**The lecture 4**

**Modeling dynamic systems**

A Simulink block diagram model is a graphical representation of a mathematical model of a dynamic system. A mathematical model of a dynamic system is described by a set of equations. The mathematical equations described by a block diagram model are known as algebraic, differential, and/or difference equations.

**Block Diagram Semantics**

A classic block diagram model of a dynamic system graphically consists of blocks and lines (signals). The history of these block diagram model is derived from engineering areas such as Feedback Control Theory and Signal Processing. A block within a block diagram defines a dynamic system in itself. The relationships between each elementary dynamic system in a block diagram are illustrated by the use of signals connecting the blocks. Collectively the blocks and lines in a block diagram describe an overall dynamic system. Simulink extends these classic block diagram models by introducing the notion of two classes of blocks, nonvirtual block and virtual blocks. Nonvirtual blocks represent elementary systems. A virtual block is provided for graphical organizational convenience and plays no role in the definition of the system of equations described by the block diagram model. Examples of virtual blocks are the Bus Creator and Bus Selector which are used to reduce block diagram clutter by managing groups of signals as a “bundle.” You can use virtual blocks to improve the readability of your models.

In general, block and lines can be used to describe many “models of computations.” One example would be a flow chart. A flow chart consists of blocks and lines, but one cannot describe general dynamic systems using flow chart semantics.

The term “time-based block diagram” is used to distinguish block diagrams that describe dynamic systems from that of other forms of block diagrams. In Simulink, we use the term block diagram (or model) to refer to a time-based block diagram unless the context requires explicit distinction.

**Creating Models**

Simulink provides a graphical editor that allows you to create and connect instances of block types selected from libraries of block types via a library browser. Simulink provides libraries of blocks representing elementary systems that can be used a building blocks. The blocks supplied with Simulink are called built-in blocks. Simulink users can also create their own block types and use the Simulink editor to create instances of them in a diagram. Customer-defined blocks are called custom blocks.

**Time**

Time is an inherit component of block diagrams in that the results of a block diagram simulation change with time. Put another way, a block diagram represents the instantaneous behavior of a dynamic system. Determining a system’s behavior over time thus entails repeatedly executing the model at intervals, called time steps, from the start of the time span to the end of the time span. Simulink refers to the repeated execution of a model at successive time steps as simulating the system that the model represents. It is possible to simulate a system manually, i.e., to execute its model manually. However, this is unnecessary as the Simulink engine performs this task automatically on command from the user.

**States**

Typically, the current values of some system, and hence model, outputs are functions of the previous values of temporal variables. Such variables are called states. Computing a model’s outputs from a block diagram hence entails saving the value of states at the current time step for use in computing the outputs at a subsequent time step. Simulink performs this task during simulation for models that define states.

Two types of states can occur in a Simulink model: discrete and continuous states. A continuous state changes continuously. Examples of continuous states are the position and speed of a car. A discrete state is an approximation of a continuous state where the state is updated (recomputed) using finite (periodic or aperiodic) intervals. An example of a discrete state would be the position of a car shown on a digital odometer where it is updated every second

as opposed to continuously. In the limit, as the discrete state time interval approaches zero, a discrete state becomes equivalent to a continuous state.

Blocks implicitly define a model’s states. In particular, a block that needs some or all of its previous outputs to compute its current outputs implicitly defines a set of states that need to be saved between time steps. Such a block is said to have states.

The following is a graphical representation of a block that has states.



Blocks that define continuous states include the following standard Simulink blocks:

**•**Integrator

**•**State-Space

**•**Transfer Fcn

**•**Zero-Pole

The total number of a model’s states is the sum of all the states defined by all its blocks. Determining the number of states in a diagram requires parsing the diagram to determine the types of blocks that it contains and then aggregating the number of states defined by each instance of a block type that defines states. Simulink performs this task during the Compilation phase of a simulation.

**Continuous States**

Computing a continuous state entails knowing its rate of change, or derivative. Since the rate of change of a continuous state typically itself changes continuously (i.e., is itself a state), computing the value of a continuous state at the current time step entails integration of its derivative from the start of a simulation. Thus, modeling a continuous state entails representing the operation of integration and the process of computing the state’s derivative at each point in time. Simulink block diagrams use Integrator blocks to indicate integration and a chain of operator blocks connected to the integrator block to represent the method for computing the state’s derivative. The chain of block’s connected to the Integrator’s is the graphical counterpart to an ordinary differential equation (ODE).

