This tutorial exposes you to the main ideas you'll need to use *Simulink* in ChemE 480. It reviews the material covered in week 1 of the lab, and introduces creation of masked blocks.

A tutorial example

Consider the heat exchange process shown in Figure 1. Suppose that you can adjust the inlet liquid rate ($\dot{w}_i$) and temperature (T_i), and the steam temperature (T_s) independently. The liquid outlet temperature (T_o) and flow rate ($\dot{w}_o$), and the heat transfer rate ($\dot{q}$) vary accordingly.

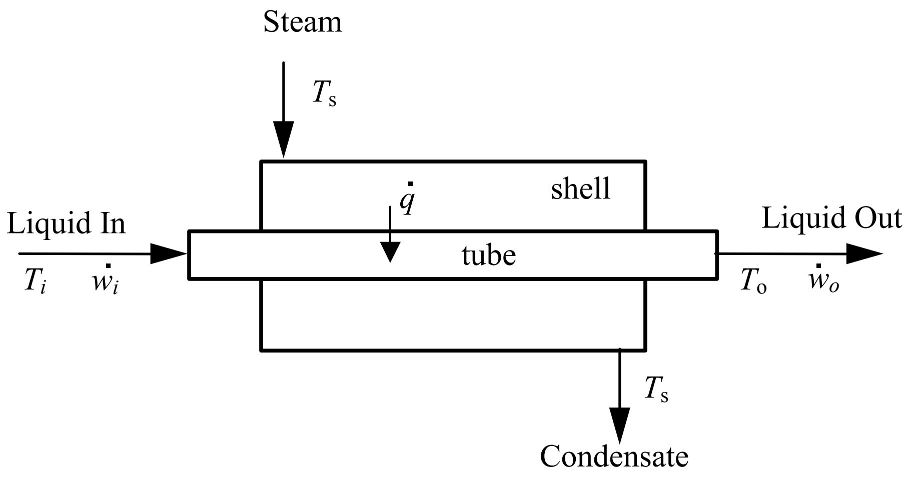


Figure 1 Shell-and-tube heat exchanger schematic

We make the following simplifying assumptions:

1. The inlet steam is saturated, and the condensate leaves as a saturated liquid at the same temperature.
2. $U = 800 \dfrac{\text{W}}{\text{m}^2\,\text{K}}$ $A = 300\ \text{m}^2$, so $UA = 240$ kW/K (constant)
3. Other constants: liquid density, $\rho_L = 800$ kg/m^3; liquid holdup in tubes, $V_L = 2.1$ m^3; liquid heat capacity, C_p= 1.8 kJ/kg-K, all independent of temperature.
4. Energy accumulation in the tube wall material is negligible.
5. Liquid in tubes is <u>well-mixed</u> in the radial and axial dimensions (a poor assumption if the tubes are long) and incompressible.

Using background from previous courses (and a few additional assumptions), you should be able to obtain the following model equations:

$$\dot{w}_i = \dot{w}_o = \dot{w} \tag{1}$$

$$\dot{q} = UA(T_s - T_o) \tag{2}$$

$$wC_p \frac{dT_o}{dt} = \dot{w}C_p(T_i - T_o) + \dot{q} = \dot{w}C_p(T_i - T_o) + UA(T_s - T_o) \tag{3}$$

where

$\dot{w}$	mass rate of liquid entering and leaving the tubes, kg/s
$\dot{q}$	rate of heat transfer to the liquid in the tubes, kW
w	mass of liquid in the tubes ($=\rho_L V_L$), kg
UA	product of overall heat transfer coefficient and tube surface area, kW/K
T_s	steam temperature, ^{o}C
T_i, T_o	temperature of liquid entering and leaving tubes, ^{o}C.
C_p	specific heat of liquid at constant pressure, kJ/kg-K

Given the values of $\dot{w}$, T_i, and T_s as functions of time – and suitable initial conditions – it should be possible to solve equations (2) and (3) for T_o and $\dot{q}$ as functions of time. One way would be to use the MATLAB techniques you used in CHEME 465. The purpose of this tutorial is to illustrate some related methods that are more convenient for process control.

NOTE: All graphics are from a PC running MATLAB Version 6, Release 12. Some window views may change on other configurations.

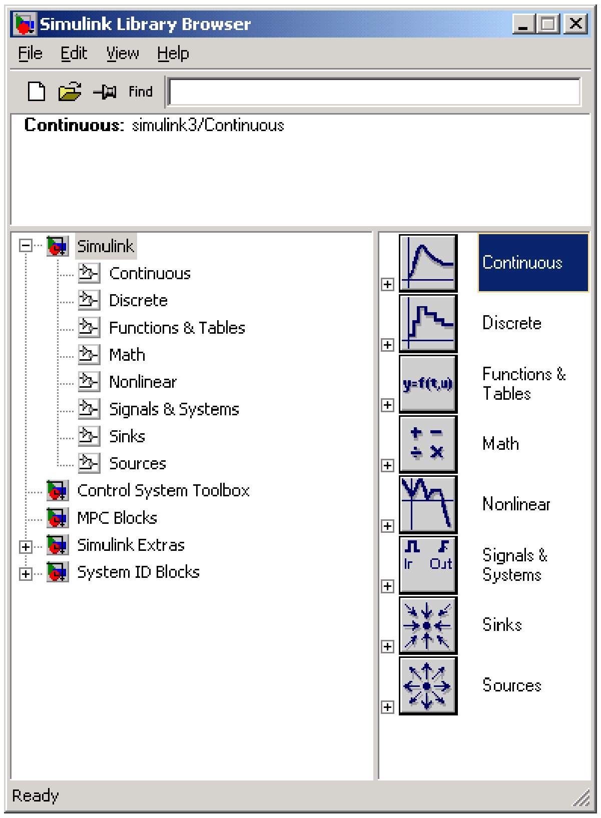


Figure 2 The Simulink Library Browser window has just been opened.

Building a Simulink model

Opening the Library Browser and model windows

With MATLAB running, type `simulink`[1] in the command window. This opens the Simulink Library Browser window (see Figure 2), which provides building blocks for model creation.

[1] I'll use a `Courier` font to represent MATLAB commands that you should type in the MATLAB Command Window.

Use the Library Browser's *File* menu (or the blank document icon) to open a new model window. We will build the model by copying Library blocks into this window.

The idea is to translate each equation into a sequence of blocks, and to hook them together in the appropriate way to form a system. The variables in the equations become *signals* that vary with time. Blocks perform mathematical operations on input signals to generate an output signals.

Modeling an algebraic equation

For example, consider the linear equation, $y = mx + b$, where *m* and *b* are constants. Suppose *m=2*, and *b=3*. Let *x* be an independent function of time. Given *x*, you could easily calculate the dependent variable, *y*, without resorting to a simulation, but we'll use Simulink here to illustrate the concepts.

Figure 3 shows one way to set it up (signal labels in green added for clarity). In this case, *x* is being modeled (arbitrarily) as a *step function* (the *Step* block). The *x* signal enters a *Gain* block, which multiplies its input by a specified constant (*m=2* in this case). Thus, the output of the *Gain* block is *2x*. To this we add the output of a *Constant* block, which is sending a value of *b=3* here. Thus, the signal going into the *Scope* is *y = 2x + 3*.

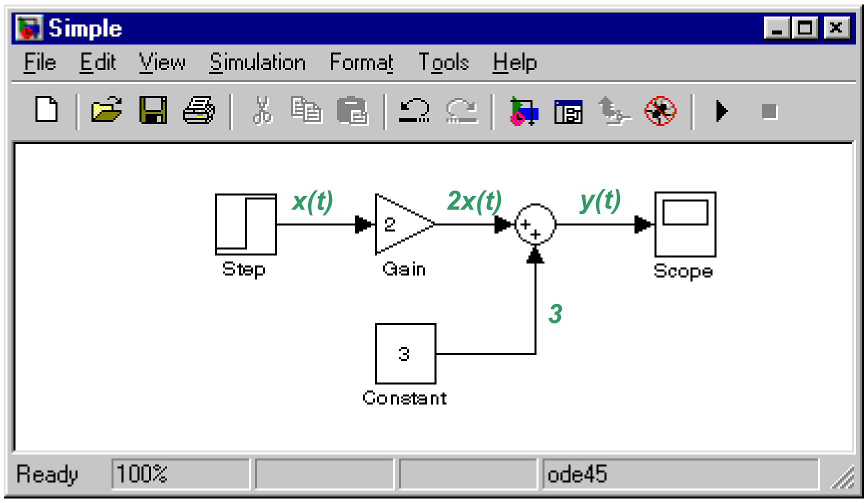


Figure 3 Simulink representation of $y = 2x + 3$, where *x* is a step function.

The *Scope* provides a graphical display of its input as a function of time, and is just one of many things we could do with our *y* signal. (Others include storing it on disk or in the MATLAB Workspace).

Copying blocks into your model

As an exercise, duplicate Figure 3 by dragging the appropriate blocks from the Library Browser to your new model window. To begin, you need to locate the *Step* block. The list in the lower-left Browser panel shows block categories. Click on the *Sources* category to reveal the standard Simulink signal source blocks, which appear in the lower-right panel (see Figure 4). You will probably have to scroll down to find the *Step* block (blocks appear in alpha order). Once you've found it, if your model window is showing you can drag a copy of the *Step* block to it. Alternatively, you can right-click on the *Step* block. The resulting menu gives you the option to paste the block into the model window.

Next, add the remaining blocks to your model window. It can be frustrating to locate them[2]. Here are some hints: a *Constant* is another signal source, the *Gain* and *Sum* are math operations, and the *Scope* is a signal sink.

