Algebraic, Differential, & Difference Equations and Transfer Functions

sT-Embed Training

Ric Kolk
Altair Engineering
rkolk@altair.com

Topics:

- Algebraic Equations
  - Static Explicit
  - Static Implicit, using the sT-Embed built in solver

- Differential Equations
  - Specifying Initial Conditions
  - Modeling Linear Differential Equations
  - Using an integrator to differentiate
  - Creating a rate limiter

- Difference Equations
  - Unit Delay, Sample Hold, Pulse Train Blocks
  - Modeling Linear Difference Equations

- Transfer functions
  - Continuous
  - Discrete
  - Continuous to Discrete Transformation

- sT-Embed Filter Design Option
  - IIR, FIR
  - Continuous, Discrete

Algebraic Equations

Static Explicit Equations

Static Explicit Equations: Equations of the form y=Y(x1, x2, x3, …).

input variables = x1, x2, x3, …

output variable(s) = y

linear or Nonlinear relationship =Y(x)

Can be modeled & solved using simple arithmetic, Boolean, and other blocks

Example: Incompressible fluid flow through a restriction

$$Q = C_d A \sqrt{\frac{2 g_c \Delta P}{\rho}}$$

$Q = flowrate, ft^3/s$

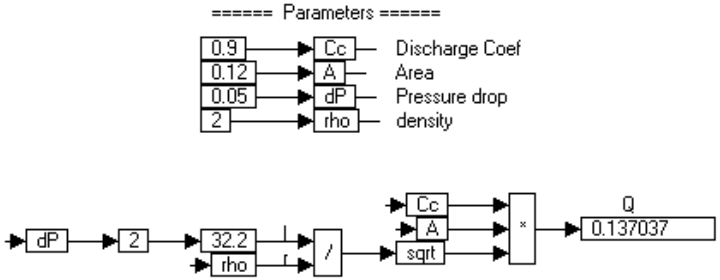
$C_d = dischargecoef, unitless$

$A = area, ft^2$

$\Delta P = pressuredrop, lb_f / ft^2$

$\rho = density, lb_m / ft^3$

$g_c = gravconversion\,factor = 32.2$



Incompressible Fluid Flow Example

Static Implicit Equations - Basics

Static Implicit Equations: Equations of the form y=Y(y).

input variables = y
output variable(s) = y
linear or Nonlinear relationship =Y(y)

In these equations, the output variable is also an input variable. Linear implicit equations are simple to solve analytically, nonlinear equations generally require root finding.

For both linear and nonlinear, the built in sT-Embed Newton Raphson Optimizer and two sT-Embed blocks are used for their solution;

Unknown Block: in the Block/Optimization menu – represents an unknown value to be determined by the optimization.

Constraint Block: in the Block/Optimization menu – represents a constraint with a value to be driven to 0, the constraint equation must be written in “error” form with the error signal applied to the constraint.

The unknown block works in conjunction with constraint blocks to solve equations for unknowns using Newton-Raphson iteration. For each unknown, there should be a constraint block that is fed directly or indirectly by the unknown.

The maximum iteration count, error tolerance, and perturbation are established under the Implicit Solver tab in the dialog box for the System > System Properties command.

Static Implicit Equations - Linear

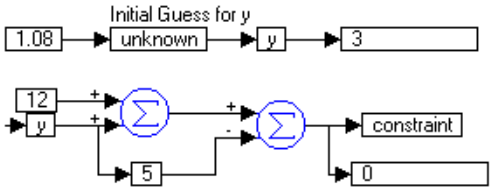
When the “unknown” and “constraint” blocks are present, sT-Embed will ask if you want to use the built in Newton Raphson “implicit solver”, you accept this.

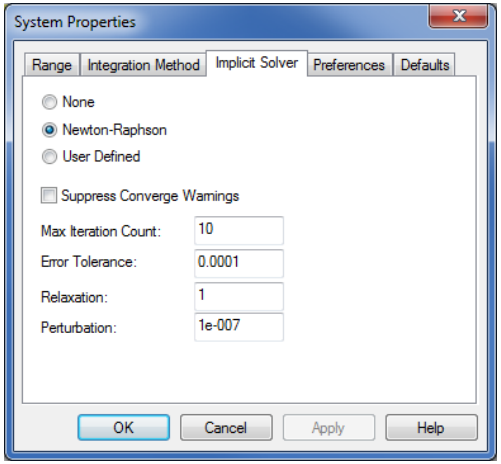
The implicit solver does not require the simulation to transition in time, it makes its calculations at a time interval within a sT-Embed “Time Step”

The setup for the implicit solver is located under “System/System Properties/Implicit Solver”

Example – Linear Equation::

$$y + 12 = 5 \cdot y$$



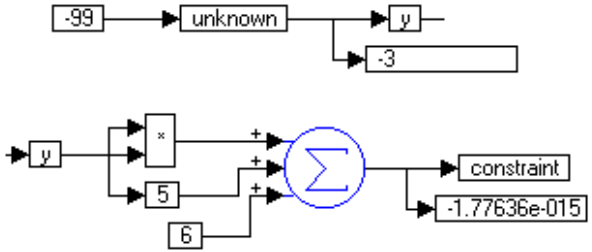


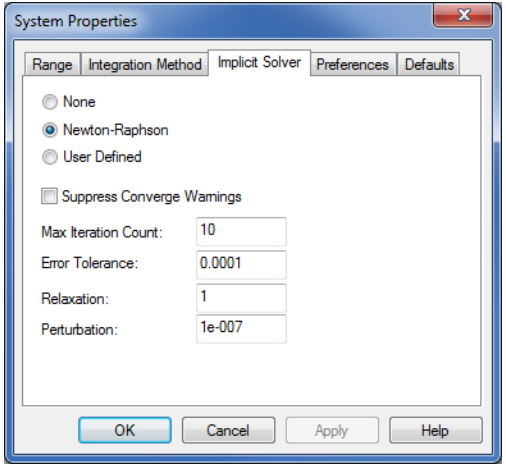
[Static Implicit Linear Equation Example]