In general, excluding simple dynamic systems, analytical methods do not exist for integrating the states of real-world dynamic systems represented by ordinary differential equations. Integrating the states requires the use of numerical methods called ODE solvers. These various methods trade computational accuracy for computational workload. Simulink comes with computerized implementations of the most common ODE integration methods and allows a user to determine which it uses to integrate states represented by Integrator blocks when simulating a system.

Computing the value of a continuous state at the current time step entails integrating its values from the start of the simulation. The accuracy of numerical integration in turn depends on the size of the intervals between time steps. In general, the smaller the time step, the more accurate the simulation. Some ODE solvers, called variable time step solvers, can automatically vary the size of the time step, based on the rate of change of the state, to achieve a specified level of accuracy over the course of a simulation. Simulink allows the user to specify the size of the time step in the case of fixed-step solvers or allow the solver to determine the step size in the case of variable-step solvers. To minimize the computation workload, the variable-step solver chooses the largest step size consistent with achieving an overall level of precision specified by the user for the most rapidly changing model state. This ensures that all model states are computed to the accuracy specified by the user.

**Discrete States**

Computing a discrete state requires knowing the relationship between the current time and its value at the time at which it previously changed value. Simulink refers to this relationship as the state’s update function. A discrete state depends not only on its value at the previous time step but also on the values of a model’s inputs. Modeling a discrete state thus entails modeling the state’s dependency on the systems’ inputs at the previous time step. Simulink block diagrams use specific types of blocks, called discrete blocks, to specify update functions and chains of blocks connected to the inputs of the block’s to model the state’s dependency on system inputs.

As with continuous states, discrete states set a constraint on the simulation time step size. Specifically, a step size must be chosen that ensure that all the sample times of the model’s states are hit. Simulink assigns this task to a component of the Simulink system called a discrete solver. Simulink provides two discrete solvers: a fixed-step discrete solver and a variable-step discrete solver. The fixed-step discrete solver determines a fixed step size that hits all the sample times of all the model’s discrete states, regardless of whether the states actually change value at the sample time hits. By contrast, the variable-step discrete solver varies the step size to ensure that sample time hits occur only at times when the states change value.

**Modeling Hybrid Systems**

A hybrid system is a system that has both discrete and continuous states Strictly speaking a hybrid model is identified as having continuous and discrete sample times from which it follows that the model will have continuous and discrete states. Solving a model of such a system entails choosing a step size that satisfies both the precision constraint on the continuous state integration and the sample time hit constraint on the discrete states. Simulink meets this requirement by passing the next sample time hit as determined by the discrete solver as an additional constraint on the continuous solver. The continuous solver must choose a step size that advances the simulation up to but not beyond the time of the next sample time hit. The continuous solver can take a time step short of the next sample time hit to meet its accuracy constraint but it cannot take a step beyond the next sample time hit even if its accuracy constraint allows it to.

**Block Parameters**

Key properties of many standard blocks are parameterized. For example, the Constant value of the Simulink Constant block is a parameter. Each parameterized block has a block dialog that lets you set the values of the parameters. You can use MATLAB expressions to specify parameter values. Simulink evaluates the expressions before running a simulation. You can change the values of parameters during a simulation. This allows you to determine interactively the most suitable value for a parameter.

A parameterized block effectively represents a family of similar blocks. For example, when creating a model, you can set the Constant value parameter of each instance of the Constant block separately so that each instance behaves differently. Because it allows each standard block to represent a family of blocks, block parameterization greatly increases the modeling power of the standard Simulink libraries.

Each time you change parameters, you change the meaning of the model. Simulink lets you modify the parameter values during execution of your model.

For example, you can pause simulation, change parameter values, and continue simulation. It should be pointed out that parameter changes do not immediately occur, but are queued up and then applied at the start of the next time step during model execution. Returning to our example of the constant block, the function it defines is *signal*(*t*) = *Cons*tan*tValue* for all time. If we were to allow the constant value to be changed immediately, then the solution at the point in time at which the change occurred would be invalid, thus we must queue the change for processing on the next time step.

