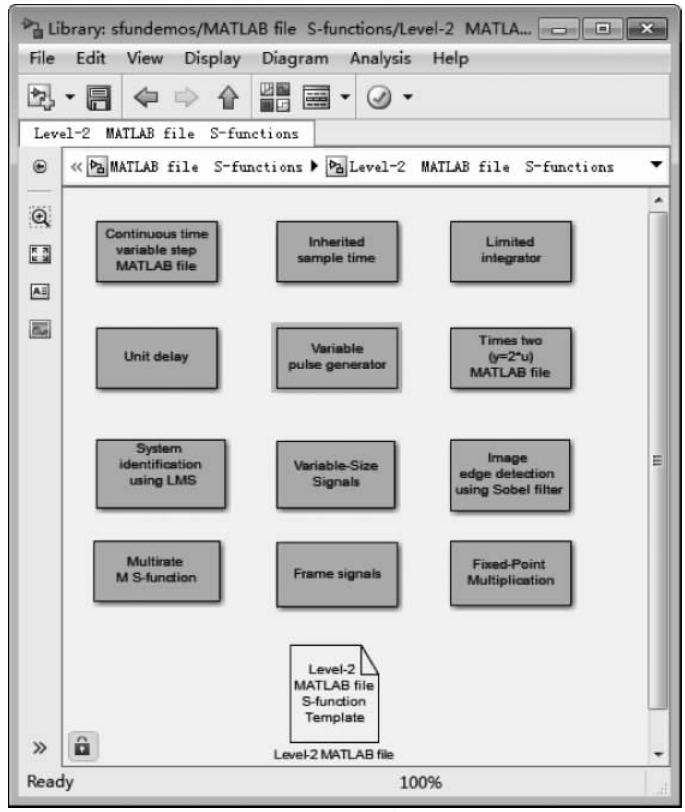

图 5-3　Level-2 例程

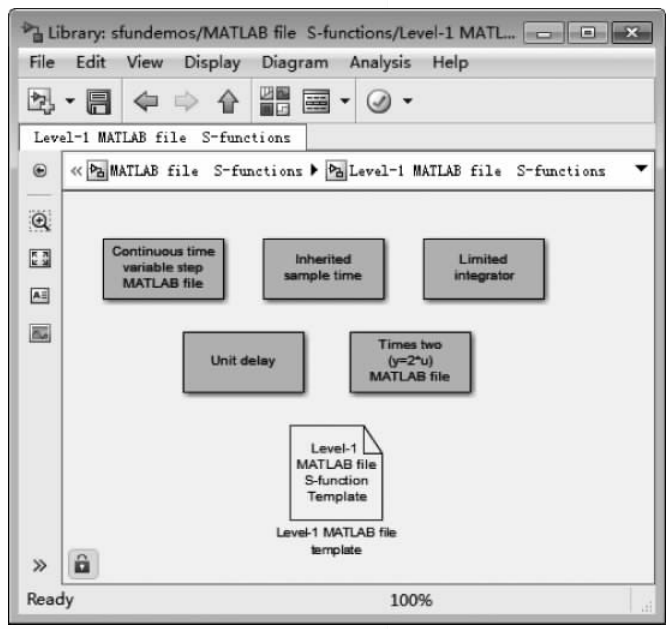

图 5-4　Level-1 例程

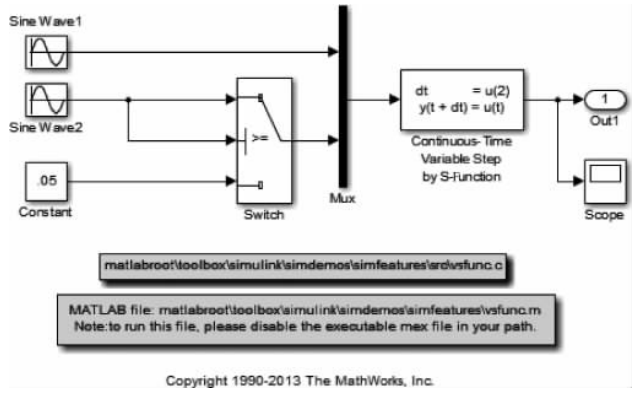

图 5-5　仿真模型

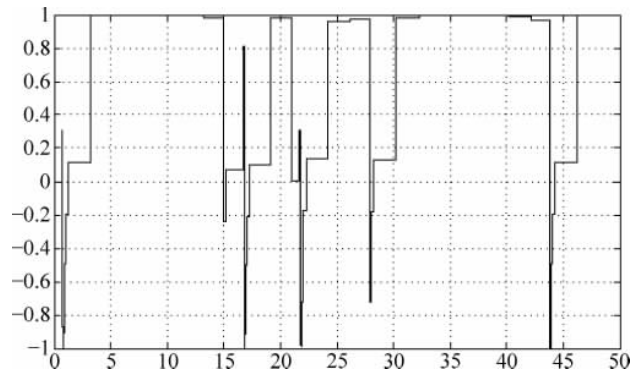

图 5-6　示波器图形

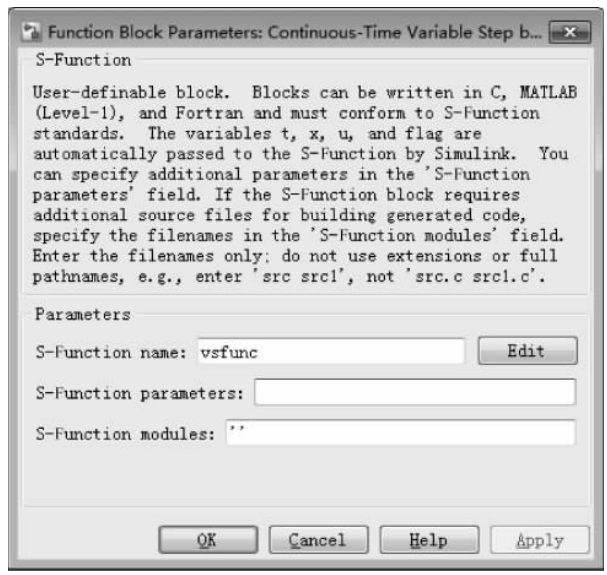

图 5-7　Continuous Time Variable Step by S-Function 模块

单击右侧的 Edit 按钮，弹出用户用 S-Function 书写的函数文件。具体的程序如下：

```
/*   File    : vsfunc.c
 *   Abstract:
*       Example C-file S-Function for defining a continuous system.
*       Variable step S-Function example.
 *       This example S-Function illustrates how to create a variable step
 *       block in Simulink. This block implements a variable step delay
 *       in which the first input is delayed by an amount of time determined
 *       by the second input:
*       dt       = u(2)
 *       y(t+dt) = u(t)
*   Copyright 1990-2013 The MathWorks, Inc.
 */
#define S_FUNCTION_NAME vsfunc
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])       /* Pointer to Input Port0 */
/* Function: mdlInitializeSizes ===========================================
 * Abstract:
 *     The sizes information is used by Simulink to determine the S-function
 *     block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);                /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return;                     /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 1);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);
    if (ssGetSimMode(S) == SS_SIMMODE_RTWGEN && !ssIsVariableStepSolver(S)) {
        ssSetErrorStatus(S, "S-function vsfunc.c cannot be used with Simulink Coder "
                         "and Fixed-Step Solvers because it contains variable"
                         " sample time");
    }
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
```

```
/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *     Variable - Step S - function
