Matlab and Simulink are the main computational tools used to design, analyze, and simulate control systems. Simulink is a graphical simulation tool that is based on the block diagram concept.

The main block is the integrator:

\[ y(t) = y_0 + \int_0^t u(\tau) \, d\tau \]

where the initial condition \( y(0) = y_0 \) is specified inside the block.

This relation is equivalent to:
\[
\begin{cases} 
\dot{y}(t) = u(t) \\
\quad y(0) = y_0 
\end{cases}
\]

Ex) \( \dot{x} = -5x + 6u \), \( x(0) = x_0 \)

The block diagram for this system is:

Simulink has "sum" and "gain" blocks in the "commonly used blocks" folder. These can be used to construct this diagram.

Note: 1) In Simulink the integrator is denoted \( \int \)

This is related to the Laplace transform which we have yet to cover.

2) Simulink also has a derivative block \( \frac{dy}{dt} \)

You could construct a block diagram for \( \dot{x} = -5x + 6u \) using this block rather than the integrator. Don't do this. Numerically it is much better to construct your diagrams using integrator blocks.
In general, if you have a (linear or nonlinear) state-space system with \( n \) states then you can construct your diagram with \( n \) integrator blocks. Alternatively, you can use matrix-valued gain blocks, etc. to construct block diagrams for \( n \)-state systems.

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

where \( x, u, y \in \mathbb{R}^n \), \( A, B, C, D \in \mathbb{R}^{n \times n} \), \( n \times 1 \) real matrix

In class we'll briefly review how to construct these diagrams in Simulink. In Matlab you can open the Simulink browser by typing "Simulink". The Simulink browser has a button to open a new model. The browser also has many libraries (i.e. folders) with different blocks. As a starting point the key folders are:

1. Commonly Used Blocks: Gain, Integrator, Sum
2. Sources: Clock, Constant, Step, Sine Wave
3. Sinks: To Workspace

Search through the other folders for useful blocks.
It is important to have a basic understanding of how Simulink solves the differential equation.

Consider the simple, scalar differential equation (nonlinear)

\[
\dot{x} = f(x, u) \quad \text{where the input } u(t) \text{ is specified (known) for } t \geq 0.
\]

The derivative is approximated by

\[
x(t+\Delta t) \approx x(t) + \frac{x(t+\Delta t) - x(t)}{\Delta t}
\]

For \( \Delta t \) "small".

Thus,

\[
\frac{x(t+\Delta t) - x(t)}{\Delta t} \approx f(x(t), u(t))
\]

\[
\Rightarrow x(t+\Delta t) \approx x(t) + \Delta t \cdot f(x(t), u(t))
\]

Given \( x_0 = x_0 \) and \( u_0 \), we can approximately compute

\[
x(\Delta t) \approx x_0 + \Delta t \cdot f(x_0, u_0)
\]

Next, using this \( x(\Delta t) \) and \( u(\Delta t) \) we can compute:

\[
x(2\Delta t) \approx x(\Delta t) + \Delta t \cdot f(x(\Delta t), u(\Delta t))
\]

We can continue stepping ahead in time using this iteration:

\[
x(k\Delta t) \approx x(k\Delta t) + \Delta t \cdot f(x(k\Delta t), u(k\Delta t))
\]

This gives an approximate numerical solution to \( x(t) \).

This algorithm is known as Euler integration.

Note that the "step" size \( \Delta t \) is assumed to be fixed.

Simulink uses more sophisticated and integration solvers that change the step size, i.e., they are variable step. These variable-step solvers are faster and more accurate. [See Math 5485 for more details]
The key point is that Simulink only approximately solves the ODE. For many problems (most systems in this class) the approximate solution will be of sufficient accuracy. However, if you suspect the Simulink solution is incorrect or sufficiently accurate (e.g., the solution looks "jagged") then you can modify the simulation settings.

In the Simulink diagram, the settings are found in the menu Simulation → Configuration Parameters... → Solver

The maximum step size is auto-generated. The default is \((\text{StepTime} - \text{StartTime}) / 50\). Reducing the max step size will typically improve the accuracy at the expense of longer simulation times. You can also try to reduce the Absolute and Relative tolerance to improve the accuracy.

One final point is that Model-based design (MBD) is becoming the standard in industry. This reduces design time.

---

**Old Way**

Matlab/Simulink

For control design and analysis

\[ \downarrow \text{Convert Simulink to production code} \]

C, C++, etc. implementation of control law

\[ \downarrow \text{Compile Code} \]

Production Processor

**MBD**

Simulink Model

Auto generate code (e.g., Real-time Workshop)

C, C++, Compiler

Production Processor