Static Implicit Equations - Nonlinear

Example – Nonlinear Equation::

$$y = -\frac{1}{5}(y^2 + 6)$$





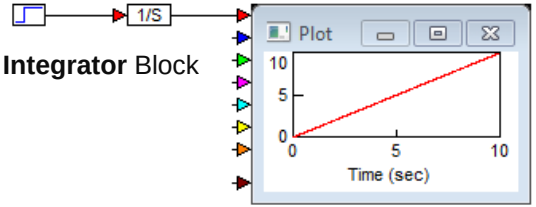
[Static Implicit Nonlinear Equation Example]

Differential Equations

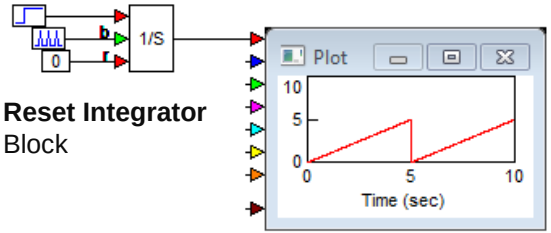
Differential Equations – Integration Operator

sT-Embed uses the “1/s” operator to represent time integration: $\int y(\tau)d\tau = \frac{1}{s}y$

sT-Embed provides three Integration blocks in the (“Blocks/Integration”) menu. Each integrator block and its unit step response behavior is presented below.

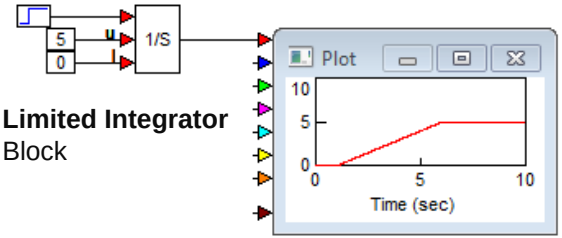


Integrator Block



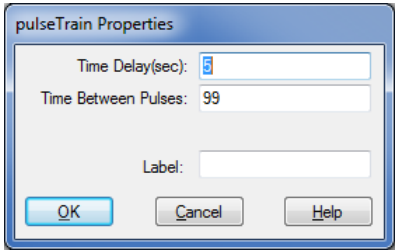
Reset Integrator Block

The integrator output is reset to the reset (“r”) value when the boolean input (“b”) goes High. Here, the boolean signal is created with a “pulseTrain” block configured:



Limited Integrator Block

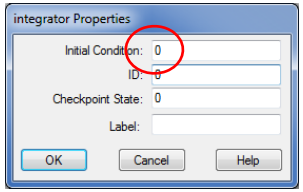
The integrator output is limited to lie between the lower limit value (“l”) and The upper limit value (“u”).



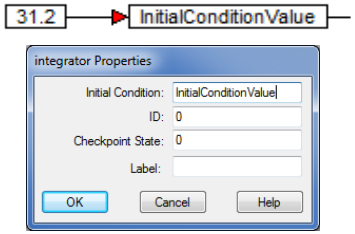
Integration – Specifying the Initial Condition

Integrator initial conditions can be implemented using any of three methods:

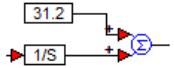
Method 1: “right click” on the integrator to expose its properties menu and enter the “Initial Condition” value



Method 2: Define a global variable with the integrator initial condition. “right click” on the integrator to expose its properties menu and enter the global Variable name as the “Initial Condition” . This method is often used when the Initial conditions must be varied.



Method 3: The initial condition, either a variable or constant value, is added to the integrator output. The “Initial Condition” value in the integrator is set to 0. This method is often used when the initial conditions must be varied.

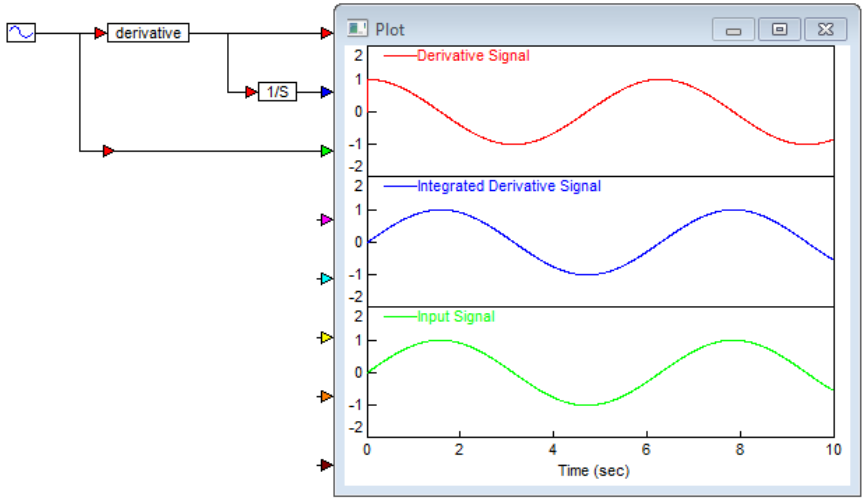


Differential Equations – Derivative Operator

sT-Embed uses the “s” operator to represent time differentiation: $\frac{dy}{dt} = \dot{y} = sy$

sT-Embed provides one differentiation block in the (“Blocks/Integration”) menu.

In the following block diagram, a 1 rad/sec unity amplitude sinusoid input signals is applied to a “derivative” block. The “Input Signal” time history is displayed in the lower plot (“green”0, the “Derivative Signal” time history in the upper plot (“red”), and the “Integrated Derivative Signal” time history in the center plot (“blue”). The “Integrated Derivative Signal” time history is identical to the “Input Signal” time history as expected.