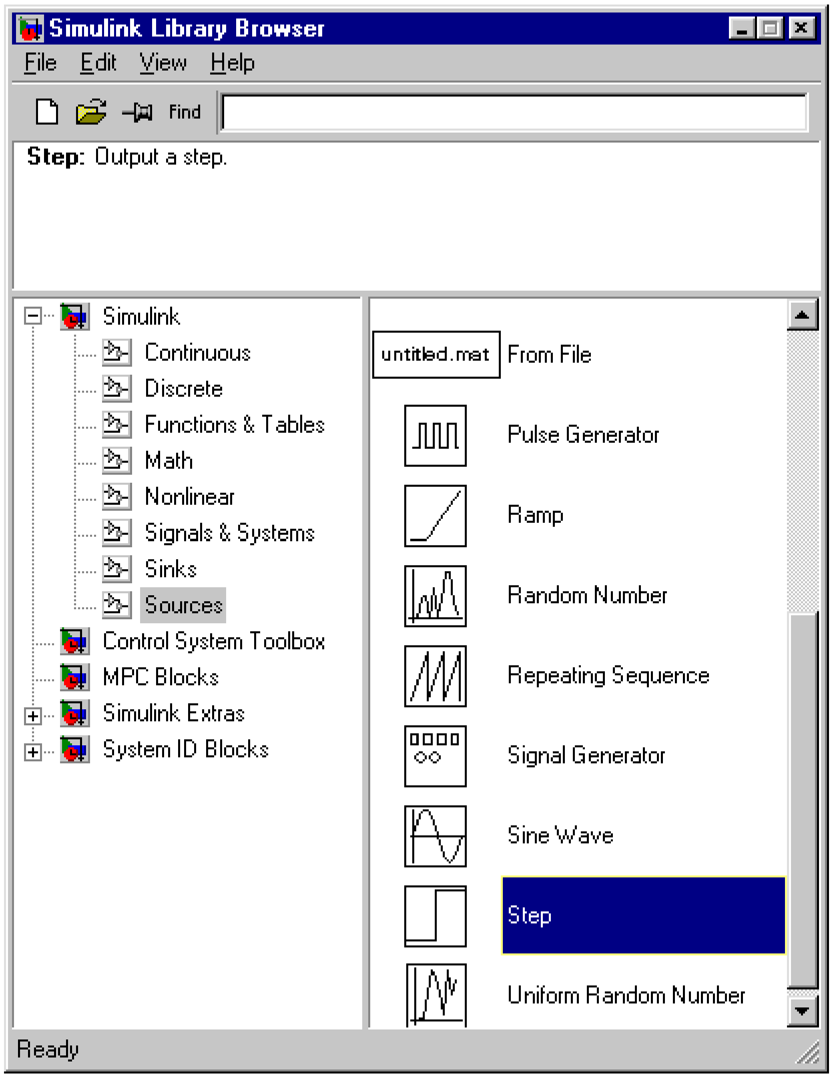


Figure 4 Simulink Library Browser after clicking on *Sources* and selecting *Step*

Note: If you're ever having trouble, type `helpdesk` in the MATLAB Command Window, which opens the Help Browser. Use the Contents panel to navigate to Simulink/Using Simulink/Block Reference/Simulink Block Libraries, which lists each block by name. Clicking on a name brings up a detailed description, including the block's location in the library. Similarly, if you're browsing the Library and are uncertain of a particular library block's purpose, right-click on it and select *help* in the resulting menu to see the detailed description. Also, the Library Browser's upper panel provides a brief description of the selected block (see Figure 4 for the *Step* block's brief description).

[2] Not only is it difficult to anticipate the operations available, but their assignments to the categories can be non-obvious, and they tend to change from one Simulink version to the next. Electrical engineers were heavily involved in Simulink's development, which makes the presentation less intuitive for us.

Entering block parameters

You should now have the five required blocks in your model window, arranged as shown in Figure 5. The *Gain* and *Constant* values are incorrect, however (the defaults are 1). To fix these, double-click on each, and enter the desired value in the parameter box.

Connecting blocks

The small > symbol on the right side of the *Step* block represents its output port. Note the input port on the *Gain* block's left side. Click and drag from the *Step*'s output to the *Gain*'s input to connect them. The resulting signal line *must* have a solid, filled-in arrowhead, as in Figure 3. If not, the blocks aren't connected; try again. Similarly, connect the rest of the blocks until you obtain the result shown earlier in Figure 3. Note the way Simulink automatically makes a neat right-angle connection when you connect the *Constant* to the *Sum*.

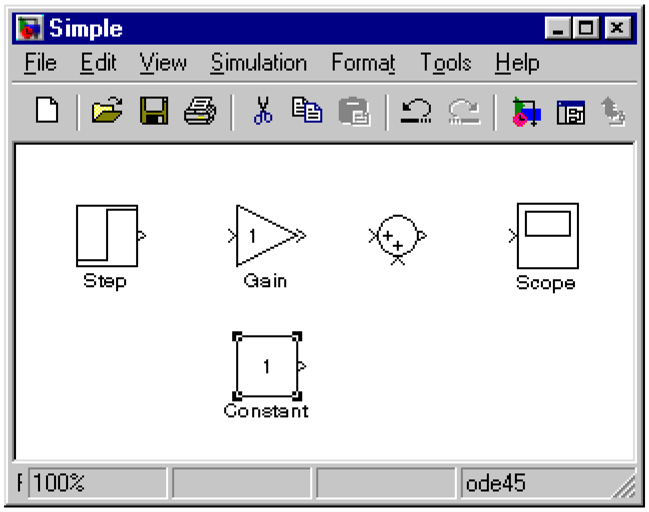
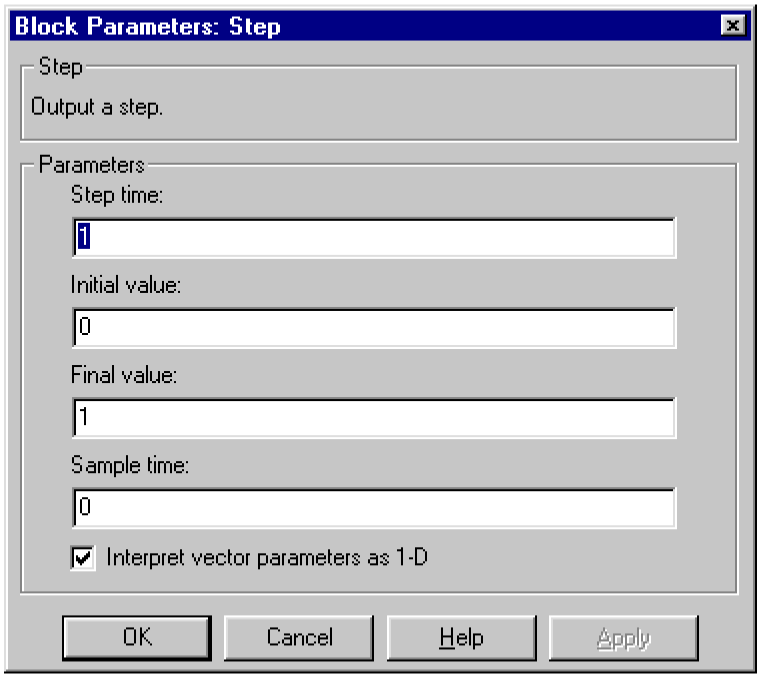


Figure 5 Blocks copied from the standard Simulink library


Block Parameters: Step

Step

Output a step.

Parameters

Step time:

1

Initial value:

0

Final value:

1

Sample time:

0

[✓] Interpret vector parameters as 1-D

Figure 6 The *Step* block's parameter menu

Defining input signals

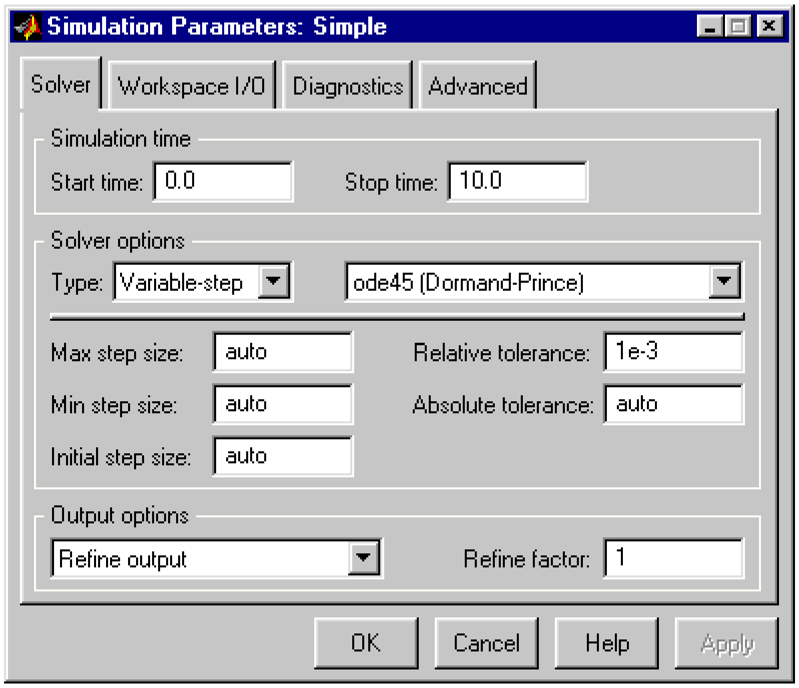
Now let's set up a simulation. First, we define the input signal details. Double-click on the *Step* block to open the menu shown in Figure 6. The default values shown will cause *x* to equal zero from the initial time until the step occurs (at time = 1), when *x* will increase to 1 (instantly).

The zero sample time signifies that the block's output will be a continuous function of time. If instead we were to use a positive value, the block's output would be defined at integer multiples of the sample time, but undefined at all other times.

Leave all parameters at their defaults for this example.

Simulation parameters

Next we check the *Simulation Parameters*. In your model window, click on the *Simulation* menu and select *Simulation Parameters* to open the window shown in Figure 7.



Simulation time
Start time: 0.0
Stop time: 10.0

Solver options
Type: Variable-step
ode45 (Dormand-Prince)
Max step size: auto
Relative tolerance: 1e-3
Min step size: auto
Absolute tolerance: auto
Initial step size: auto

Output options
Refine output
Refine factor: 1

Figure 7 Simulation parameters menu, Solver tab

The *Solver* tab allows you to specify the time at which the simulation starts and stops (which depends on the time scale of your problem[3]).

The other key area on the *Solver* tab is *Solver options*. The default, *ode45*, is a variable-step-size, 4th-order Runge Kutta method, which is a good all-around choice, but is poor for *stiff* problems[4], and there are situations where a fixed-step-size algorithm would be better. See the *Simulink Help* for details. We'll stick with *ode45* here.

