Simulation Examples in LabVIEW

Hans-Petter Halvorsen

Contents

Differential Equations

Hans-Petter Halvorsen

Differential Equations

A general continuous differential equation can be written on this general form:

$$\frac{dx}{dt} = f(t, x)$$

A differential equation or a set of differential equations describes the dynamic behavior of a system

Differential Equations

Differential Equation on general form:

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0$$

Initial condition

Different notation is used:

$$\frac{dy}{dt} = y' = \dot{y}$$

Example:

$$\frac{dy}{dt} = 3y + 2, \qquad y(t_0) = 0$$

Differential equation

Initial condition

ODE – Ordinary Differential Equations

Simulation Examples 1.order System

Hans-Petter Halvorsen

Table of Contents

1. Order System

Differential Equation of a 1. order System:

$$\dot{x} = -ax + bu$$

$$y = x$$

For control systems, u is typically the control value that comes from the controller, e.g., a PID controller.

Typically, in general we use x for internal variables in the process and y for the measured output(s). For larger systems we can have multiple x $(x_1, x_2 \cdots)$ and multiple y $(y_1, y_2 \cdots)$.

To simulate this model in LabVIEW you can e.g., make a discrete version of the model

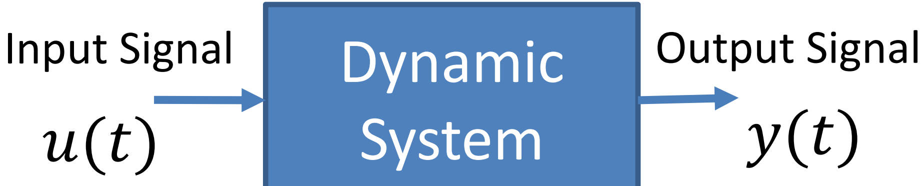
1. Order System

Assume the following general Differential Equation:

$$\dot{y} = -ay + bu$$

or:

$$\boxed{\dot{y} = \frac{1}{T}(-y + Ku)}$$



Where $a = \frac{1}{T}$ and $b = \frac{K}{T}$

Where K is the Gain and T is the Time constant

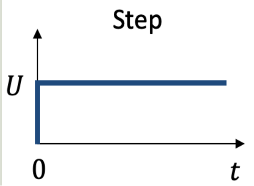
This differential equation represents a 1. order dynamic system