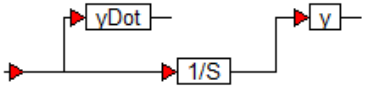


Modeling a Differential Equation (1/2)

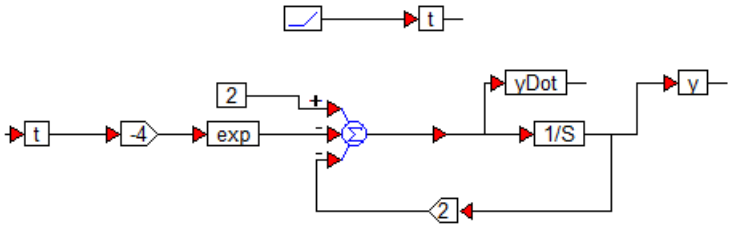
Model & solve the differential equation: $\dot{y} + 2y = 2 - e^{-4t}; y(0) = 1$

Step 1. Identify the order, *n*, of the equation, for this equation, *n* = 1.

Step 2. Serially place and connect *n*- "integrators" ("Blocks/Integration"), use "variables" ("Blocks/Annotation") to label the states (integrator outputs) from right to left beginning with "*y*" , also label the input to the leftmost integrator even though it is not a state.

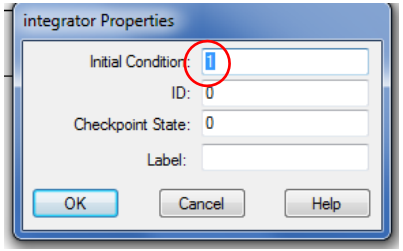


Step 3. Solve the differential equation for the largest derivative of the output variable as a function of the input(s) and states. Incorporate the solution into the step 2 diagram. Use a unit "ramp" for time.

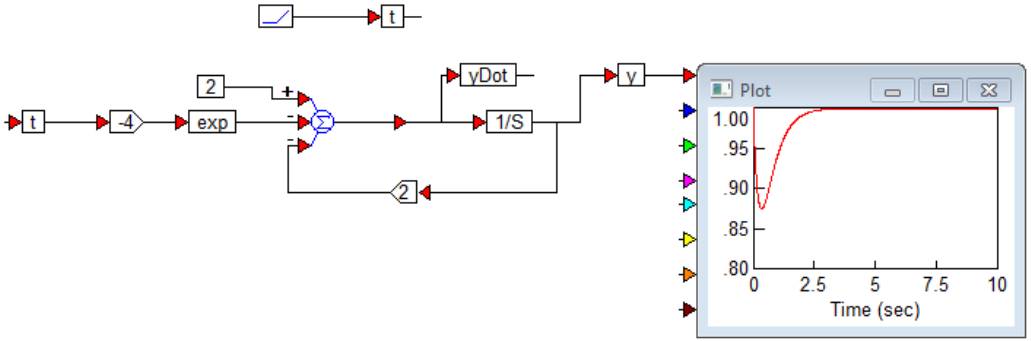


Modeling a Differential Equation (2/2)

Step 4. “right click/Integrator Properties” to set the y(0) =1 initial condition:



Step 5. Connect y to pin 1 of a “plot” block, Click the “Go” button or press “F5” or “System/Go“ to run the simulation.



Modeling a Differential Equation

Linear Differential Equation with Input Dynamics (1/3)

Control system design often deals with linear differential equations. Although most physical system equations are non-linear, it is frequently possible to linearize them over an envelope of operating conditions. The resulting linear differential equations are usually of the same order but with coefficients that vary over the operating conditions.

Model & solve the differential equation (initial conditions = 0) and the input, u, set to a unit step at time = 1.25 seconds.

$$2\dddot{y} + 4\ddot{y} + 6\dot{y} + 8y = \ddot{u} + 2\dot{u} + 3u$$

Since the equation is linear, it can be rewritten as two differential equations based on a newly introduced variable named the state and its derivatives. Normally the state variable is assigned as "x". The two equations are called (1) the state equation and (2) the output equation.

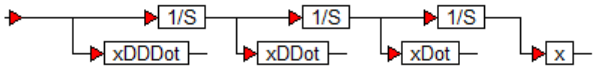
State Equation: $2\ddddot{x} + 4\dddot{x} + 6\ddot{x} + 8\dot{x} = u$

Output Equation: $y = \ddot{x} + 2\dot{x} + 3x$

The steps to create the block diagram model introduced previously are slightly **generalized**.

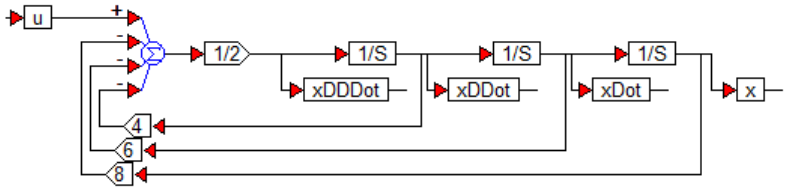
Step 1. Identify the order, *n*, of the **state** equation, for this equation, *n* = 3.

Step 2. Serially place and connect *n*- "integrators" ("Blocks/Integration"), use "variables" ("Blocks/Annotation") to label the states (integrator outputs) from right to left beginning with "***x***" . also label the input to the leftmost integrator even though it is not a state.