Finally, the tolerances for the numerical solution can have an impact. If you ever suspect that your results are inaccurate, try decreasing the tolerance by an order of magnitude or two. If the results change significantly, decrease them even further. Also make sure your problem definition is reasonable, and that

[3] This can be a confusing and subtle issue. Your problem definition implicitly determines the units of time used in the simulation. All your equations must use consistent time units or the results will be incorrect.

[4] See also Riggs, Sec. 3.7, pp 123-125.

you are using the appropriate solver option. You should check the entries on the Workspace I/O tab; we'll use the defaults here.

Close the simulation parameters window. As the final preparation for the simulation, double-click on the *Scope* to open the graphical display.

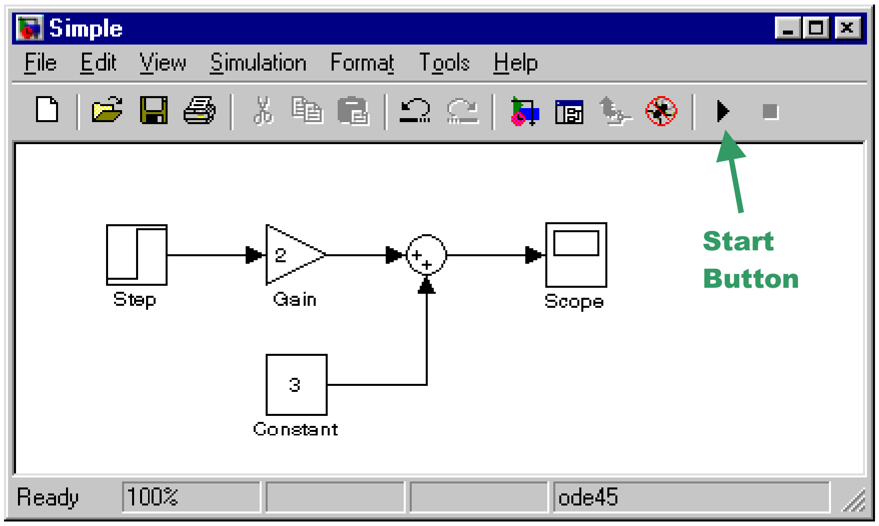


Figure 8 Ready to start the simulation

Running the simulation

Now click the *Start Button* (see Figure 8). The simulation begins (the *Start* button changes to a *Pause* icon, and the *Stop* button to its right becomes active). The elapsed time appears in the box just to the left of the simulation option (bottom of model window) and the *y* signal trace appears on the scope. (Since this simulation is trivial, it runs so rapidly that you may not be able to see things changing.) On a computer with sound you should hear a beep when the simulation ends.

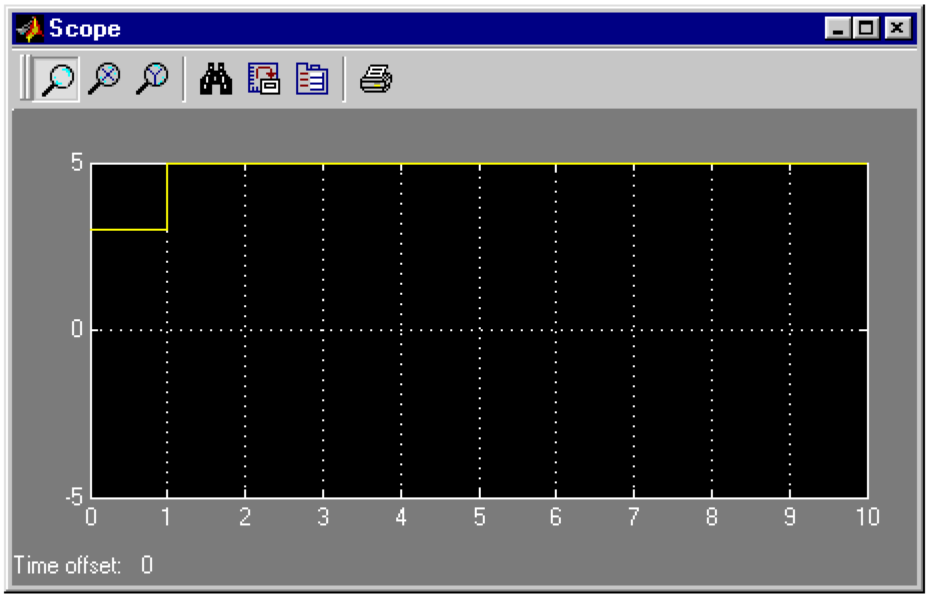


Figure 9 Scope display of the *y* signal

Figure 9 shows the final result (the yellow line). As one would expect from the problem definition, *y* starts at *3*, and increases to *5* at *t = 1*.

The default *y* scale (*-5* to *+5*) isn't a very good choice here. To zoom in on the *y* signal, try clicking on the *binocular* icon. Other icons allow you to zoom in on selected parts of the plot, permanently set the axis scales (for use in a later simulation), *etc*. See the *Scope* block's detailed description for more information.

Modeling a first-order differential equation

Most of the models you'll develop in ChemE 480 include unsteady-state conservation equations (mass, energy, and/or momentum), usually first-order differential equations (see, *e.g.*, Equation 3). As an example we consider a linear, first-order ODE with constant coefficients,

$$\frac{dy}{dt} + 2y = 3x + 1 \tag{4}$$

where *y* is a dependent function of time, and *x* is independent. To form a Simulink model we first rearrange (4) to solve for the derivative,

$$\frac{dy}{dt} = 3x + 1 - 2y \tag{5}$$

Then we arrange Simulink blocks to compute the *y* derivative, and *integrate* it to calculate *y*. (We need *y* to compute the derivative, so the model must include a "recycle".) Figure 10 shows one way to set it up (I have added the signal labels to help you understand it – they aren't part of the normal display).

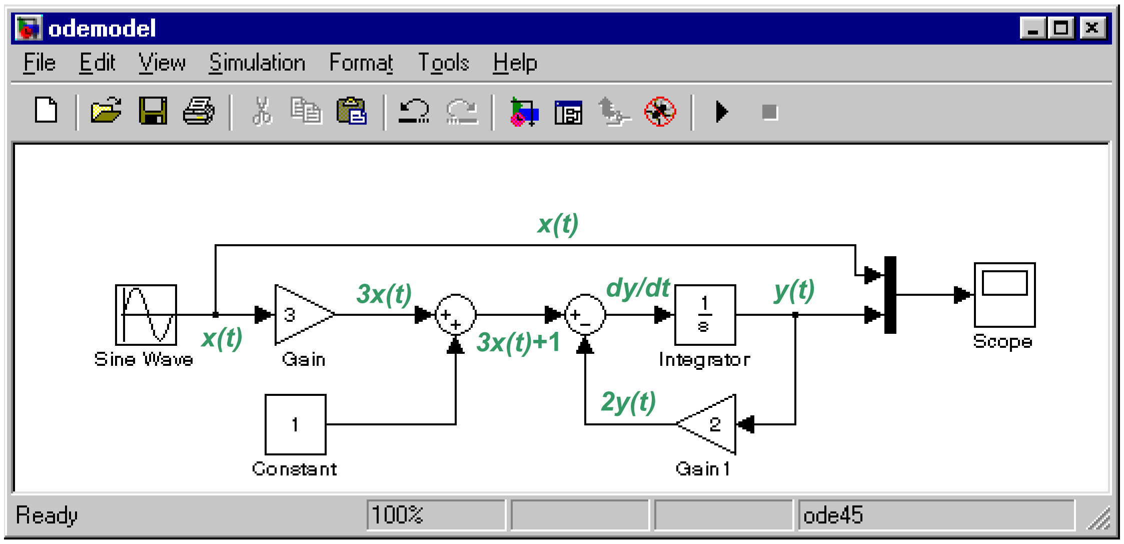


Figure 10 Simulink model of a linear, first-order ODE

Try duplicating this, keeping the following points in mind:

We're assuming (arbitrarily) that *x(t)* is a sinusoid. Use the default parameters for the *Sine Wave* block (in *Sources*), which gives you a continuous sine wave with a unity amplitude and a frequency of 1 radian per time unit (the documentation claims that the time unit is seconds, but that isn't true in general as explained previously). Thus, the sine wave's period will be 2π time units.

I dragged a *Gain* block in from the Library Browser, then used *control-click and drag* to make a copy. Simulink automatically gives the copy a different name (*Gain 1*). It requires all the block names in a system to be unique. Then I right-clicked on the copy and selected Format/Flip Block to reverse its direction (you can also rotate, *etc*.).

I also duplicated the *Sum* block. By default, Simulink hides its name, but each is unique. (You can right-click and use the format menu to show the name if you wish.)

One of the *Sum* blocks must do a subtraction. To accomplish this, double-click on the block and edit the *sign symbols* in the block parameter box. Try different combinations of the signs (and the vertical bar), clicking *apply* each time to see the effect (or read the block help). You can add a third input by including a third sign symbol.

The *Integrator* block is from the *Continuous* category. The significance of its 1/s icon derives from Laplace transforms. Its key parameter is the *initial condition*, which sets the value of its output when the simulation begins (*i.e.*, $y(0)$ in our case). Use the default (zero).

The black vertical bar just to the left of the scope is a *Mux* block (from *Signals & Systems*). It combines the scalar *x(t)* and *y(t)* signals into a *vector* signal. The only reason for doing this is to plot *x(t)* and *y(t)* on the same scope (the scope has only one input port, which can accept either a scalar or a vector signal). Another way would be to define two scopes, one connected to each scalar signal, which would be better if the *x* and *y* magnitudes were very different.