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, VARIABLE_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}
#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions ========================================
 * Abstract:
 *     Initialize discrete state to zero.
 */
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetRealDiscStates(S);
    x0[0] = 0.0;
}
#define MDL_GET_TIME_OF_NEXT_VAR_HIT
static void mdlGetTimeOfNextVarHit(SimStruct *S)
{
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    /* Make sure input will increase time */
    if (U(1) <= 0.0) {
        /* If not, abort simulation */
        ssSetErrorStatus(S,"Variable step control input must be "
                           "greater than zero");
        return;
    }
    ssSetTNext(S, ssGetT(S) + U(1));
}
/* Function: mdlOutputs =======================================================
 * Abstract:
 *      y = x
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T *y = ssGetOutputPortRealSignal(S,0);
    real_T *x = ssGetRealDiscStates(S);

    /* Return the current state as the output */
    y[0] = x[0];
}
#define MDL_UPDATE
/* Function: mdlUpdate ======================================================
 * Abstract:
 *     This function is called once for every major integration time step.
```

```
 *     Discrete states are typically updated here, but this function is useful
 *     for performing any tasks that should only take place once per integration
 *     step.
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    real_T             *x     = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    x[0] = U(0);
}
/* Function: mdlTerminate =====================================================
 * Abstract:
 *     No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}
#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
```

对比 Level-1 MATLAB Files，单击 Level-2 MATLAB Files，仍选择 Continuous Time Variable Step by S-Function 模块，弹出仿真模型如图 5-8 所示。运行仿真文件得到相应的输出图形，如图 5-9 所示。