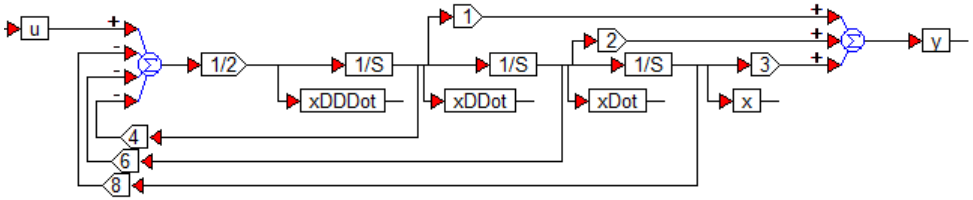


Linear Differential Equation with Input Dynamics (2/3)

Step 3. Solve the **state** equation for the largest derivative **of the state** variable as a function of the input(s) and states. Incorporate the solution into the step 2 diagram.



Step 3a. Incorporate the output equation into the step 3 diagram.

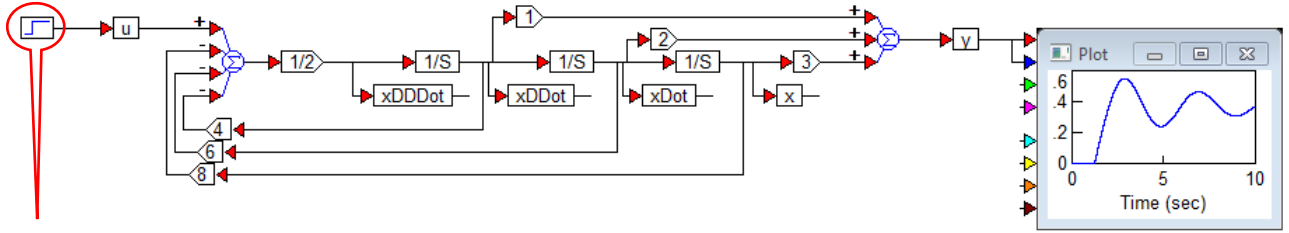


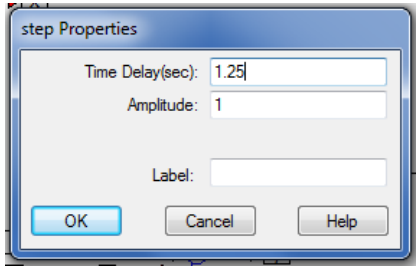
Step 4. "right click/Integrator Properties" to set the initial condition, note that any output initial conditions will need to be transformed to state initial conditions.

[Modeling a Differential Equation with Input Dynamics]

Linear Differential Equation with Input Dynamics (3/3)

Step 5. Configure the “step” block (“Blocks/Signal Producer”) with a “Time Delay (sec)” = 1.25 seconds. Connect y to pin 1 of a “plot” block, Click the “Go” button or press “F5” or “System/Go“ to run the simulation.





Van der Pol Oscillator & Strip Charts

The Van der Pol oscillator obeys the second order differential algebraic equation $\frac{d^2x}{dt^2} - u(1-x^2)\frac{dx}{dt} + x = 0$

Where:

x = position

$\frac{dx}{dt}$ = xDot = velocity

$\frac{d^2x}{dt^2}$ = xDDot = acceleration

u = damping coefficient

x(0) = 1 = Initial position

We will use sT-Embed to model the oscillator and plot its behavior while varying the damping, *u*, between 0.01 to 4.

Van der Pol Oscillator Example & Strip Chart

Difference Equations

Difference Equations – Unit Delay Operator

Difference equations are based on sequences instead of signals. A sequence takes on values at discrete instances on time. Often the sequence time interval is constant and called the “discrete update time”, Δt.

Continuous time, t, is related to the discrete update time by the relationship: $t = k\Delta t; k = 0,1,2,3..$ Where: *k* is the sequence index

Sequences with constant discrete update times are written using only the sequence index; y_k

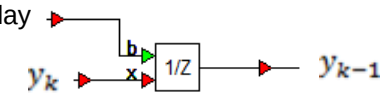
The “*z*” operator is used to represent one unit of time advance (one unit of discrete update time); $y_{k+1} = zy_k$

Similarly, the “1/z” “operator is used to represent one unit of time delay. $y_{k-1} = \frac{1}{z}y_k$

sT-Embed uses the “unitDelay” block (“Blocks/Time Delay”) to model the unit delay. The “unitDelay” block can be used for both constant and variable discrete update times.

The sT-Embed “unitDelay” block, below, accepts two inputs; a boolean input “b”, ,used to trigger the delay and a sequence or signal input, “x”;

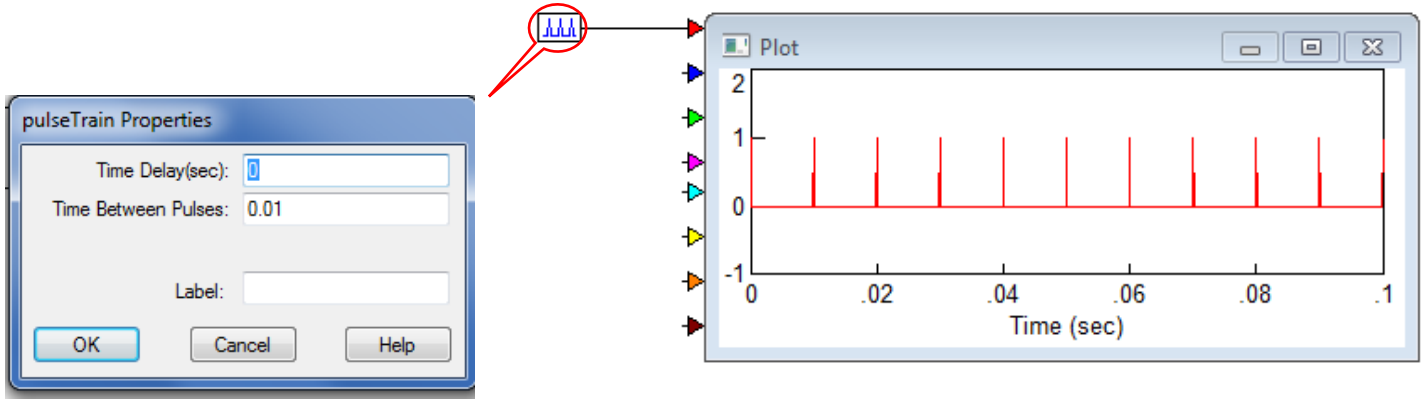
Pulse input used to trigger the unitDelay