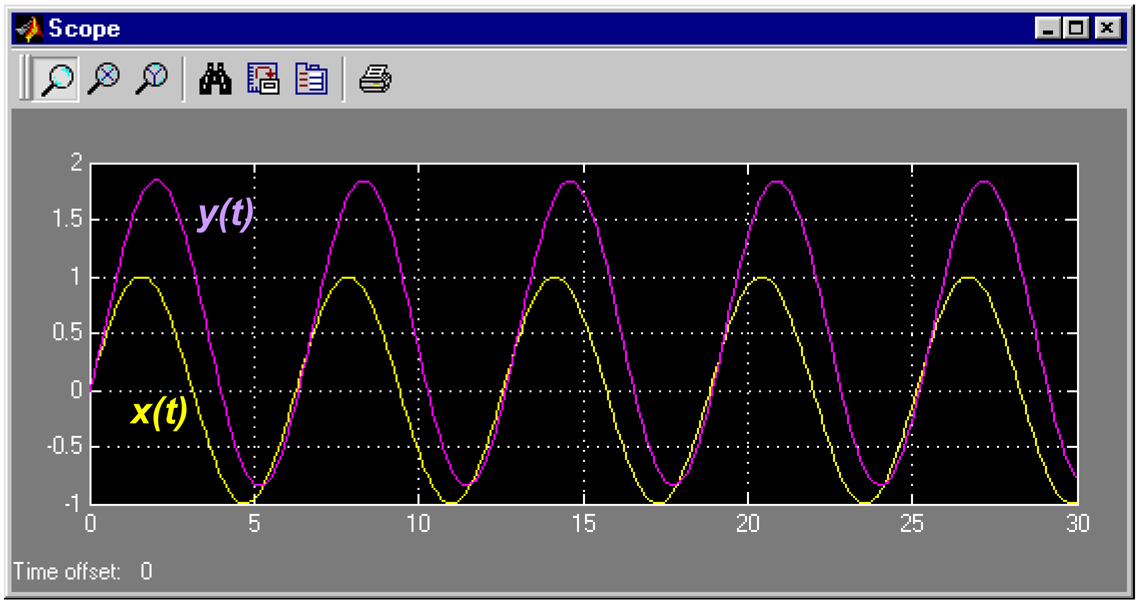


Figure 11 Simulation results for ODE example

The small black dots on two of the signal lines are *solder junctions* (another EE influence). All lines entering/leaving a solder junction are connected and carry the same signal.

To make a solder junction, move the cursor to the desired location, then control-click-and-drag to start drawing the new signal line. After you've made a solder joint you can move it around by clicking-and-dragging. You can also move signal line sections and blocks.

It's possible for signal lines to cross without forming a solder junction (try it). In that case, the signals are unconnected.

Use the *Simulation Parameters* menu to set the stop time to 30. To make the plots look better, set the maximum step size to 0.1[5]. Next, run the simulation

Figure 11 shows the results (with superimposed *x(t)* and *y(t)* signal labels for clarity). Use the *binoculars* icon to zoom in on the curves. The steady sinusoidal input, *x(t)*, eventually causes a sinusoidal output having the same frequency but a different phase. We will study such *frequency responses* in more detail later.

Using subsystems

Modeling a single algebraic or differential equation is fairly easy, but what about multi-equation systems? It's really no different. You define each equation and its variables (signals), then combine equations by connecting the signals they have in common.

One problem is that the diagram can become complicated, making it hard to understand the model. You can reduce the apparent complexity by defining *subsystems*.

For example, consider combining (5) with another differential equation:

$$\frac{dz}{dt} = 2x - y - 3z \tag{6}$$

Adding this to our previous model would be easy, and the diagram would still be fairly clear, but we will use subsystems to illustrate the concept.

Start with a new model window. Drag in a *SubSystem* block (from the *SubSystems* group). Edit the block name, changing it to "Equation 5". Next make a copy, naming it "Equation 6." Your model window should now resemble Figure 12 (I have named the model *TwoODEs*).

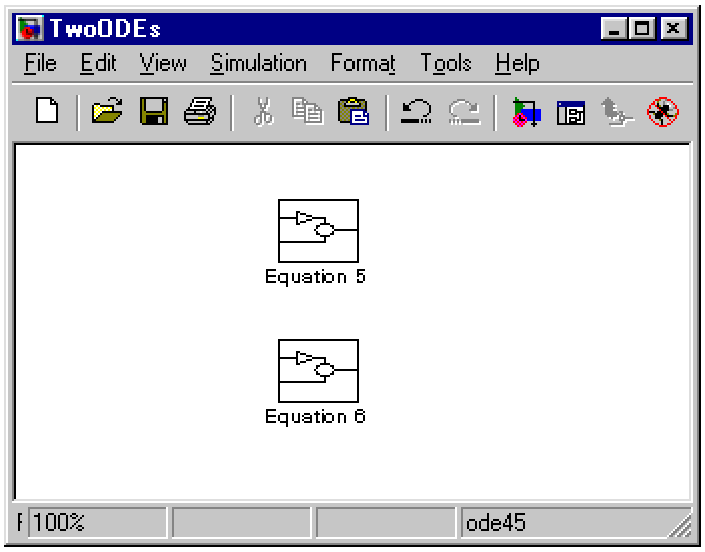


Figure 12 Two (renamed) subsystem blocks

Double-click on *Equation 5* to open it. You get a new blank window, which looks just like a normal model window. We will use this to define equation 5.

[5] If you don't do this, Simulink will maximize the step size to speed up the calculations. As a result, some of the points will be far apart and joined by straight lines, so they won't look like true sinusoids. The calculated points will still be accurate, however.

First note that equation 5 involves one "input" variable (x) and one "output" (y). To allow x to enter the subsystem we need to define an *input port*. Find the block named *In1* in the Library Browser's *Signals & Systems* category. Drag it into the Equation 5 window and rename it "x".

Similarly, we need to send the y signal out of this subsystem (it's needed in equation 6 and we may wish to plot it). Find the Browser's *Out1* block, drag it into the Equation 5 window, and rename it y. Then define Equation 5 as before[6]. Figure 13 shows the completed subsystem window.

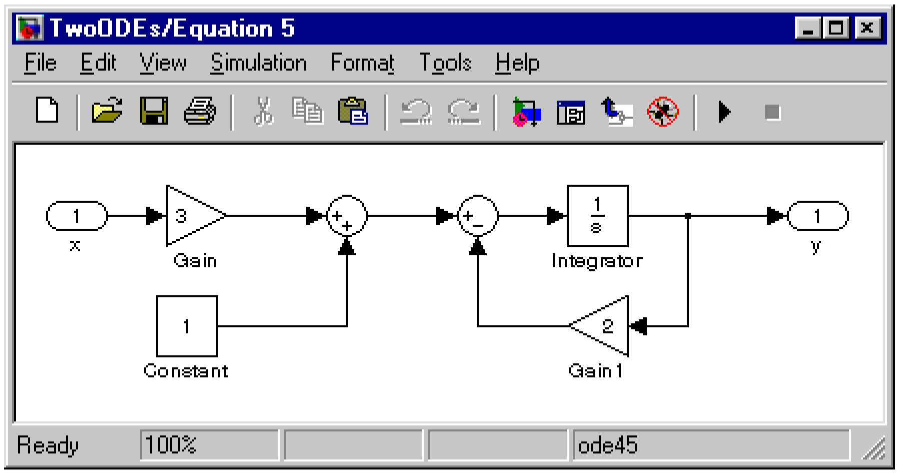


Figure 13 Equation 5 subsystem

Close the Equation 5 subsystem and return to the main model window. The Equation 5 icon has changed. It now shows the input and output ports, which are labeled with the appropriate names. Repeat the procedure to create equation 6, which requires two inputs (x and y) and generates one output (z).

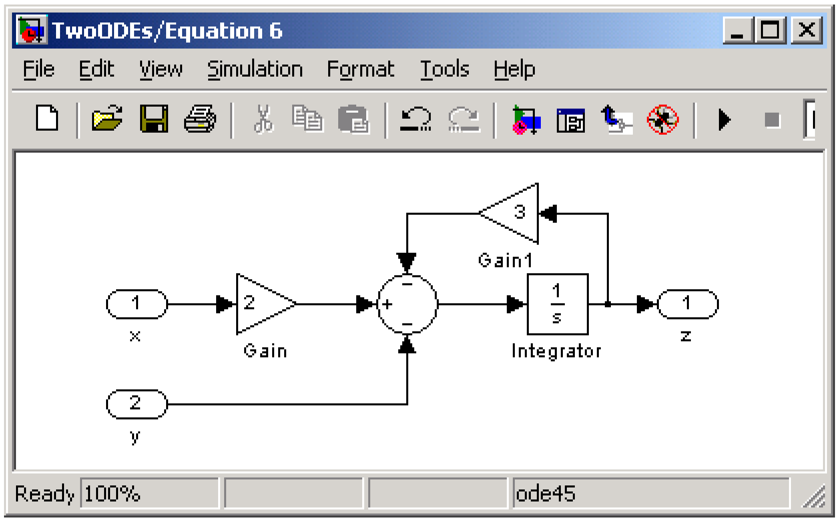


Figure 14 Equation 6 subsystem

Figure 14 shows one possible arrangement. Note the use of a 3-input summation block (you could also use two 2-input summation blocks). Close Equation 6 and return to the main model window.

[6] If you still have the model we developed in the previous section you can copy-and-paste its blocks into the Equation 5 window. Just drag a selection rectangle around the blocks you want to copy.

Connect the subsystems (output y of Equation 5 to input y of Equation 6). Then define the *x(t)* signal. This time use the *Pulse Generator*. Set its period to 10 and leave its other block parameters at their defaults. Add a *Mux* (with 3 inputs – change the block parameter from 2 to 3) and *Scope*, and set up to plot *x*, *y*, and *z* on the same scope.

Figure 15 shows the final arrangement. (Note the way the *x* and *y* signals cross – they are unconnected.) Define the *Simulation Parameters* for a 30 time-unit run, with a maximum step size of 0.1. Then run the simulation

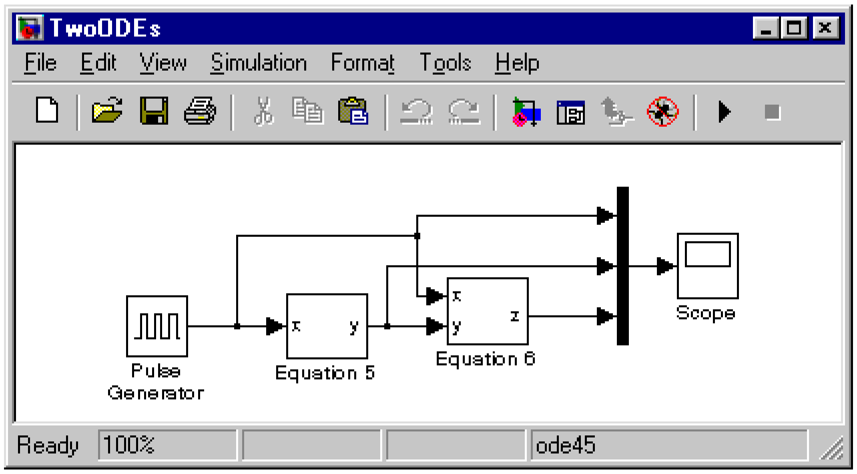


Figure 15 Main model window with defined subsystems and input

Figure 16 shows the results. The yellow trace is the periodic pulse input, *x(t)*. The purple is *y(t)*, and the cyan is *z*(t). As with the sinusoidal input, the outputs eventually behave periodically. Would you have anticipated this result?