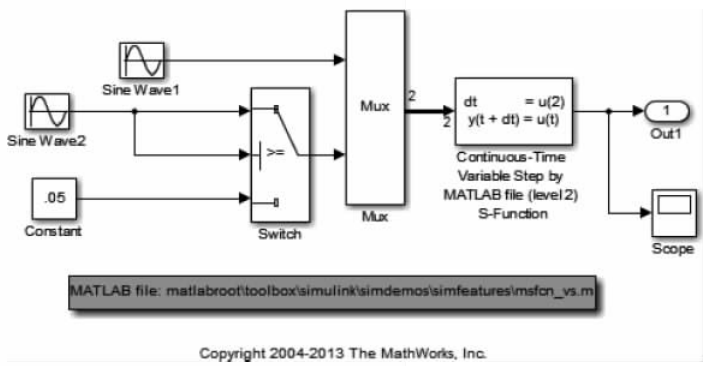

图 5-8　仿真模型

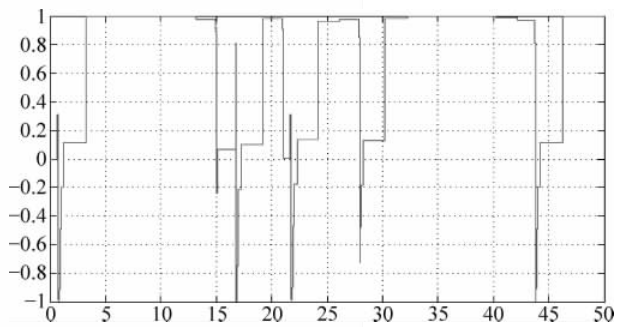

图 5-9　输出图形

编辑 msfcn_vs S-function 文件，弹出 S-Function 函数文件。具体如下：

```
function msfcn_vs(block)
% Level - 2 MATLAB file S - Function for continuous time variable step demo.
%   Copyright 1990 - 2009 The MathWorks, Inc.
 setup(block);
% endfunction
function setup(block)
 % % Register number of input and output ports
 block.NumInputPorts   = 1;
 block.NumOutputPorts = 1;
 % % Setup functional port properties to dynamically
 % % inherited.
 block.SetPreCompInpPortInfoToDynamic;
 block.SetPreCompOutPortInfoToDynamic;
 block.InputPort(1).Dimensions         = 2;
 block.InputPort(1).DirectFeedthrough = true;
 block.OutputPort(1).Dimensions        = 1;
 % % Set block sample time to variable sample time
 block.SampleTimes = [ - 2 0];
 % % Set the block simStateCompliance to default (i.e., same as a built - in block)
 block.SimStateCompliance = 'DefaultSimState';
 % % Register methods
 block.RegBlockMethod('PostPropagationSetup',     @DoPostPropSetup);
 block.RegBlockMethod('InitializeConditions',     @InitConditions);
 block.RegBlockMethod('Outputs',                  @Output);
 block.RegBlockMethod('Update',                   @Update);
% endfunction
function DoPostPropSetup(block)
 % % Setup Dwork
 block.NumDworks = 1;
 block.Dwork(1).Name = 'X';
 block.Dwork(1).Dimensions       = 1;
 block.Dwork(1).DatatypeID       = 0;
 block.Dwork(1).Complexity       = 'Real';
 block.Dwork(1).UsedAsDiscState = true;
% endfunction
function InitConditions(block)
 % % Initialize Dwork
 block.Dwork(1).Data = 0;
% endfunction
function Output(block)
 block.OutputPort(1).Data = block.Dwork(1).Data;
 % % Set the next hit for this block
 block.NextTimeHit = block.CurrentTime + block.InputPort(1).Data(2);
% endfunction
function Update(block)
 block.Dwork(1).Data = block.InputPort(1).Data(1);
% endfunction
```

对比 Level-1 MATLAB Files 和 Level-2 MATLAB Files，Level-2 MATLAB Files 更加简捷，且结构较清晰，模型采用块输入的方式，使得编程更加简单，因此 Level-2 MATLAB Files 将逐渐取代 Level-1 MATLAB Files，在较高版本的 Simulink S-Function 中，系统逐渐强化 Level-2 MATLAB Files 应用，而相应地弱化 Level-1 MATLAB Files 应用。