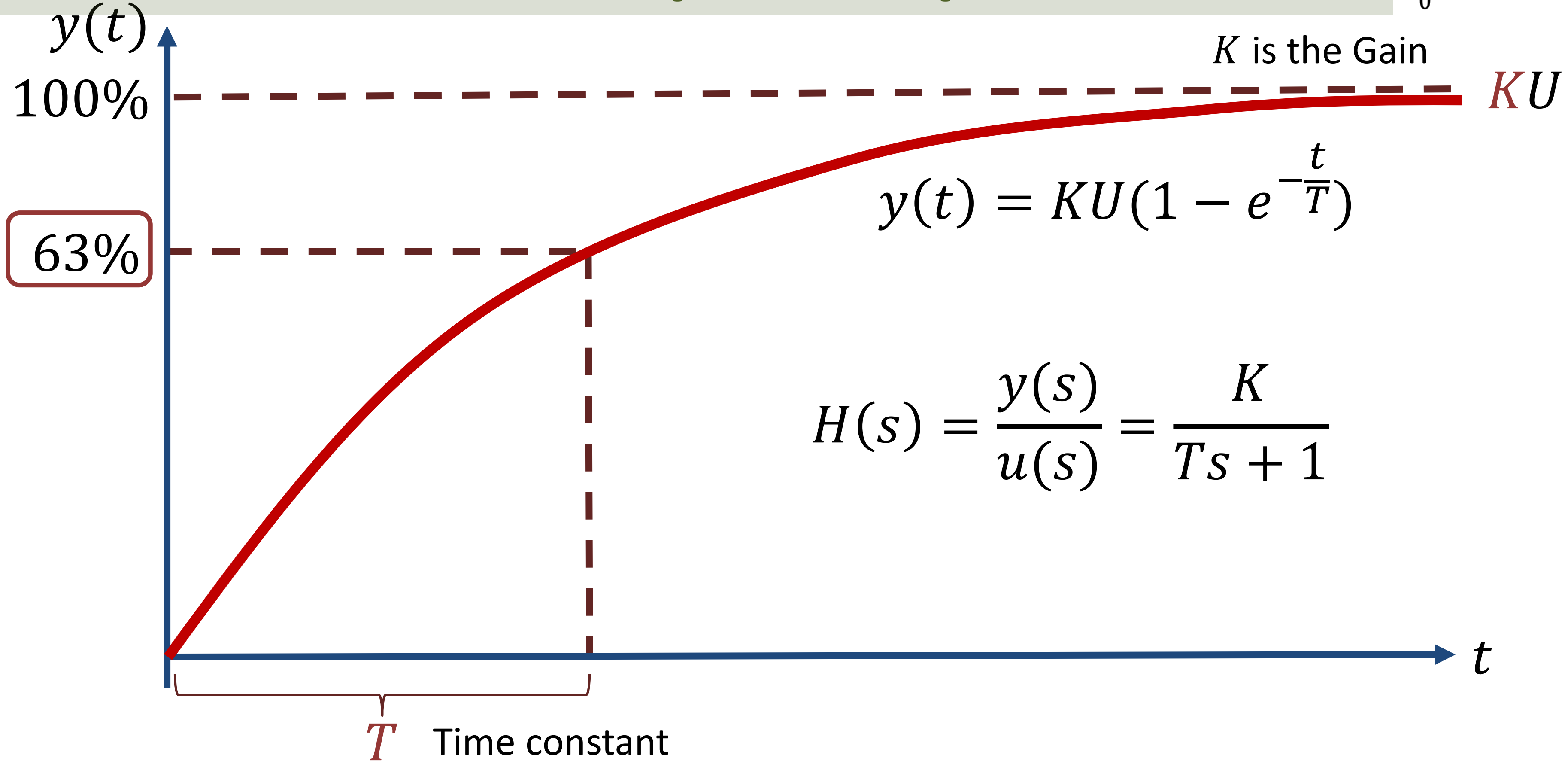
Assume $u(t)$ is a step (U), then we can find that the solution to the differential equation is:

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

(by using Laplace)

1. Order Step Response





$$y(t) = KU\left(1 - e^{-\frac{t}{T}}\right)$$

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{Ts+1}$$

Find Solution for Diff. Equation

Given the 1. order System:

$$\dot{y} = \frac{1}{T}(-y + Ku)$$

Let's use Laplace:

$$sy(s) = -\frac{1}{T}y(s) + \frac{K}{T}u(s)$$

$$y(s)\left[S + \frac{1}{T}\right] = \frac{K}{T}u(s)$$

$$y(s) = \frac{K}{Ts+1}u(s)$$

Then let's assume a Unit Step and use the Laplace Transformation pair $u(t) \Leftrightarrow \frac{U}{s}$

$$y(s) = \frac{K}{Ts+1} \cdot \frac{U}{s} = \frac{K}{(Ts+1)s} \cdot U$$

Then we use the Laplace Transformation pair $\frac{K}{(Ts+1)s} \Leftrightarrow K(1 - e^{-\frac{t}{T}})$ to transform the system back to the time domain. This gives the following solution:

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

The Laplace Transformation pairs can be found in a Table of Laplace Transformations, e.g., https://www.intmath.com/laplace-transformation/table-laplace-transforms.php

Simulation

The 1. order System:

$$\dot{y} = \frac{1}{T}(-y + Ku)$$

Has the following known solution:

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

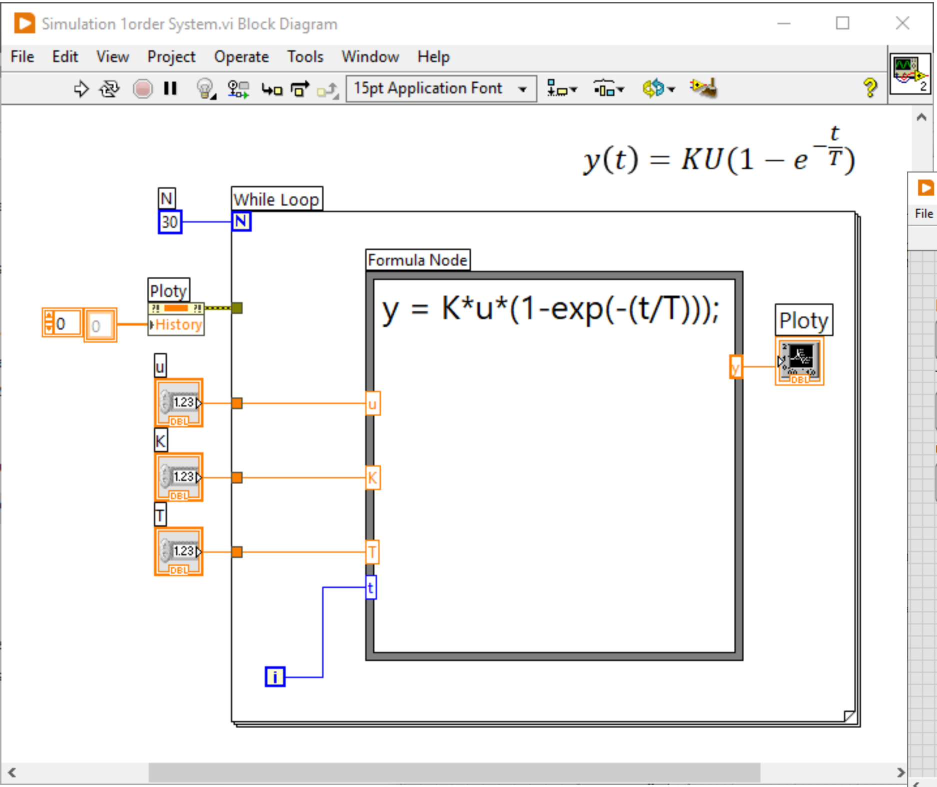
(You can use Laplace to find the solution, see previous page)