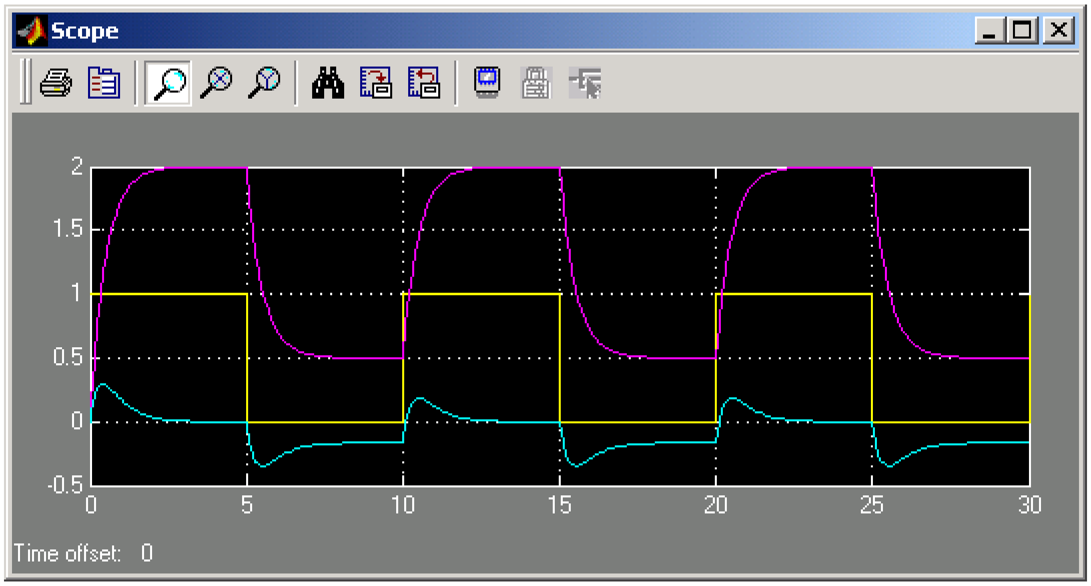


Figure 16 Simulation results for TwoODE system

Heat exchanger model development

Specifications

The previous sections have covered the tools needed to simulate the heat exchanger. As an exercise, define a Simulink model of equations 2 and 3 (you can ignore equation 1, which is just a definition). It should have three independent variables (*inputs*): T_i, T_s, and $\dot{w}$. Also define two dependent variables (*outputs*): $\dot{q}$ and T_o. The model's initial conditions should be a steady-state with the inputs at the following values:

$$T_i = 100$$
$$\dot{w} = 100/3$$
$$T_s = 150$$

Define each input to be a step starting from the above initial conditions. Plot the two outputs on separate scopes. (Hint: the *Gain* block multiplies a signal by a constant. It doesn't multiply two signals. You'll need another block for that.)

Run three simulations. In each case, make a *unit-step* change in one of the inputs (beginning at $t = 5$), and hold the other two inputs constant. Each simulation should vary a different input. Run each simulation for 50 time units. Save the model for use later. Answer the following questions:

1. What time units are being used?
2. What are the initial conditions for the two outputs?
3. Are the responses qualitatively reasonable in each case?

A solution – Simulink diagrams

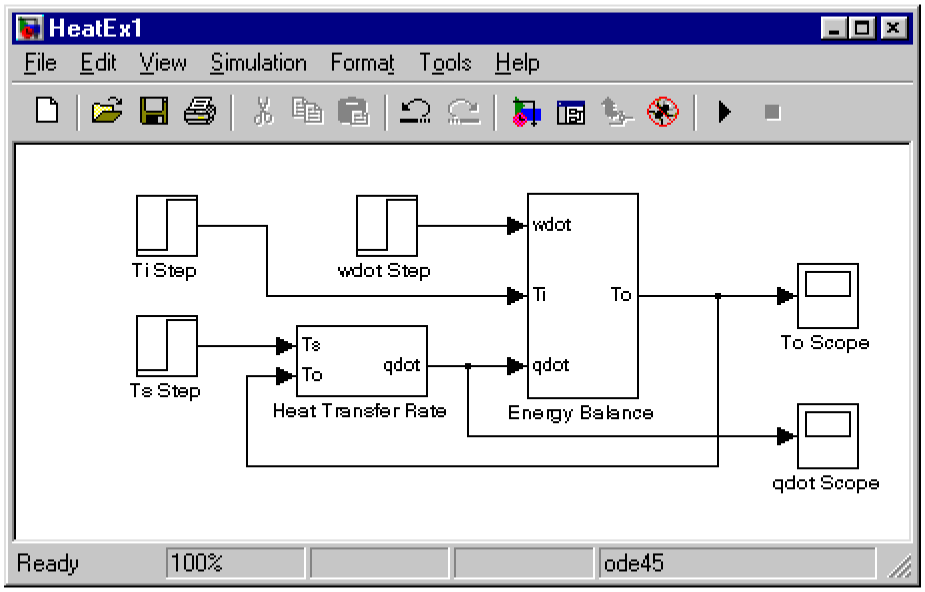


Figure 17 Heat exchanger model -- main window

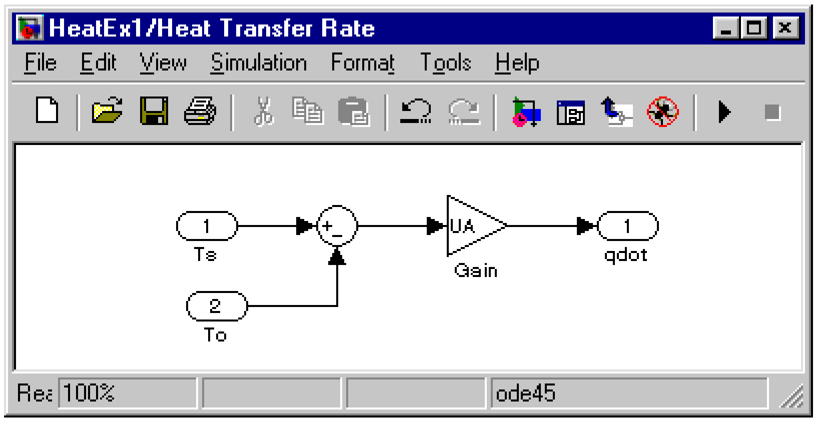

HeatEx1/Heat Transfer Rate
File Edit View Simulation Format Tools Help
1
Ts
2
To
+
_
UA
Gain
1
qdot
Rea 100%
ode45

Figure 18 Heat transfer rate subsystem

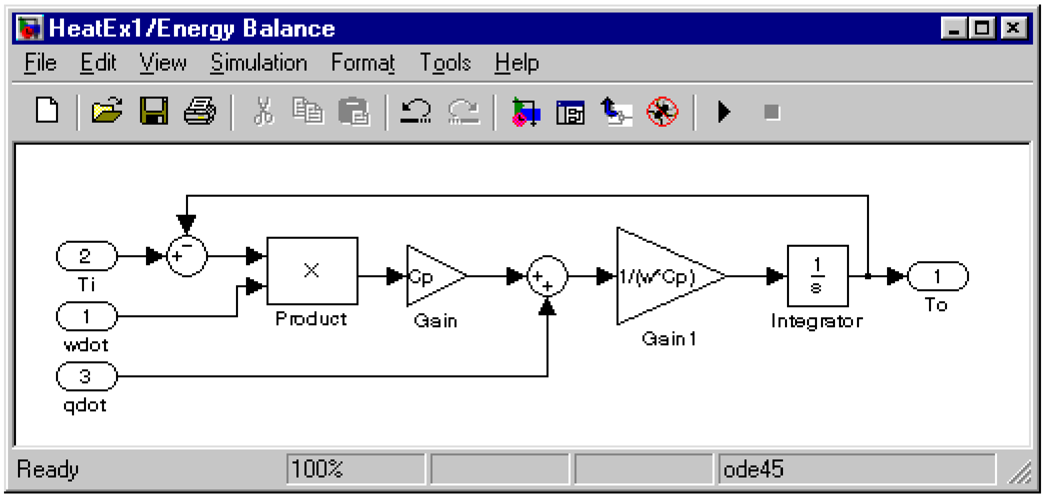
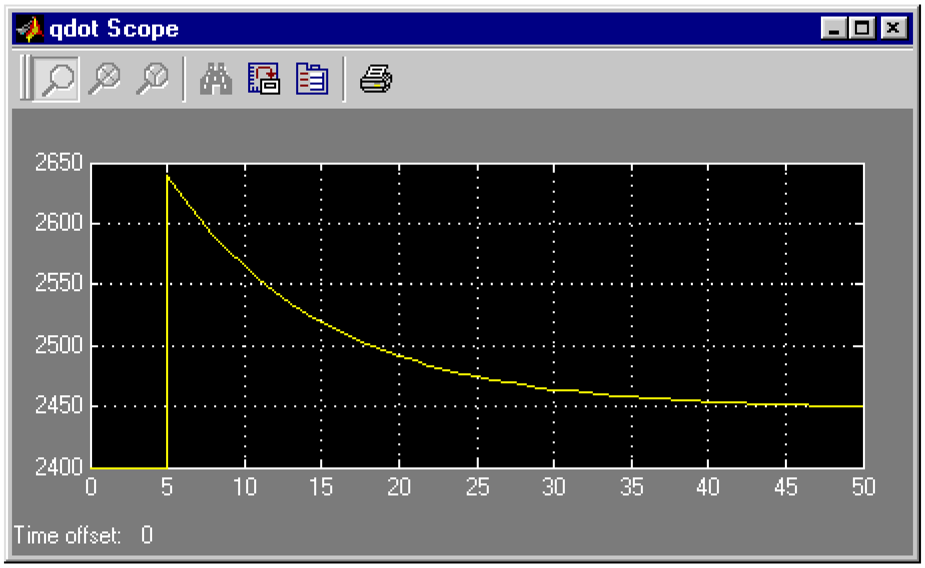
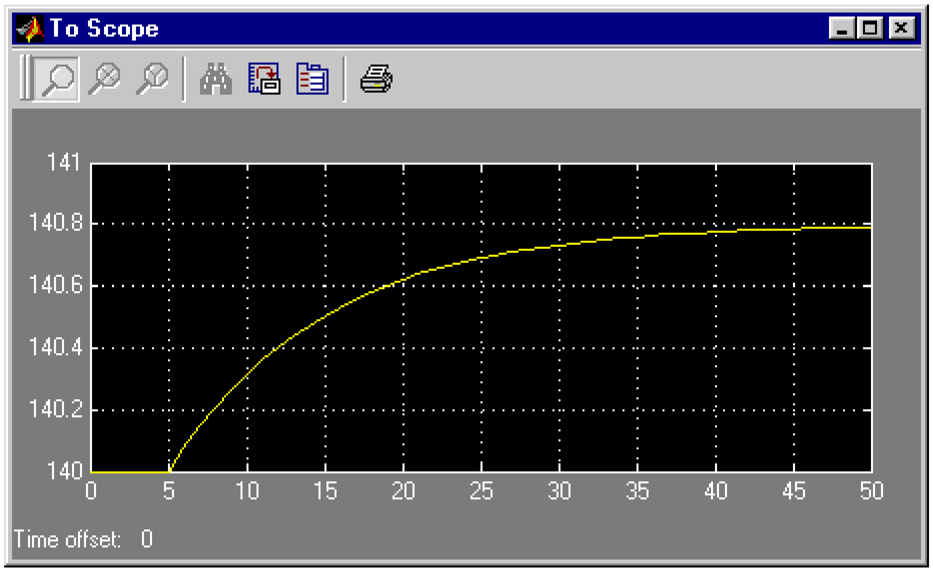