Difference Equations – Unit Delay & Pulse Train (1/2)

When the discrete update time is constant, the sT-Embed “pulseTrain” operator (“Blocks/Signal Producers”) is used to trigger the “unitDelay”. The behavior of a “PulseTrain” configured with a 0.01 second “Time Between Pulses” is shown below:

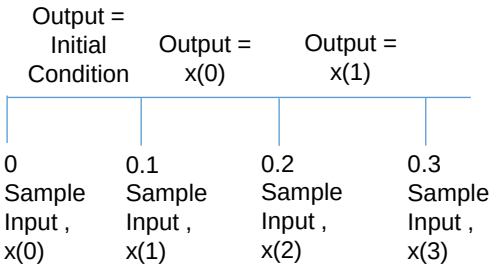
.

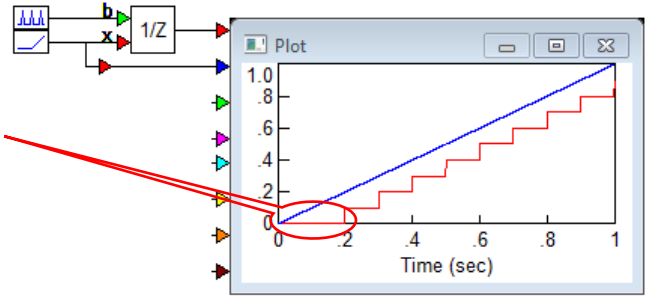


Difference Equations – Unit Delay & Pulse Train (2/2)

The following block diagram illustrates the behavior of the “unitDelay” block operating on a unit “ramp” input and triggered by a “pulseTrain” configured with a 0.1 second “Time Between Pulses”. The “unitDelay” is configured with a 0 valued initial condition.

.



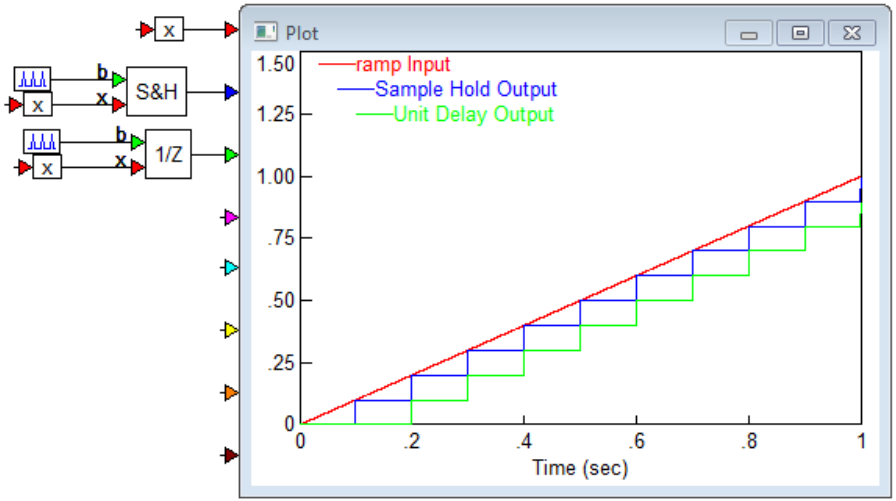


Difference Equations – Unit Delay & Sample Hold

The “sampleHold” block (“Blocks/Nonlinear”) is similar in behavior to the “unitDelay” block except it does not apply a delay to the input signal or sequence.

The following block diagram illustrates the behavior of the “sampleHold” and “unitDelay” blocks when a unit ramp signal is input to both blocks and triggering is performed using a “pulseTrain” configured with a “Time Between Pulses” = 0.1 seconds.

.



Unit Delay and Sample Hold Example

Linear Difference Equation with Input Dynamics (1/3)

Model & solve the difference equation (initial conditions = 0) and the input, u, set to a unit step at time = 1.25 seconds. The digital update time is 0.01 seconds.

$$y_{k+2} + .2y_{k+1} + .8y_k = 2u_{k+2} + .2u_{k+1} + u_k$$

Since the equation is linear, it can be rewritten as two difference equations based on a newly introduced variable named the state and its derivatives. Normally the state variable is assigned as "x". The two equations are called (1) the state equation and (2) the output equation.

State Equation: $x_{k+2} + .2x_{k+1} + .8x_k = u_k$

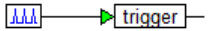
Output Equation: $y_k = 2x_{k+2} + .2x_{k+1} + x_k$

The steps to create the block diagram model are identical to those used for the Linear Differential Equation with Input Dynamics except the discrete update time is defined in step 1a.

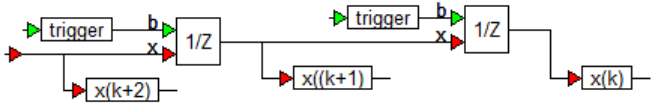
.

Step 1. Identify the order, *n*, of the **state** equation, for this equation, *n* = 3.

Step 1a. Create the discrete update time as the variable "trigger" defined by a "pulseTrain" block configured with the "Time Between Pulses" = 0.01 seconds.