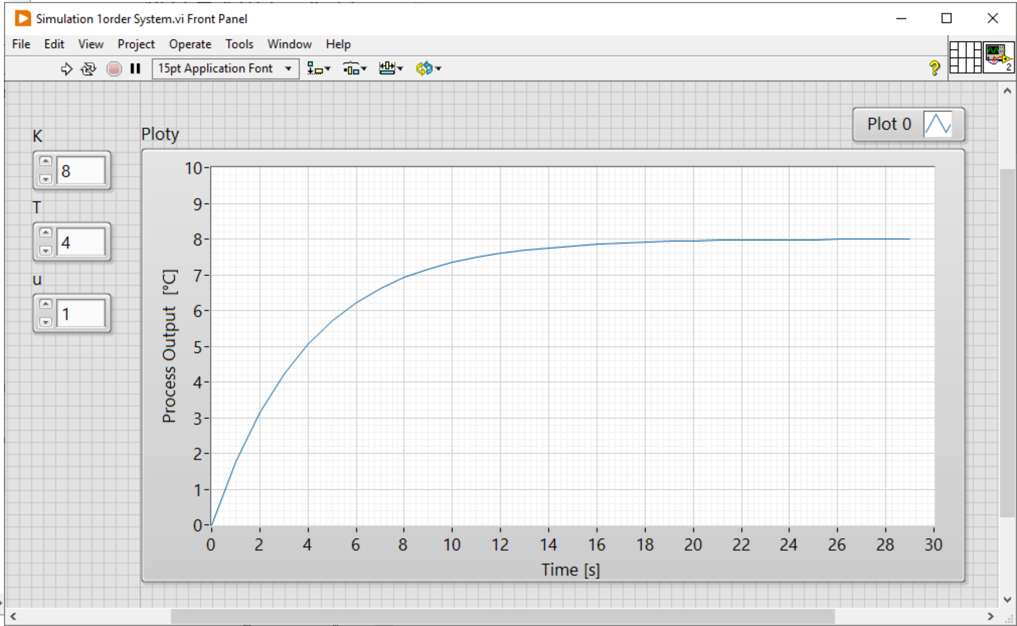
Let's Simulate this system in LabVIEW

Simulation - LabVIEW

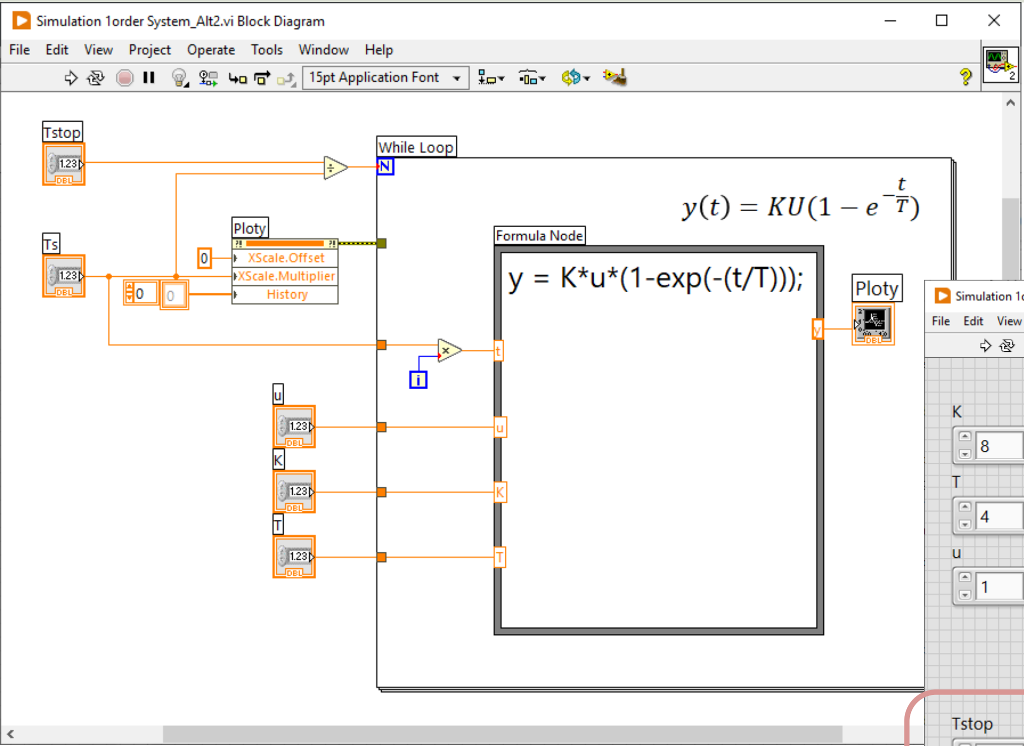
$y(t) = KU(1 - e^{-\frac{t}{T}})$ Let's start with $K = 8$ and $T = 4$ and a step $U = 1$

Here we have implemented the equation inside a **Formula Node**. In that way we can write the equations as we do on paper.