5.3 M-File S-Function 模板

在 MATLAB 主界面中直接输入：

```
edit sfuntmpl
```

即可弹出 S-Function 模板编辑的 M-File 环境，用户在里面修改即可。具体如下：

```
function [sys,x0,str,ts,simStateCompliance] = sfuntmpl(t,x,u,flag)
% SFUNTMPL General MATLAB S-Function Template
%   With MATLAB S-Functions, you can define you own ordinary differential
%   equations (ODEs), discrete system equations, and/or just about
%   any type of algorithm to be used within a Simulink block diagram.
%   The general form of an MATLAB S-Function syntax is:
%       [SYS,X0,STR,TS,SIMSTATECOMPLIANCE] = SFUNC(T,X,U,FLAG,P1,…,Pn)
%   What is returned by SFUNC at a given point in time, T, depends on the
%   value of the FLAG, the current state vector, X, and the current
%   input vector, U.
%   FLAG   RESULT             DESCRIPTION
%   -----   ------              ----------------------------------------
%   0      [SIZES,X0,STR,TS]      Initialization, return system sizes in SYS,
%                                 initial state in X0, state ordering strings
%                                 in STR, and sample times in TS.
%   1      DX                     Return continuous state derivatives in SYS.
%   2      DS                     Update discrete states SYS = X(n+1)
%   3      Y                      Return outputs in SYS.
%   4      TNEXT                  Return next time hit for variable step sample
%                                 time in SYS.
%   5                             Reserved for future (root finding).
%   9      []                     Termination, perform any cleanup SYS=[].
%   The state vectors, X and X0 consists of continuous states followed
%   by discrete states.
%   Optional parameters, P1,…,Pn can be provided to the S-Function and
%   used during any FLAG operation.
%   When SFUNC is called with FLAG = 0, the following information
%   should be returned:
%      SYS(1) = Number of continuous states.
%      SYS(2) = Number of discrete states.
%      SYS(3) = Number of outputs.
%      SYS(4) = Number of inputs.
%               Any of the first four elements in SYS can be specified
```

```
%                as -1 indicating that they are dynamically sized. The
%                actual length for all other flags will be equal to the
%                length of the input, U.
%       SYS(5) = Reserved for root finding. Must be zero.
%       SYS(6) = Direct feedthrough flag (1=yes, 0=no). The s-Function
%                has direct feedthrough if U is used during the FLAG=3
%                call. Setting this to 0 is akin to making a promise that
%                U will not be used during FLAG=3. If you break the promise
%                then unpredictable results will occur.
%       SYS(7) = Number of sample times. This is the number of rows in TS.
%       X0     = Initial state conditions or [] if no states.
%       STR    = State ordering strings which is generally specified as [].
%       TS     = An m-by-2 matrix containing the sample time
%                (period, offset) information. Where m = number of sample
%                times. The ordering of the sample times must be:
%                TS = [0      0,       : Continuous sample time.
%                      0      1,       : Continuous, but fixed in minor step
%                                        sample time.
%                      PERIOD OFFSET,  : Discrete sample time where
%                                        PERIOD > 0 & OFFSET < PERIOD.
%                      -2     0];      : Variable step discrete sample time
%                                        where FLAG=4 is used to get time of
%                                        next hit.
%                There can be more than one sample time providing
%                they are ordered such that they are monotonically
%                increasing. Only the needed sample times should be
%                specified in TS. When specifying more than one
%                sample time, you must check for sample hits explicitly by
%                seeing if
%                   abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
%                is within a specified tolerance, generally 1e-8. This
%                tolerance is dependent upon your model's sampling times
%                and simulation time.
%                You can also specify that the sample time of the S-Function
%                is inherited from the driving block. For functions which
%                change during minor steps, this is done by
%                specifying SYS(7) = 1 and TS = [-1 0]. For functions which
%                are held during minor steps, this is done by specifying
%                SYS(7) = 1 and TS = [-1 1].
%       SIMSTATECOMPLIANCE = Specifices how to handle this block when saving and
%                            restoring the complete simulation state of the
%                            model. The allowed values are: 'DefaultSimState',
%                            'HasNoSimState' or 'DisallowSimState'. If this value
%                            is not speficified, then the block's compliance with
%                            simState feature is set to 'UknownSimState'.
%   Copyright 1990-2010 The MathWorks, Inc.
% The following outlines the general structure of an S-Function.
switch flag,
 % % % % % % % % % % % % % % % % % % %
 % Initialization %
```

```
 % % % % % % % % % % % % % % % % % %
 case 0,
    [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes;
 % % % % % % % % % % % % % % % % % %
 % Derivatives %
 % % % % % % % % % % % % % % % % % %
 case 1,
    sys = mdlDerivatives(t,x,u);
 % % % % % % % % % %
 % Update %
 % % % % % % % % % %
 case 2,
    sys = mdlUpdate(t,x,u);
 % % % % % % % % % %
 % Outputs %
 % % % % % % % % % %
 case 3,
    sys = mdlOutputs(t,x,u);
 % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
 % GetTimeOfNextVarHit %
 % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
 case 4,
    sys = mdlGetTimeOfNextVarHit(t,x,u);
 % % % % % % % % % % % % %
 % Terminate %
 % % % % % % % % % % % % %
 case 9,
    sys = mdlTerminate(t,x,u);
 % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
 % Unexpected flags %
 % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
 otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
end
% end sfuntmpl

% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S - Function.
function [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S - Function parameters.
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 0;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
```

```
sizes.NumSampleTimes = 1;          % at least one sample time is needed
sys = simsizes(sizes);
% initialize the initial conditions
x0  = [];
% str is always an empty matrix
str = [];
% initialize the array of sample times
ts  = [0 0];
% Specify the block simStateCompliance. The allowed values are:
%    'UnknownSimState', < The default setting; warn and assume DefaultSimState
%    'DefaultSimState', < Same sim state as a built-in block
%    'HasNoSimState',   < No sim state
%    'DisallowSimState' < Error out when saving or restoring the model sim state
simStateCompliance = 'UnknownSimState';

% end mdlInitializeSizes

% mdlDerivatives
% Return the derivatives for the continuous states.
function sys = mdlDerivatives(t,x,u)
sys = [];
% end mdlDerivatives

% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
function sys = mdlUpdate(t,x,u)
sys = [];
% end mdlUpdate

% mdlOutputs
% Return the block outputs.
function sys = mdlOutputs(t,x,u)
sys = [];
% end mdlOutputs

% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
function sys = mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;          %   Example, set the next hit to be one second later.
sys = t + sampleTime;
% end mdlGetTimeOfNextVarHit

% mdlTerminate
% Perform any end of simulation tasks.
function sys = mdlTerminate(t,x,u)
sys = [];
% end mdlTerminate
```