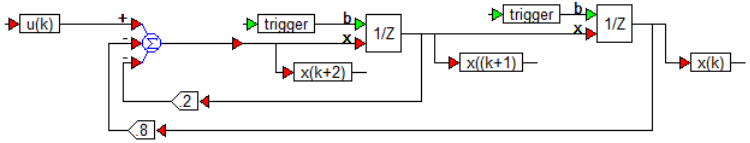


Step 2. Serially place and connect *n*- "unitDelays" ("Blocks/Integration"), use "variables" ("Blocks/Annotation") to label the states (unit delay outputs) from right to left beginning with "***x(k)***" . also label the input to the leftmost unitDelay even though it is not a state.

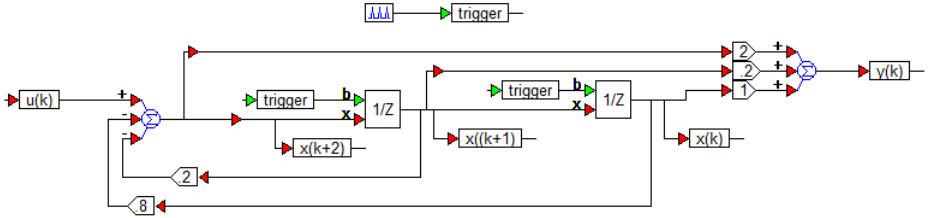


Linear Difference Equation with Input Dynamics (2/3)

Step 3. Solve the state equation for the largest time advance of the state variable as a function of the input(s) and states. Incorporate the solution into the step 2 diagram.



Step 3a. Incorporate the output equation into the step 3 diagram.

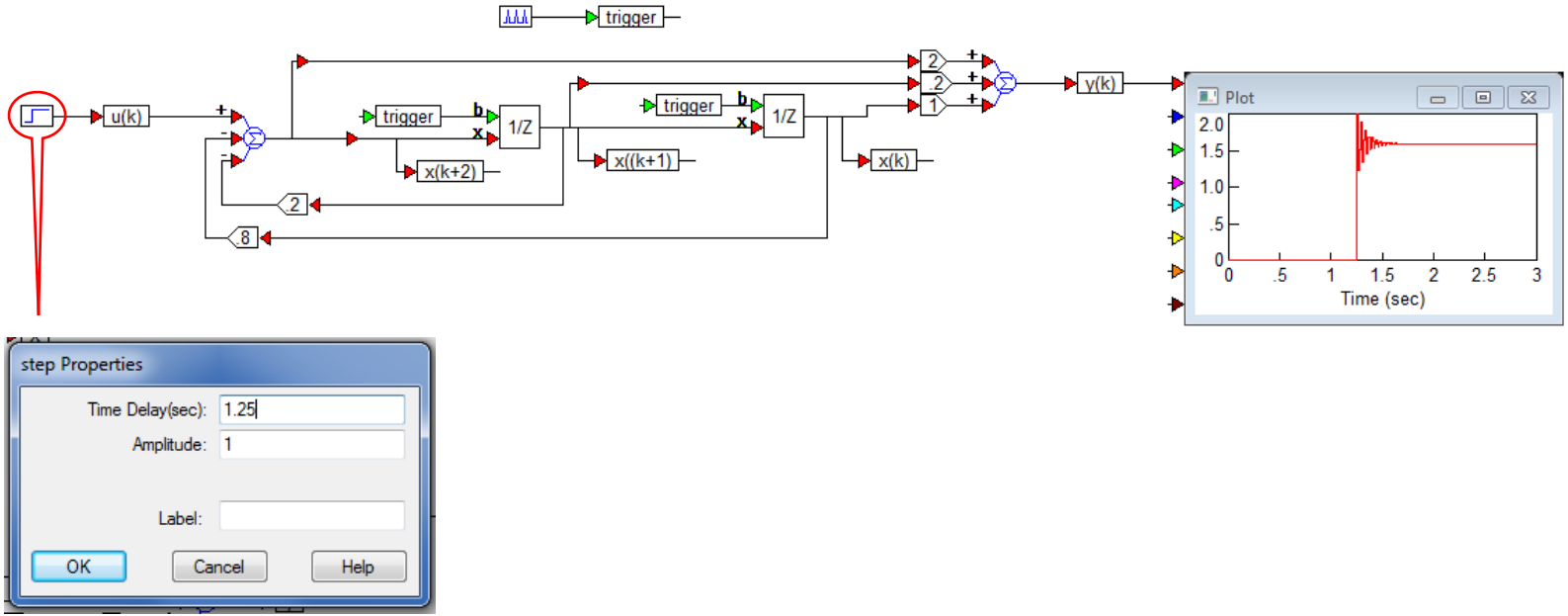


Step 4. "right click/unitDelay Properties" to set the initial condition, note that any output initial conditions will need to be transformed to state initial conditions.

Difference Equation Example

Linear Difference Equation with Input Dynamics (3/3)

Step 5. Configure the “step” block (“Blocks/Signal Producer”) with a “Time Delay (sec)” = 1.25 seconds. Connect y(k) to pin 1 of a “plot” block, Click the “Go” button or press “F5” or “System/Go“ to run the simulation.



Transfer Functions

To here

Transfer Functions

Linear Differential and Difference Equations can be represented by transfer functions. Transfer functions provide an efficient way or representing a Plant, Controller, or Control System. A transfer function is a ratio of the systems output/input expressed as the ratio of two polynomials, a numerator and a denominator, represented as coefficients of descending powers of either "s" (continuous) or "z" (discrete).