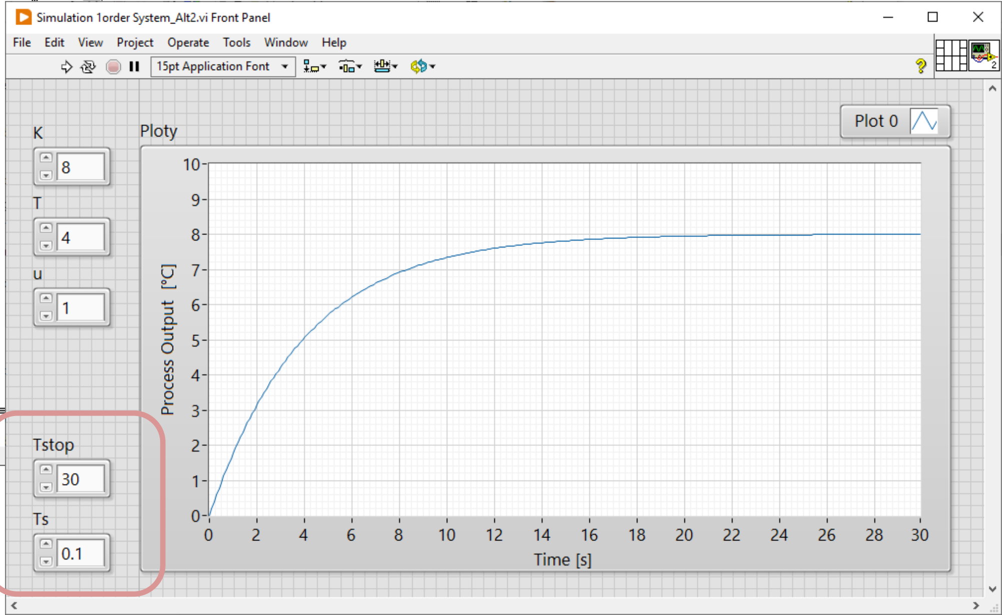




Improved Version



Here we can set the Sampling Time Ts and the simulation time Tstop from the GUI



Discretization

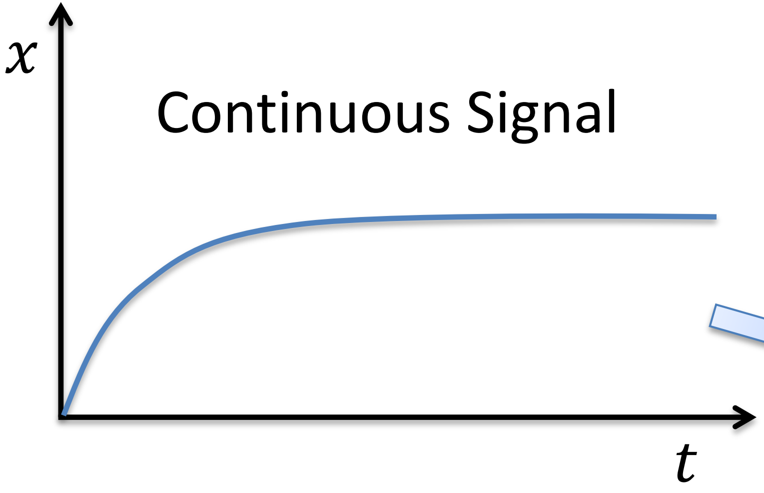
Hans-Petter Halvorsen

Table of Contents

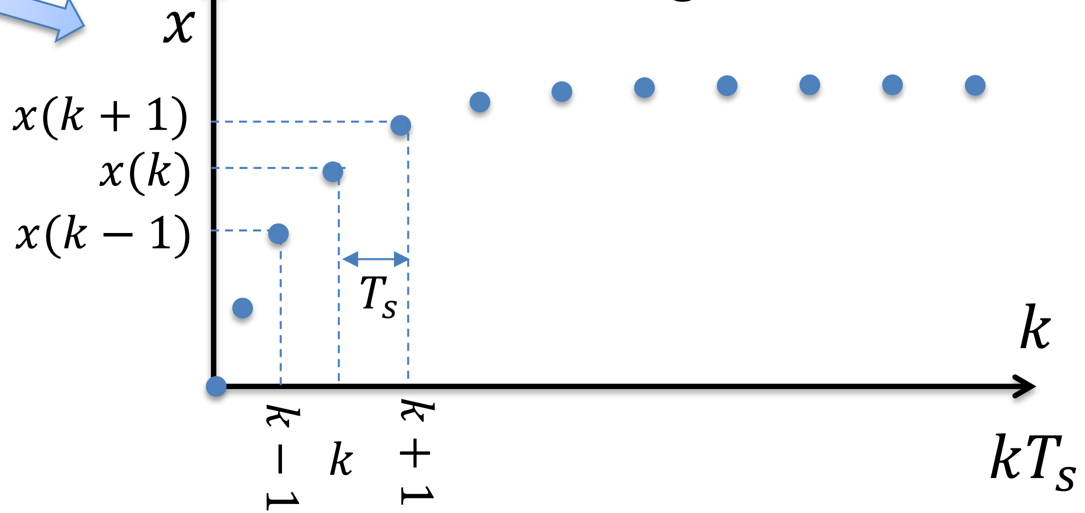
Continuous vs. Discrete Systems

A computer can only deal with discrete signals

Continuous Signal



Discrete Signal



T_s - Sampling Interval

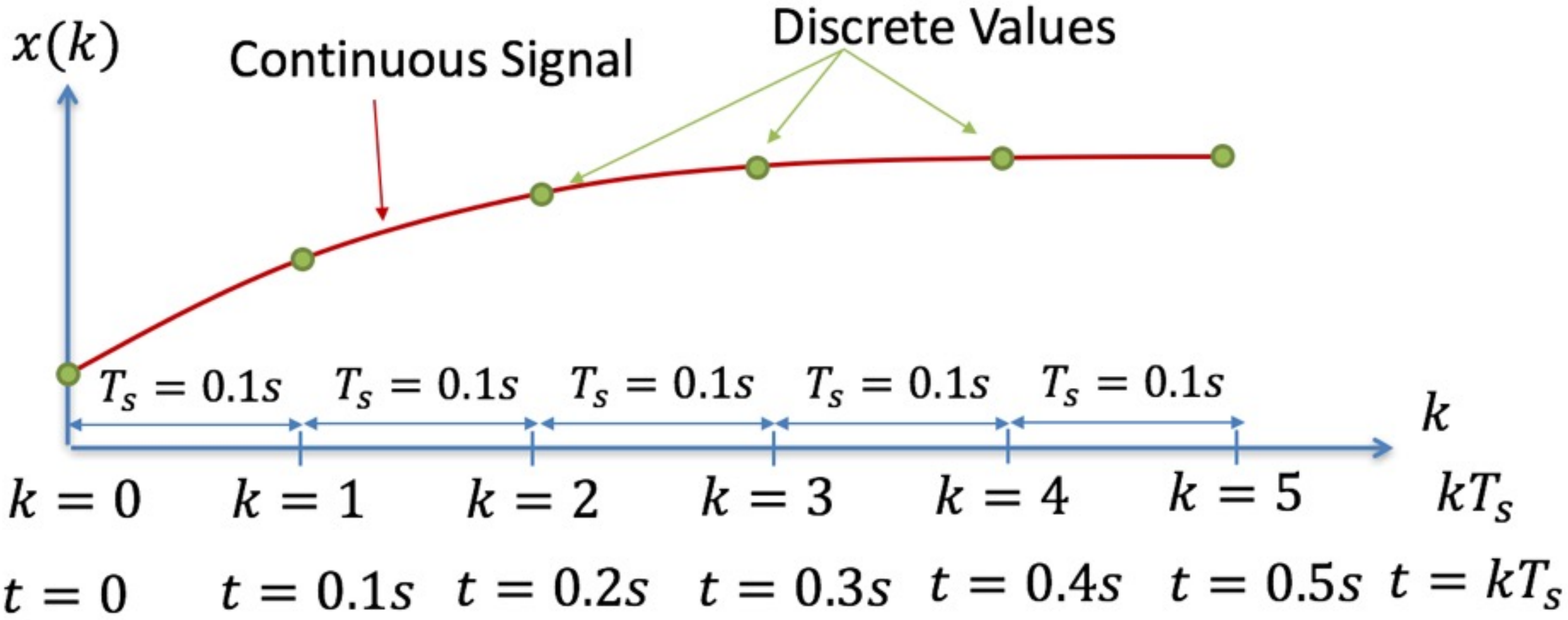
$x(k-1)$ - Previous Value

$x(k)$ - Current Value

$x(k-1)$ - Next Value

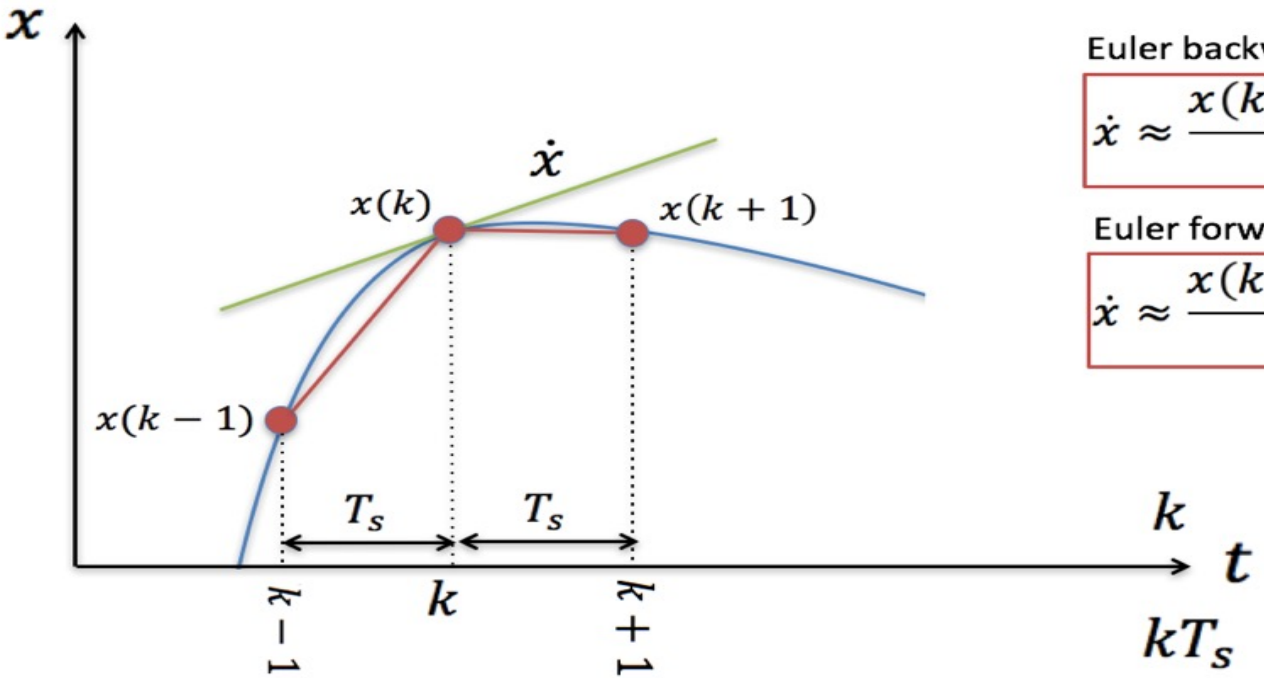
Discrete Data