由 S-Function 程序模块，S-Function 格式如下：

```
[sys,x0,str,ts,simStateCompliance] = sfuntmpl(t,x,u,flag)
```

S-Function 默认的四个输入参数：t，x，u，flag。

S-Function 默认的四个输出函数：sys，x0，str，ts。

各个参数的含义如下：

t：代表当前的仿真时间，该输入决定了下一个采样时间。

x：表示状态向量，行向量，引用格式为 x(1)，x(2)。

u：表示输入向量。

flag：控制在每一个仿真阶段调用哪一个子函数的参数，由 Simulink 在调用时自动取值。

sys：通用的返回变量，返回的数值决定 flag 值。

(1) mdlUpdates 里，列向量，引用格式为 sys(1,1)，sys(2,1)。

(2) mdlOutputs 里，行向量，引用格式为 sys =x. 。

x0：初始的状态值；列向量，引用格式为 x0=[0；0；0]。

str：空矩阵，无具体含义。

ts：包含模块采样时间和偏差的矩阵，[period，offset]。

当 ts 为－1 时，表示与输入信号同采样周期。

S-Function 具体包括函数名称和功能如表 5-1 所示。

表 5-1　S-Function 子函数

S-Function 仿真	仿 真 阶 段
mdlInitialization	初始化
mdlGetTimeofNextVarHit	计算下一个采样点
mdlOutput	计算输出
mdlUpdate	更新离散状态
mdlDerivatives	计算导数
mdlTerminate	结束仿真

5.3.1　S-Function 工作方式

S-Function 工作方式如下：

```
switch flag,
 case 0,
    [sys,x0,str,ts,simStateCompliance] = 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
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
end
```

其中，flag = 0 时，调用 mdlInitializeSizes 函数，定义 S-Function 的基本特性，包括采样时间、连续或者离散状态的初始条件和 Sizes 数组；flag = 1 时，调用 mdlDerivatives 函数，计算连续状态变量的微分方程；求所给表达式的等号左边状态变量的积分值的过程；flag = 2 时，调用 mdlUpdate 函数，用于更新离散状态，采样时间和主时间步的要求；flag = 3 时，调用 mdlOutputs 函数，计算 S-Function 的输出；flag = 4 时，调用 mdlGetTimeOfNextVarHit 函数，计算下一个采样点的绝对时间，这个方法仅仅是使用户在 mdlInitializeSize 里说明一个可变的离散采样时间；flag = 9 时，调用 mdlTerminate 函数，实现仿真任务的结束。

5.3.2 S-Function 仿真过程

S-Function 仿真过程中主要函数如下：

```
function [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates       = 0;
sizes.NumDiscStates       = 0;
sizes.NumOutputs          = 0;
sizes.NumInputs           = 0;
sizes.DirFeedthrough      = 1;
sizes.NumSampleTimes      = 1;       % at least one sample time is needed
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
simStateCompliance = 'UnknownSimState';
function sys = mdlDerivatives(t,x,u)
sys = [];
function sys = mdlUpdate(t,x,u)
sys = [];
function sys = mdlOutputs(t,x,u)
sys = [];
function sys = mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;                     % Example, set the next hit to be one second later.
sys = t + sampleTime;
function sys = mdlTerminate(t,x,u)
sys = [];
```