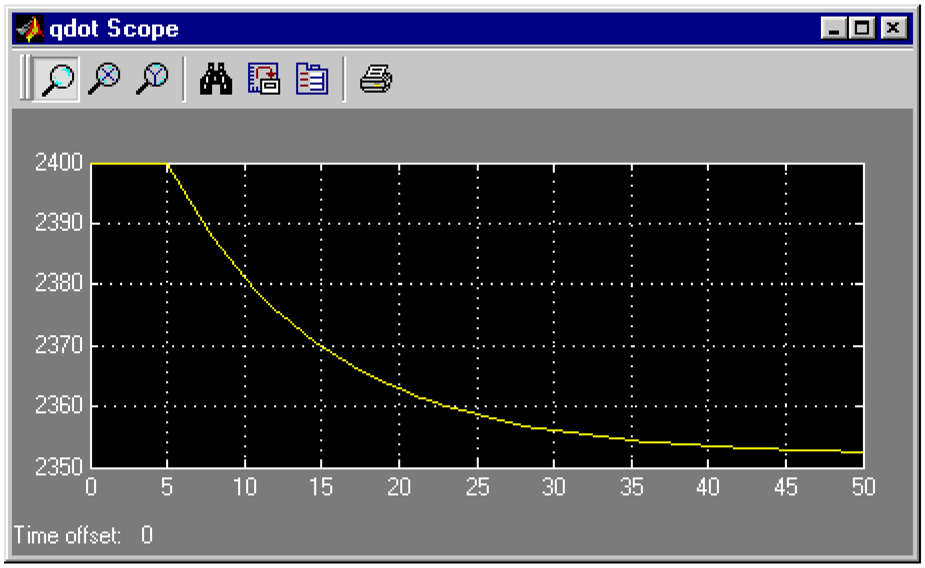
Figure 19 Energy balance subsystem

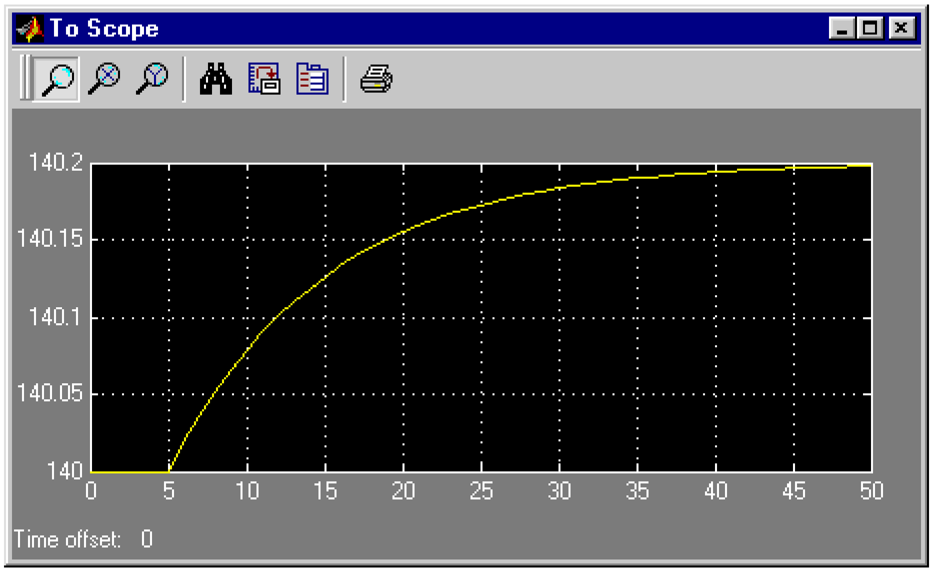
Response to T_s unit step



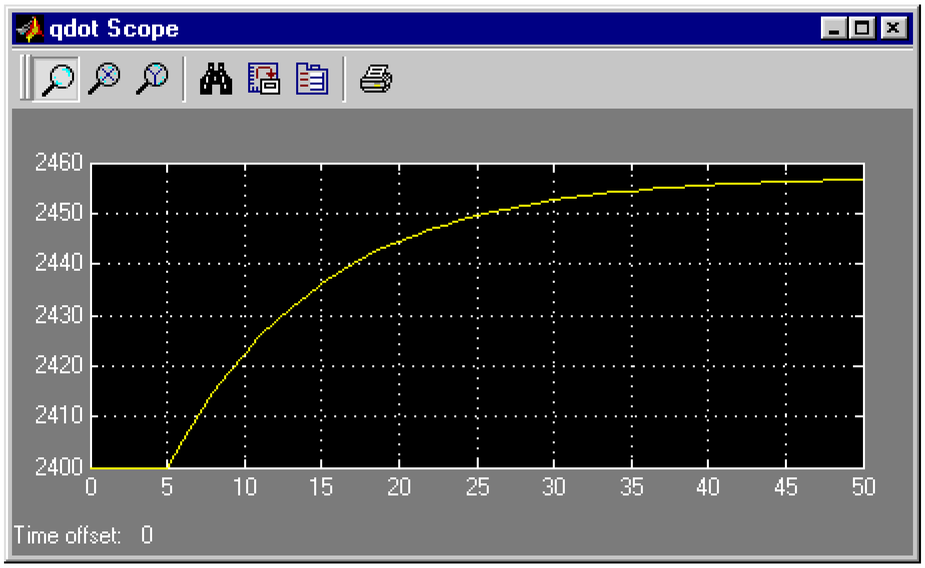


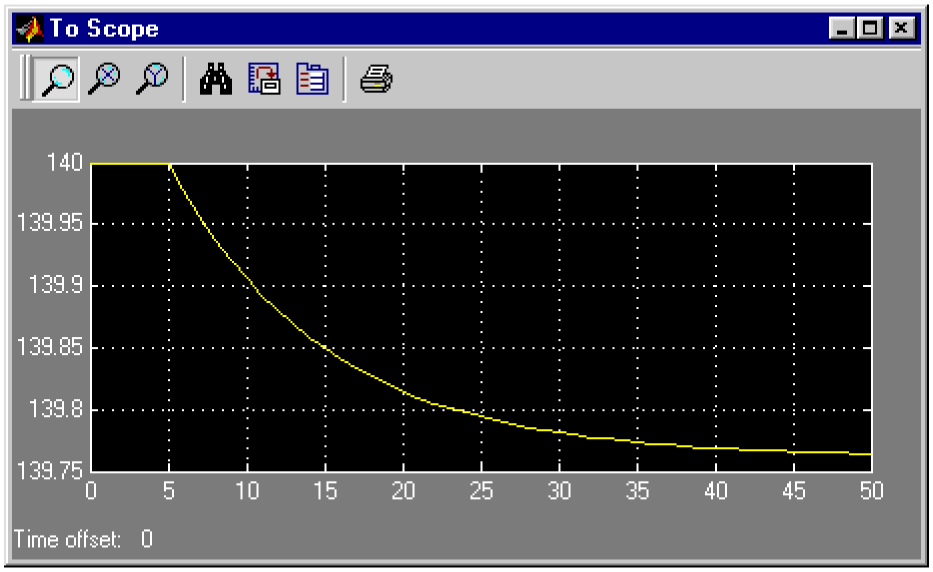
Response to T_i unit step





Response to inlet flow rate unit step





Remarks

- The initial conditions are $T_o = 140$, $\dot{q} = 2400$.
- The time units are seconds (because we defined all heat exchanger model parameters in terms of seconds).
- The integrator in Figure 19 requires an initial condition for T_o (the initial steady-state value).
- Note the use of variable names rather than numbers in the gain block parameter definitions. If you do this you must define each such variable in the MATLAB workspace before running the simulation. Thus, for example, you would type `Cp=1.8;` in the Command window to define the specific heat value. This may seem cumbersome, but it makes the model easier to understand. It also makes it easy to modify a parameter – you just change its value in the workspace instead of worrying about modifying all the blocks in which that parameter appears.
- The response of T_o to a unit-step in T_i is the *least* realistic. If the tubes were long and flow were turbulent (*plug flow*) it would take a while for the change in T_i to show up in T_o (because the fluid temperature change would move down the tube, carried along by convection).

A more realistic heat exchanger model

To better approximate plug flow in the tubes we divide the exchanger into *sections*. Each section will use the same equations and model structure developed previously. In other words, each section will have well-mixed tubes at a uniform temperature. The fluid leaving one section flows to the next. The more sections, the more closely we approximate plug flow.