**Tunable Parameters**

Many block parameters are tunable. A *tunable parameter* is a parameter whose value can change while Simulink is executing a model. For example, the gain parameter of the Gain block is tunable. You can alter the block’s gain while a simulation is running. If a parameter is not tunable and the simulation is running, Simulink disables the dialog box control that sets the parameter. Simulink allows you to specify that all parameters in your model are nontunable except for those that you specify. This can speed up execution of large models and enable generation of faster code from your model.

**Block Sample Times**

Every Simulink block is considered to have a sample time, even continuous blocks (e.g., blocks that define continuous states, such as the Integrator block) and blocks that do not define states, such as the Gain block. Discrete blocks allow you to specify their sample times via a Sample Time parameter. Continuous blocks are considered to have an infinitesimal sample time called a continuous sample time. A block that is neither discrete or continuous is said to have an implicit sample time that it inherits from its inputs. The implicit sample time is continuous if any of the block’s inputs are continuous. Otherwise, the implicit sample time is discrete. An implicit discrete sample time is equal to the shortest input sample time if all the input sample times are integer multiples of the shortest time. Otherwise, the implicit sample time is equal to the *fundamental sample time* of the inputs, where the fundamental sample time of a set of sample times is defined as the greatest integer divisor of the set of sample times.

**Custom Blocks**

Simulink allows you to create libraries of custom blocks that you can then use in your models. You can create a custom block either graphically or programmatically. To create a custom block graphically, you draw a block diagram representing the block’s behavior, wrap this diagram in an instance of the Simulink Subsystem block, and provide the block with a parameter dialog, using the Simulink block mask facility. To create a block programmatically, you create an M-file or a MEX-file that contains the block’s system functions. The resulting file is called an S-function. You then associate the S-function with instances of the

Simulink S-Function block in your model. You can add a parameter dialog to your S-Function block by wrapping it in a Subsystem block and adding the parameter dialog to the Subsystem block.

**Systems and Subsystems**

A Simulink block diagram can consist of layers. Each layer is defined by a subsystem. A subsystem is part of the overall block diagram and ideally has no impact on the meaning of the block diagram. Subsystems are provided primarily to help in the organization aspects a block diagram. Subsystem do not define a separate block diagram.

Simulink differentiates between two different types of subsystems virtual and nonvirtual subsystems. The main difference is that nonvirtual subsystems provide the ability to control when the contents of the subsystem are evaluated.

**Flattening the Model Hierarchy**

While preparing a model for execution, Simulink generates internal “systems” that are collections of block methods (equations) that are evaluated together. The semantics of time-based block diagrams doesn’t require creation of these systems. Simulink creates these internal systems as a means to manage the execution of the model. Roughly speaking, there will be one system for the top-level block diagram window which is referred to as the root system, and several lower-level system derived from the nonvirtual subsystem and other elements within the block diagram. You will see these systems within the Simulink Debugger. The act of creating these “internal” systems is often referred to as flattening the model hierarchy.

**Conditionally Executed Subsystems**

You can create conditionally executed subsystems that are executed only when a transition occurs on a triggering, function-call, action, or enabling input. Conditionally executed subsystems are atomic. Unconditionally executed subsystems are virtual by default. You can, however, designate an unconditionally executed subsystem as atomic. This is useful if you need to ensure that the equations defined by a subsystem are evaluated “together” as a unit.

**Signals**

Simulink uses the term *signal* to refer to a time varying quantity that has values at all points in time. Simulink allows you to specify a wide range of signal attributes, including signal name, data type (e.g., 8-bit, 16-bit, or 32-bit integer), numeric type (real or complex), and dimensionality (one-dimensional or two-dimensional array). Many blocks can accept or output signals of any data or numeric type and dimensionality. Others impose restrictions on the

attributes of the signals they can handle. On the block diagram, you will find that the signals are represented with lines that have an arrow head. The source of the signal corresponds to the block that writes to the signal during evaluation of its block methods (equations). The

destinations of the signal are blocks that read the signal during the evaluation of its block methods (equations). A good analogy of the meaning of a signal is to consider a classroom. The teacher is the one responsible for writing on the white board and the students read what is written on the white board when they choose to. This is also true of Simulink signals, a reader of the signal (a block method) can choose to read the signal as frequently or infrequently as so desired.