Example: Continuous System, *u* = input, *x* = output

..

$$\ddot{x} + 2\dot{x} + 3x = 7\dot{u} + 3u$$
$$s^2x + 2sx + 3x = 7su + 3u$$
$$x(s^2 + 2s + 3) = u(7s + 3)$$
$$\frac{x}{u} \equiv T(s) = \frac{7s + 3}{s^2 + 2s + 3}$$

Transfer Function

General Form of a Transfer Function:

Continuous $T(s) = k\frac{num(s)}{den(s)}$

Discrete $T(z) = k\frac{num(z)}{den(z)}$

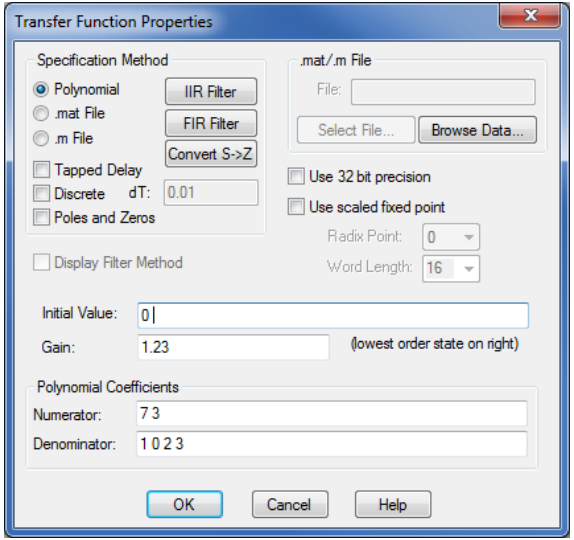
$$T() = gain\frac{numerator\ polynomial}{denominator\ polynomial}$$

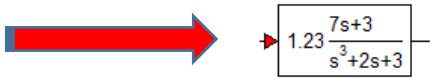
Continuous Transfer Functions

Continuous Transfer functions are defined using the sT-Embed “transferFunction” block located in “Blocks/Linear System”

Example: $T(s) = \frac{1.23(7s+3)}{s^3+2s+3}$

.





Transfer Function Example

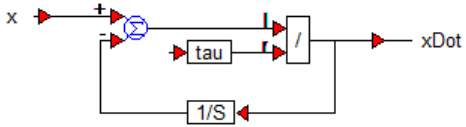
Using an Integrator to Differentiate

Since it is numerically more stable and accurate to solve differential equations using numerical integration methods rather than differentiation., differential equation models will be constructed using integrators.

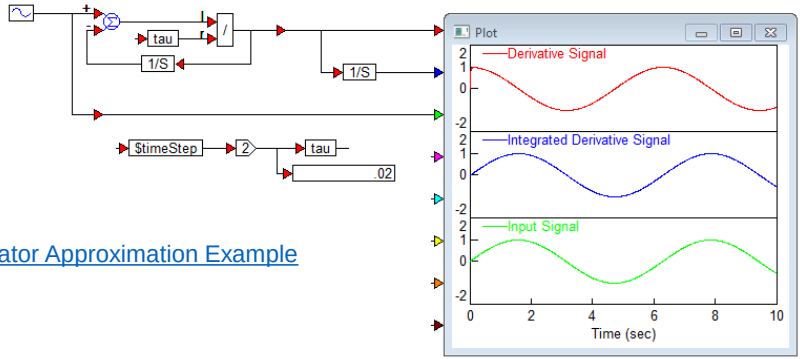
In situations where differentiation is necessary, the following approximation to differentiation can be used (right).

$$s = \frac{s}{1} \cong \frac{s}{\tau s + 1} \quad as\ \tau \to 0$$

The "Approximate Derivative" block diagram model (using "tau" in place of "τ") is written as (right). For stability, "tau" should be set as tau >= "Time Step" * 2



In the following block diagram, a 1 rad/sec unity amplitude sinusoid input signals is applied to the "Approximate Derivative" model. "derivative" block. Tau is set equal to 2*Time Step. The "Input Signal" time history is displayed in the lower plot ("green"0, the "Derivative Signal" time history in the upper plot ("red"), and the "Integrated Derivative Signal" time history in the center plot ("blue"). The "Integrated Derivative Signal" time history is identical to the "Input Signal" time history as expected.



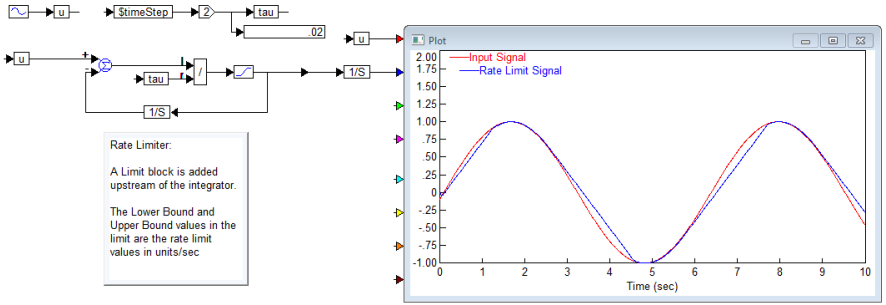
Differentiator Approximation Example

How to create a Rate Limiter

We can extend the “Approximate Derivative” block developed previously to create a rate limiter block.

A limit block (“Blocks/nonlinear”) is added upstream of the integrator to implement the rate limit action. The Lower Bound and Upper Bound values are set the rate limit values in units/sec.