Creating a masked block

To make this easy to do we introduce another Simulink feature: *creating a masked block*. (This is how the blocks in the Library were made.) You should consider doing this any time you create a subunit that could be re-used in another model[7].

To begin, create a new model window and put a single subsystem block in it, and name it *HeatX Section*. Open the subsystem window. Open your saved heat exchanger model and copy it into the subsystem block. Define T_s, T_i, and $\dot{w}$ as input ports, and T_o and $\dot{w}$ as output ports (eliminate the step function and

[7] You can maintain your own library of specialized blocks. See the Simulink documentation for more about libraries.

scope blocks). All block parameters should be in terms of the variable names *w*, *Cp*, and *UA*, as shown in Figure 18 and Figure 19. Also, use the name *To_0* to represent the integrator's initial condition.

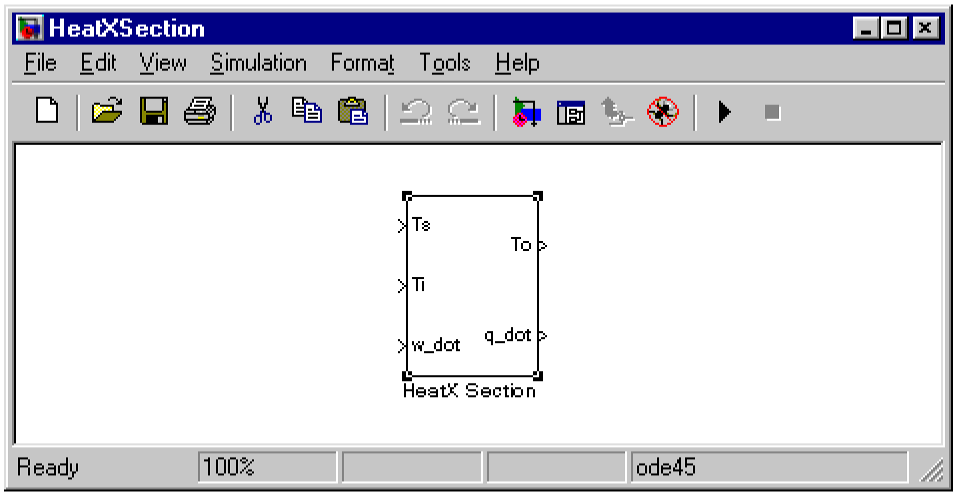


Figure 20 Block to model a heat exchanger section

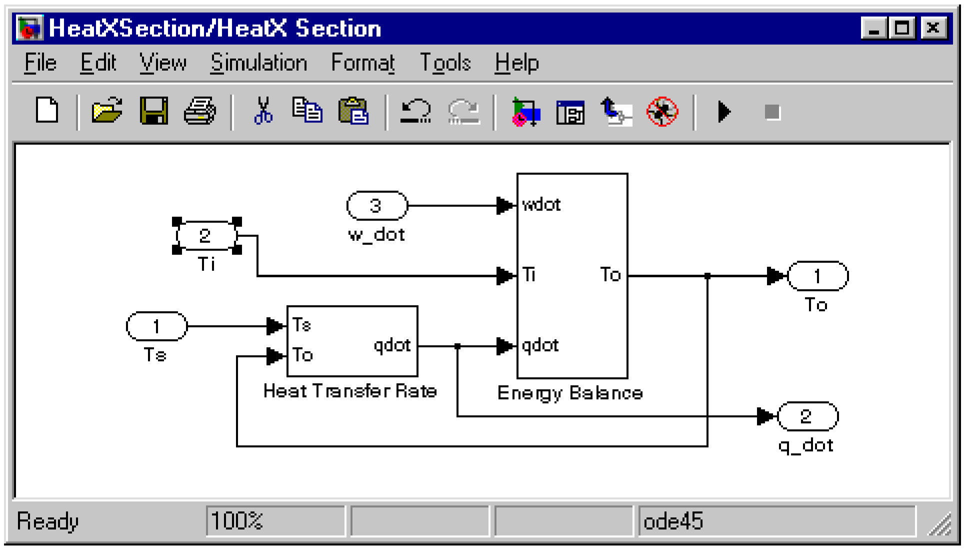
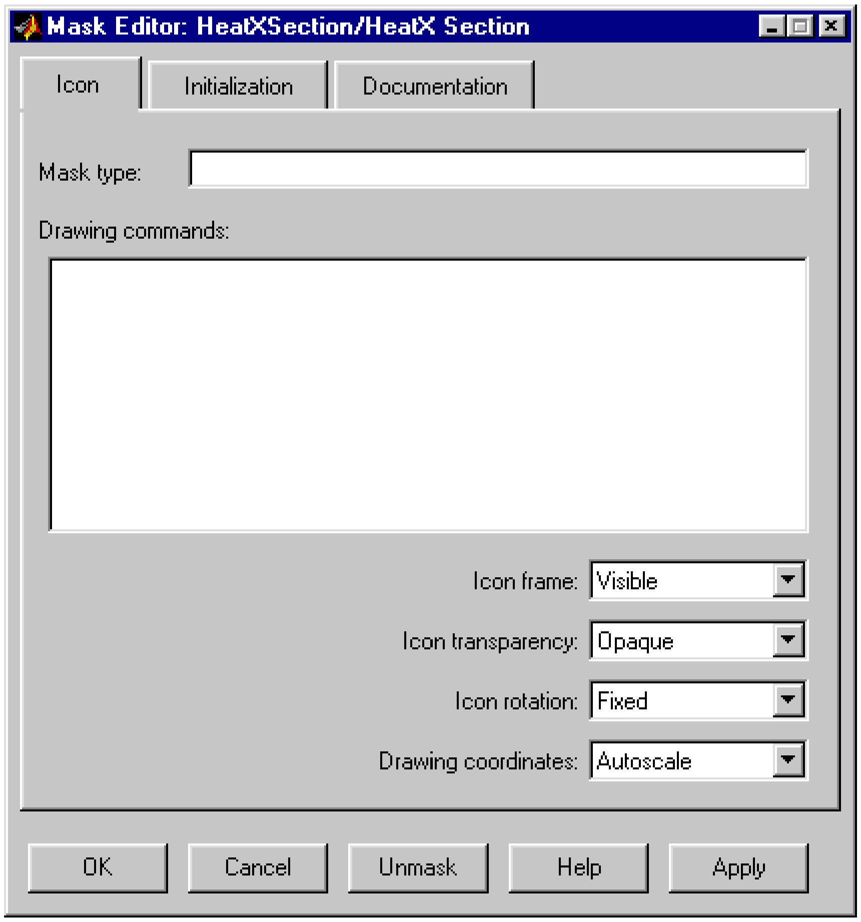


Figure 21 Heat exchanger section details

The result should resemble Figure 20 and Figure 21. Now we mask the block, which isolates its parameters from those defining other blocks, and provides a convenient user interface. Close the subsystem block window (Figure 21), right-click on the HeatX Section block in the main window (Figure 20), and select *mask subsystem*. This opens the *Mask Editor* window (see Figure 22).



Mask type:

Drawing commands:

Icon frame: Visible

Icon transparency: Opaque

Icon rotation: Fixed

Drawing coordinates: Autoscale

Figure 22 Mask editor -- Icon tab

The *Icon* tab allows you to create an icon for your block. We won't bother with that, but you might want to use it in the future.

Select the *Initialization* tab (see Figure 23). Its main purpose is to define any variables appearing in your model. We will create an interface that allows one to specify the specific heat, liquid mass, *UA* value, and initial tube fluid temperature.

Start by typing "Initial fluid temperature, C" in the *prompt* box. Then type the corresponding variable name (*To_0*) in the *variable* box. You can leave the *Control type* and *Assignment* selections at their defaults. Then click on the *Add* button, which allows you to define an additional variable. Repeat for the remaining three.