由函数调用顺序得 S-Function 仿真步骤如下：

(1) 初始化。mdlInitializeSizes，初始化 S-Function，即 simsizes。

(2) 初始化 SimStruct，包含了 S-Function 的所有信息，主要设置如下：

① 设置输入 u、输出端口 sys；

② 设置采样时间 ts；

③ 分配存储空间 str。

(3) 数值积分：mdlDerivatives，用于连续状态的求解和非采样过零点。分如下两种情况：

① 如果存在连续状态，调用 mdlDerivatives 和 mdlOutput 两个子函数。

② 如果存在非采样过零点，调用 mdlOutput 和 mdlZeroCrossings 子函数，以定位过零点。

(4) 更新离散状态：mdlUpdate。

(5) 计算输出：mdlOutputs，计算所有输出端口的输出值。

(6) 计算下一个采样时间点：mdlGetTimeOfNextVarHit。

(7) 仿真结束：mdlTerminate，在仿真结束时调用。

5.3.3 S-Function 的编写

(1) 参数初始设定。初始化 sizes 结构，再调用 simsizes 函数。

Sizes 结构体在程序中体现如下：

```
sizes = simsizes;
sizes.NumContStates         = 0;
sizes.NumDiscStates         = 0;
sizes.NumOutputs            = 0;
sizes.NumInputs             = 0;
sizes.DirFeedthrough        = 1;
sizes.NumSampleTimes        = 1;       % at least one sample time is needed
sys = simsizes(sizes);
```

其中，NumContStates 为连续状态的个数；NumDiscStates 为离散状态的个数；NumOutputs 为输出变量的个数；NumInputs 为输入变量的个数；DirFeedthrough 表示有无直接馈入，值为 1 时表示输入直接传到输出口；NumSampleTimes 表示采样时间的个数，值为 1 时表示只有一个采样周期。

simsizes 函数的调用：sys = simsizes (sizes)，即将 sizes 结构体中的信息传递给 sys。

(2) 连续模块的状态更新由 mdlDerivatives 函数来进行。

(3) 离散模块的状态更新由 mdlUpdate 函数来进行。

(4) 输出信号的计算。计算出模块的输出信号，系统的输出仍然由 sys 变量返回。

5.3.4 M-File S-Function 的模块化

S-Function 为 Simulink 的“系统”函数，能够响应 Simulink 求解器命令的函数，采用非图形化的方法实现一个动态系统。

在动态系统仿真设计、分析中，用户可以使用 S-Function 模块来调用 S-Function。

(1) S-Function 模块是一个单输入单输出的模块，如果有多个输入与输出信号，可以使用 Mux 模块与 Demux 模块对信号进行组合和分离操作。

(2) 在 S-Function 模块的参数设置对话框中，包含了调用的 S 函数名和用户输入的参数列表，如图 5-10 所示。

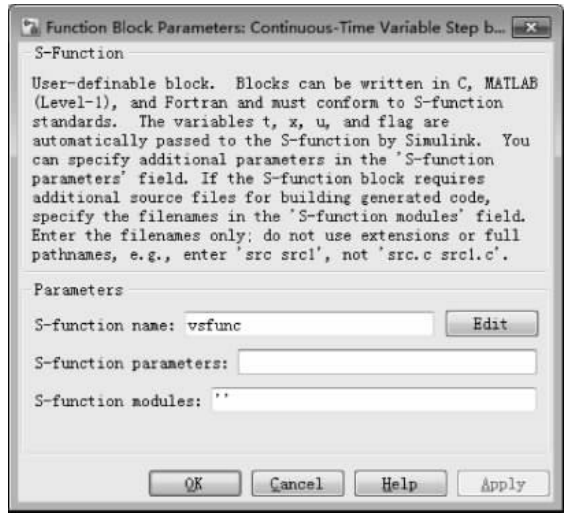

图 5-10 S-Function 模块函数调用

(3) S-Function 模块是以图形的方式提供给用户一个调用 S-Function 的接口，S-Function 中的源文件必须由用户自行编写。

(4) S-Function 模块中的 S-函数名和参数值列表必须与用户填写的 S-Function 源文件的名称和参数列表完全一致，包括参数的顺序。

具体的 M-File S-Function 流程如图 5-11 所示。

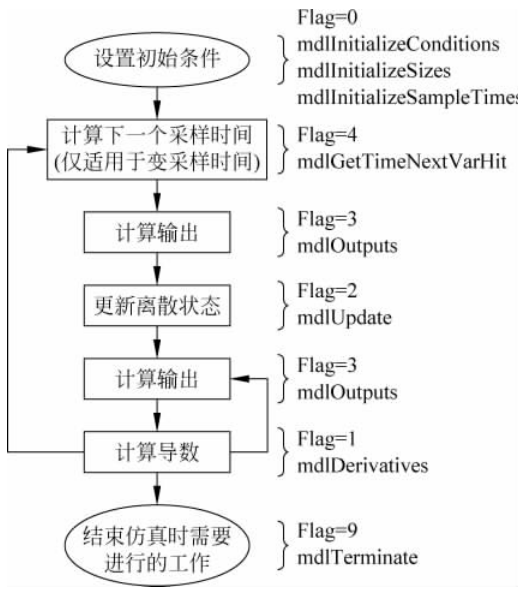

图 5-11 M-File S-Function 流程

5.4 M-File S-Function 实现

【例 5-1】 用 S-Function 实现 gain 模块。

增益值作为 S-Function 用户自定义参数由用户输入。

(1) 对 M-File S-Function 的主函数定义做了修改,增加新的参数,并采用新的函数名:

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

(2) 由于增益参数只是用来计算输出值,因而对 mdlOutputs 的调用可修改成:

```
case 3,
  sys = mdlOutputs(t,x,u);
```

(3) 修改初始化例程:

```
sizes.NumContStates     = 0;
sizes.NumDiscStates     = 0;
sizes.NumOutputs        = 1;
sizes.NumInputs         = 1;
sizes.DirFeedthrough    = 1;
sizes.NumSampleTimes    = 0;          % at least one sample time is needed
sys = simsizes(sizes);
```

(4) mdlOutputs 子函数的定义也做了相应的修改,将增益作为参数输入:

```
function sys = mdlOutputs(t,x,u)
sys = 2 * u;
```

输出通过增益和输入的乘积得到,并通过 sys 返回。

完整的 S-Function 如下:

```
function [sys,x0,str,ts,simStateCompliance] = sfun_ysw(t,x,u,flag)
switch flag,
 % Initialization %
 case 0,
    [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes;
 % Derivatives %
 case 1,
    sys = mdlDerivatives(t,x,u);
 % Update %
 case 2,
    sys = mdlUpdate(t,x,u);
 % Outputs %
```

```
 case 3,
    sys = mdlOutputs(t,x,u);
 % GetTimeOfNextVarHit %
 case 4,
    sys = mdlGetTimeOfNextVarHit(t,x,u);
 % Terminate %
 case 9,
    sys = mdlTerminate(t,x,u);
 % Unexpected flags %
 otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
end

% mdlInitializeSizes
function [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates     = 0;
sizes.NumDiscStates     = 0;
sizes.NumOutputs        = 1;
sizes.NumInputs         = 1;
sizes.DirFeedthrough    = 1;
sizes.NumSampleTimes    = 0;          % at least one sample time is needed
sys = simsizes(sizes);
% initialize the initial conditions
x0  = [];
% str is always an empty matrix
str = [];
% initialize the array of sample times
ts  = [];
simStateCompliance = 'UnknownSimState';

% mdlDerivatives
function sys = mdlDerivatives(t,x,u)
sys = [];

% mdlUpdate
function sys = mdlUpdate(t,x,u)
sys = [];

% mdlOutputs
function sys = mdlOutputs(t,x,u)
sys = 2*u;

% mdlGetTimeOfNextVarHit
function sys = mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;          % Example, set the next hit to be one second later.
sys = t + sampleTime;

% mdlTerminate
function sys = mdlTerminate(t,x,u)
sys = [];
```

搭建仿真模型，如图 5-12 所示。运行仿真模型得到如图 5-13 所示的图形。

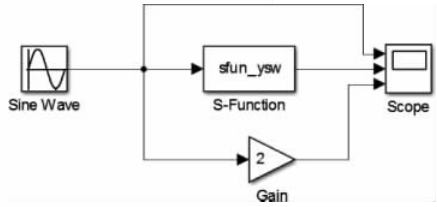

图 5-12　仿真模型

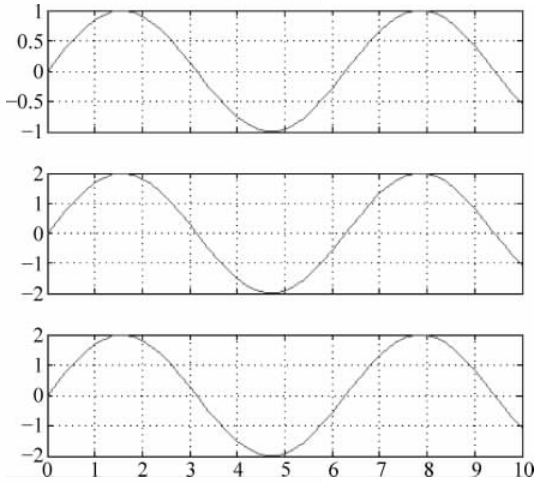

图 5-13　仿真图形

如图 5-13 所示，采用系统增益模块 Gain 与 S-Function 书写程序执行效果一样。

【例 5-2】　用 M-File S-Function 实现一个积分器。

对于一个积分器，输入输出之间的关系：$\dot{y}=x$。系统输出方程为 $Y(S)=X(S)\cdot\frac{1}{x}$。下面修改 S-Function 模板文件。