In the following example, a unit amplitude sin wave is passed through a rate limit set to +/- 0.8units/sec



Rate Limiter Example

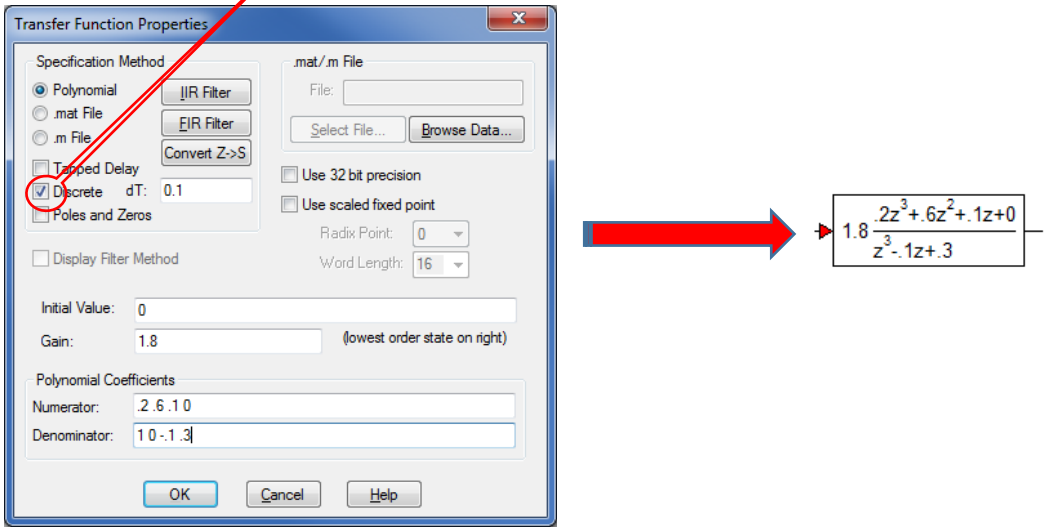
DiscreteTransfer Functions

Discrete Transfer functions are defined using the sT-Embed “transferFunction” block located in “Blocks/Linear System”

.Example: DiscreteTransfer Function with digital update time = .1 seconds $T(z) = \frac{1.8(.2z^3 + .6z^2 + .1z)}{z^3 - .1z + .3}$

.

Make sure you check “Discrete” and enter the dT (discrete update time)



Discrete Transfer Function Example

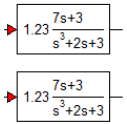
Continuous to DiscreteTransfer Function Conversion

The “transferFunction” block can be used to convert continuous transfer functions to discrete form.

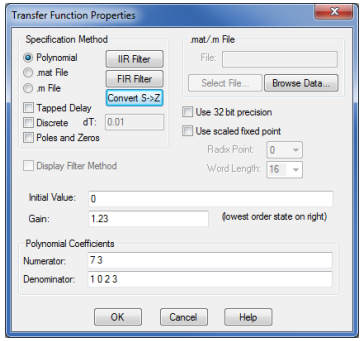
.Example: Convert T(s) to T(z) using a discrete update time = .01 sec and compare the unit step responses using Simulation Time Step = .001 sec and End = 10 sec

$$T(s) = \frac{1.23(7s+3)}{s^3+2s+3}$$

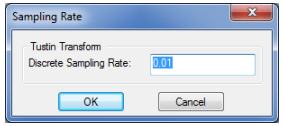
Step 1. Copy and Paste the T(s) transfer function so there are two copies.



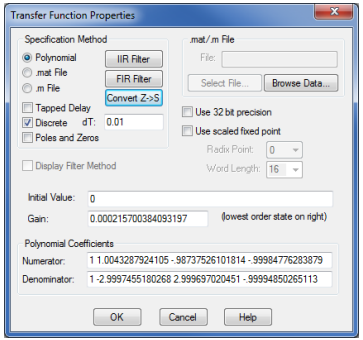
Step 2. right click on the lower “transferFunction” block to reveal “Transfer Function Properties”



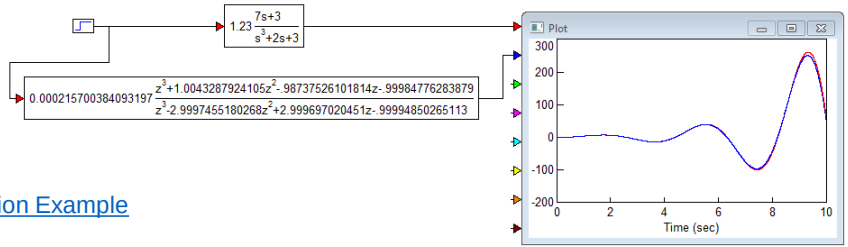
Step 3. click “Convert S->Z”, enter the “Discrete Update Time” value in seconds.



Step 4. click “OK” and see the discrete transfer function , click “OK” again



Connect a “Step” input (“Blocks/Signal Producers”) to both transfer functions and plot their outputs using Time Step = .001 sec and End = 10.



Continuous to Discrete Conversion Example

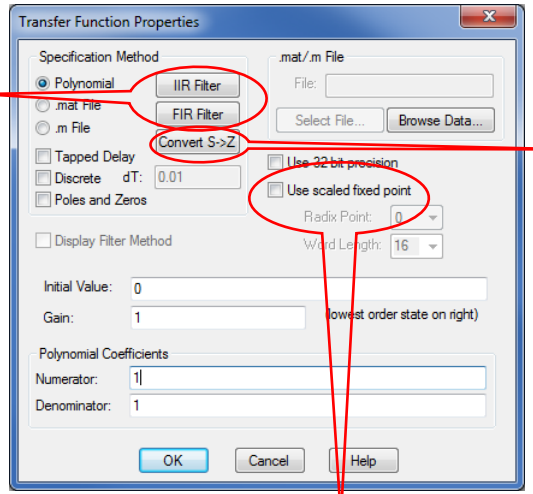
Filter Design Option

Filter Design Option - Features

The Filter Design option is located in the “Blocks/Linear Systems/Transfer Function” Block. Two filter types are supported:

IIR = Infinite Impulse Response (filter with feedback)
FIR = Finite Impulse Response (sometimes called a tapped delay filter)

Filters can be configured as:
Low Pass
High Pass
Band Pass
Band Reject



Analog (S-Domain) filters can be converted to Discrete (Z-Domain) filters

Discrete Filters can be converted to Fixed Point for improved Real Time Performance

End of Section

EMBED™
solidThinking