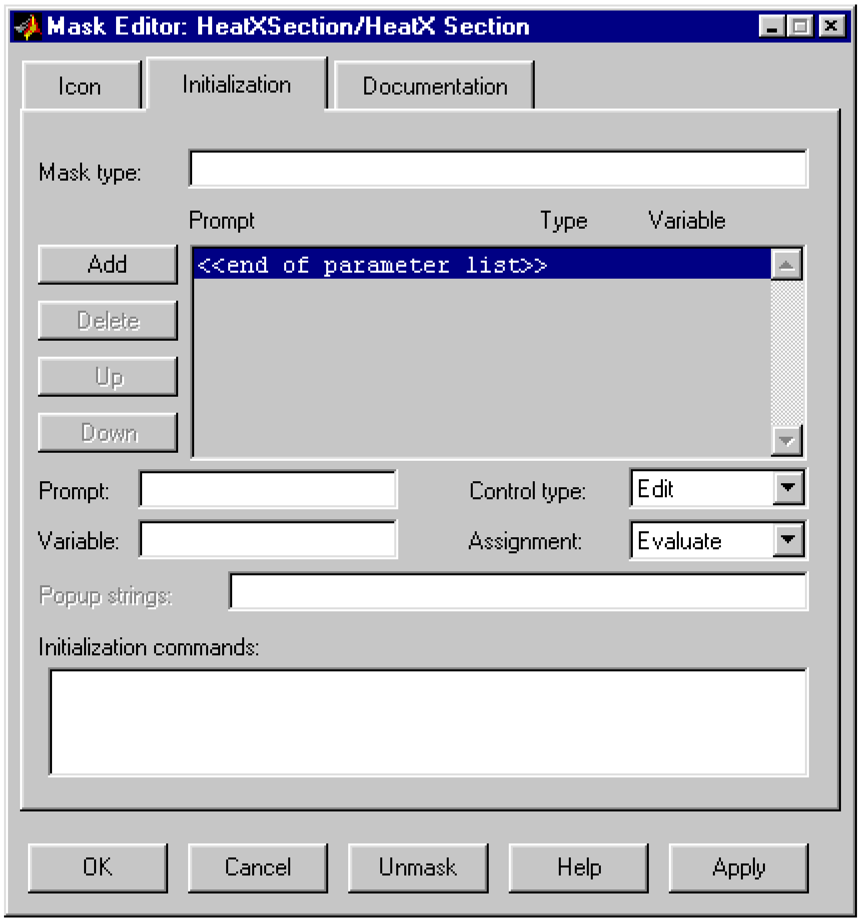
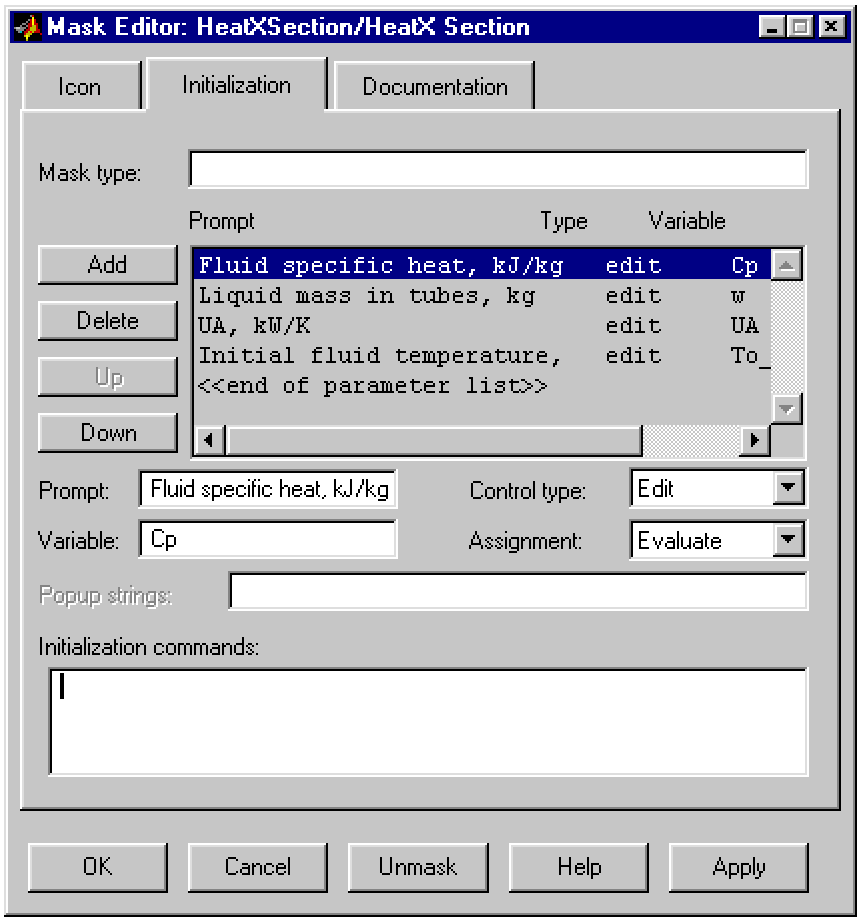


Figure 23 Mask editor -- Initialization tab

The Mask editor window should now resemble Figure 24. It will depend, or course, on the descriptions you used and the order in which you entered them. If you wish, you can modify the order by selecting one of the entries in the parameter list box and using the *Up* or *Down* button.

The *Initialization commands* area allows you to enter MATLAB commands as needed to define things in your model prior to execution. This provides a great deal of flexibility. These commands can use variables in your parameter list as inputs[8]. But leave it blank for this exercise.

[8] See the Simulink documentation for more on masking.



Mask type:

```
Prompt                           Type     Variable
Fluid specific heat, kJ/kg       edit     Cp
Liquid mass in tubes, kg         edit     w
UA, kW/K                         edit     UA
Initial fluid temperature,       edit     To_
<<end of parameter list>>
```

Prompt: Fluid specific heat, kJ/kg

Control type: Edit

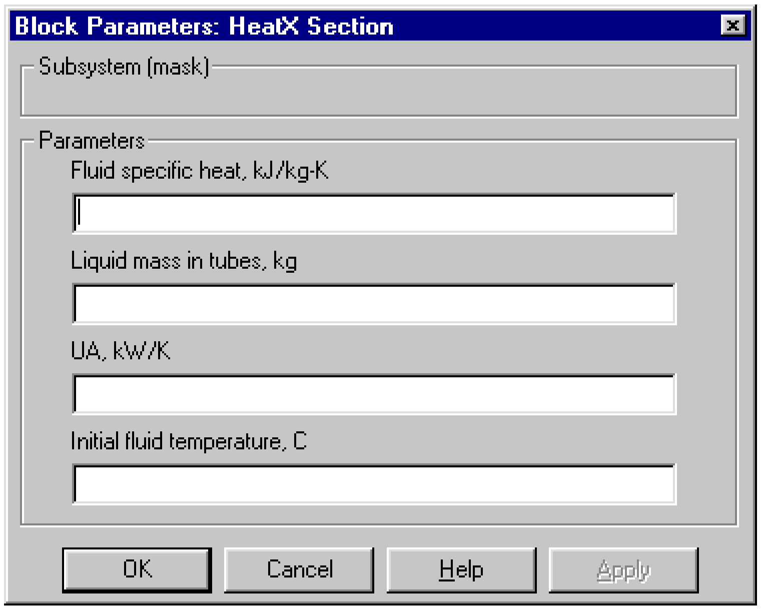
Variable: Cp

Assignment: Evaluate

Popup strings:

Initialization commands:

Figure 24 Mask editor with block parameters defined



Fluid specific heat, kJ/kg-K

Liquid mass in tubes, kg

UA, kW/K

Initial fluid temperature, C

Figure 25 User interface for the HeatX Section block

Now click the *OK* button to close the Mask Editor (don't use the close button or you'll lose your entries). Double-click on your HeatX Section block. Now instead of opening the subsystem window you get a block parameter dialog window, which should resemble Figure 25.

The variables within the masked block are now isolated from the MATLAB workspace and all other blocks. The only way to set the parameters is through the dialog window. This makes inadvertent changes less likely.

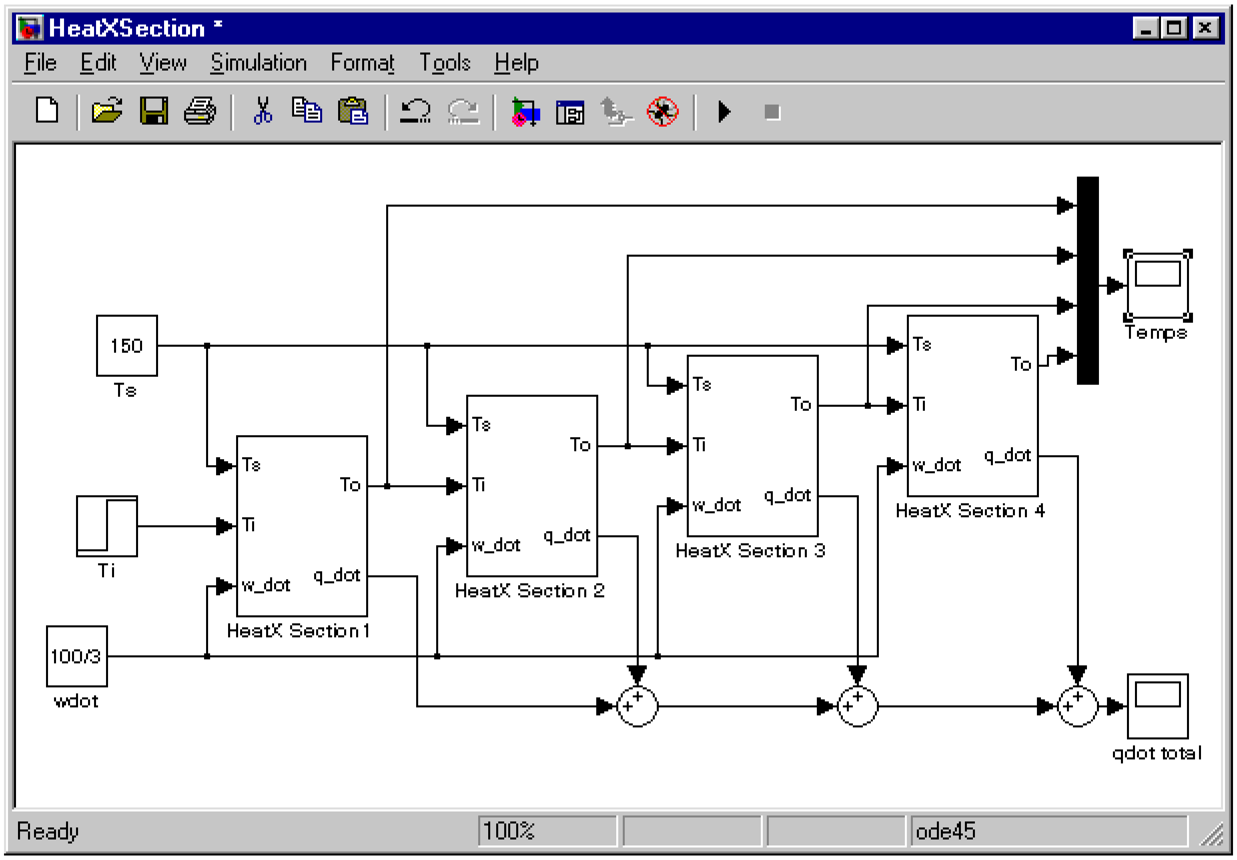
If you wish to see or modify the underlying model, right-click on the block and select *Look under mask*. Some changes may require you to modify the mask, in which case you should right-click on the block and select *Edit mask*.

Using the masked block

Before using a new block as a building block you need to test it to make sure it's working as expected. Check its operation against the results you obtained previously. When convinced that it's OK, use the block to create a 4-section heat exchanger. This should have the same *total* heat exchange area and tube volume as before, and the tube volume in each section should be ¼ of the total. Plot the outlet temperature from each section on one scope, and the total heat transfer rate on another. Answer the following questions:

1. If the inputs are the same as before, what are the steady-state outputs?
2. Is there a significant qualitative difference in the response to a unit-step in T_i?

A 4-section exchanger configuration using the masked block



Remarks

- The steady-state temperatures leaving each section are 125, 137.5, 143.75, and 146.875 C, respectively. Thus, plug flow provides more efficient use of the heat exchanger area (higher outlet temperature). A convenient way to find these values is to start the model at a reasonable initial condition and run it with constant inputs for long enough to reach steady state. You could also solve the steady-state equations algebraically.

- The step in T_i now causes a much slower increase in T_o, including an initial delay of about 5 seconds before anything happens. To see this you will need to zoom in on the final outlet temperature, or plot it on a separate scope.