（1）修改 S-Function 模板的第一行：

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

（2）初始状态应当传递给 mdlInitializeSizes：

```
% Derivatives %
case 1,
    sys = mdlDerivatives(t,x,u);
% Update %
case 2,
    sys = mdlUpdate(t,x,u);
% Outputs %
case 3,
    sys = mdlOutputs(t,x,u);
```

(3) 设置初始化参数：

```
sizes = simsizes;
sizes.NumContStates  = 1;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;       % at least one sample time is needed
sys = simsizes(sizes);
% initialize the initial conditions
x0  = [0];
% str is always an empty matrix
str = [];
% initialize the array of sample times
ts  = [];
simStateCompliance = 'UnknownSimState';
```

(4) 书写状态方程：

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

(5) 添加输出方程：

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

完整的 S-Function 如下：

```
function [sys,x0,str,ts,simStateCompliance] = sfun_ysw_s(t,x,u,flag)
switch flag,
 % Initialization %
 case 0,
    [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes;
 % Derivatives %
 case 1,
    sys = mdlDerivatives(t,x,u);
 % Update %
 case 2,
    sys = mdlUpdate(t,x,u);
 % Outputs %
 case 3,
    sys = mdlOutputs(t,x,u);
 % GetTimeOfNextVarHit %
 case 4,
    sys = mdlGetTimeOfNextVarHit(t,x,u);
 % Terminate %
 case 9,
```

```
    sys = mdlTerminate(t,x,u);
  % Unexpected flags %
  otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
end

% mdlInitializeSizes
function [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates       = 1;
sizes.NumDiscStates       = 0;
sizes.NumOutputs          = 1;
sizes.NumInputs           = 1;
sizes.DirFeedthrough      = 1;
sizes.NumSampleTimes      = 0;          % at least one sample time is needed
sys = simsizes(sizes);
% initialize the initial conditions
x0  = [0];
% str is always an empty matrix
str = [];
% initialize the array of sample times
ts  = [];
simStateCompliance = 'UnknownSimState';

% mdlDerivatives
function sys = mdlDerivatives(t,x,u)
sys = u;

% mdlUpdate
function sys = mdlUpdate(t,x,u)
sys = [];

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

% mdlGetTimeOfNextVarHit
function sys = mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;       % Example, set the next hit to be one second later.
sys = t + sampleTime;

% mdlTerminate
function sys = mdlTerminate(t,x,u)
sys = [];
```

令输入的初始状态为 0，搭建仿真模型，如图 5-14 所示。运行仿真模型得到如图 5-15 所示的图形。

如图 5-15 所示，采用系统积分模块 Integrator 与 S-Function 书写程序执行效果一样。

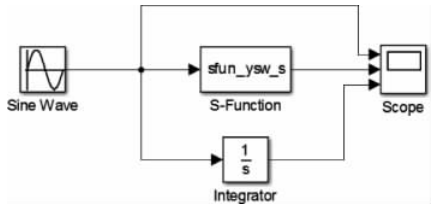

图 5-14 仿真模型

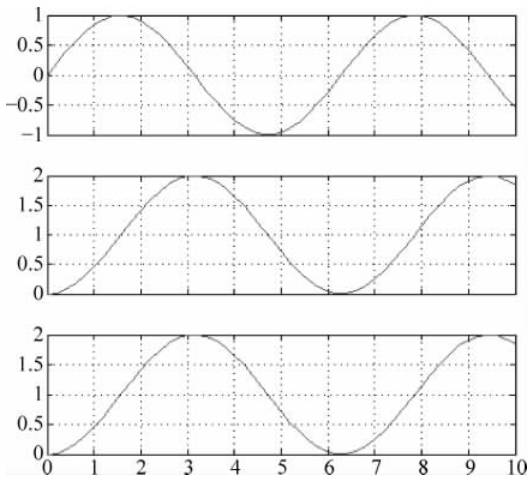

图 5-15 仿真图形

5.5 本章小结

S-Function 为 Simulink 的“系统”函数，能够响应 Simulink 求解器命令的函数，采用非图形化的方法实现一个动态系统。S-Function 可以开发新的 Simulink 模块，可以与已有的代码相结合进行仿真，采用文本方式输入复杂的系统方程，M-File S-Function 可以扩展图形能力，C MEX S-Function 可以提供与操作系统的接口，S-Function 的语法结构是为实现一个动态系统而设计的(默认用法)，其他 S-Function 的用法是默认用法的特例(如用于显示目的)。