Below we see a continuous signal vs the discrete signal for a given system with discrete time interval $Ts = 0.1s$. For continuous systems we use t, while for discrete intervals we use k.



Euler Forward method

A simple discretization method is the Euler Forward method



Euler backward method:

$$\boxed{\dot{x} \approx \frac{x(k) - x(k-1)}{T_s}}$$

Euler forward method:

$$\boxed{\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}}$$

Lots of other discretization methods do exists, such as Euler backward, Zero Order Hold (ZOH), Tustin's method, etc.

Discretization

We have a general continuous differential equation:

$$\frac{dx}{dt} = f(t)$$

We can use Euler:

$$\frac{dx}{dt} \approx \frac{x(k+1)-x(k)}{T_s}$$

Then we get:

$$\frac{x(k+1) - x(k)}{T_s} = f(k)$$

This gives the following discrete differential equation:

$$x(k+1) = x(k) + T_s f(k)$$

Discretization

We have the continuous differential equation: $\dot{x} = -ax + bu$

We apply **Euler**: $\dot{x} \approx \frac{x(k+1)-x(k)}{T_s}$

Then we get:

$$\frac{x(k+1) - x(k)}{T_s} = -ax(k) + bu(k)$$

This gives the following discrete differential equation (difference equation):

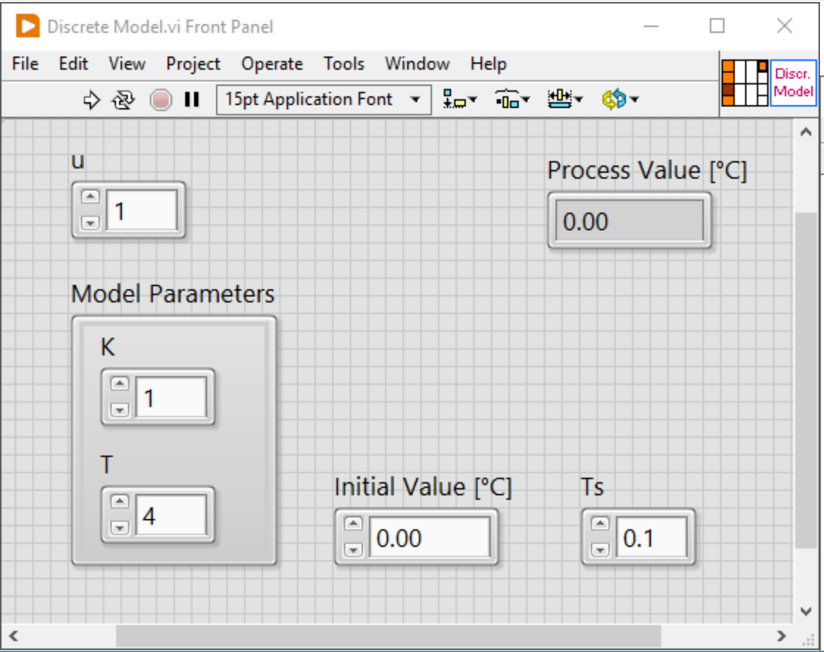
$$\boxed{x(k+1) = (1 - T_s a)x(k) + T_s bu(k)}$$

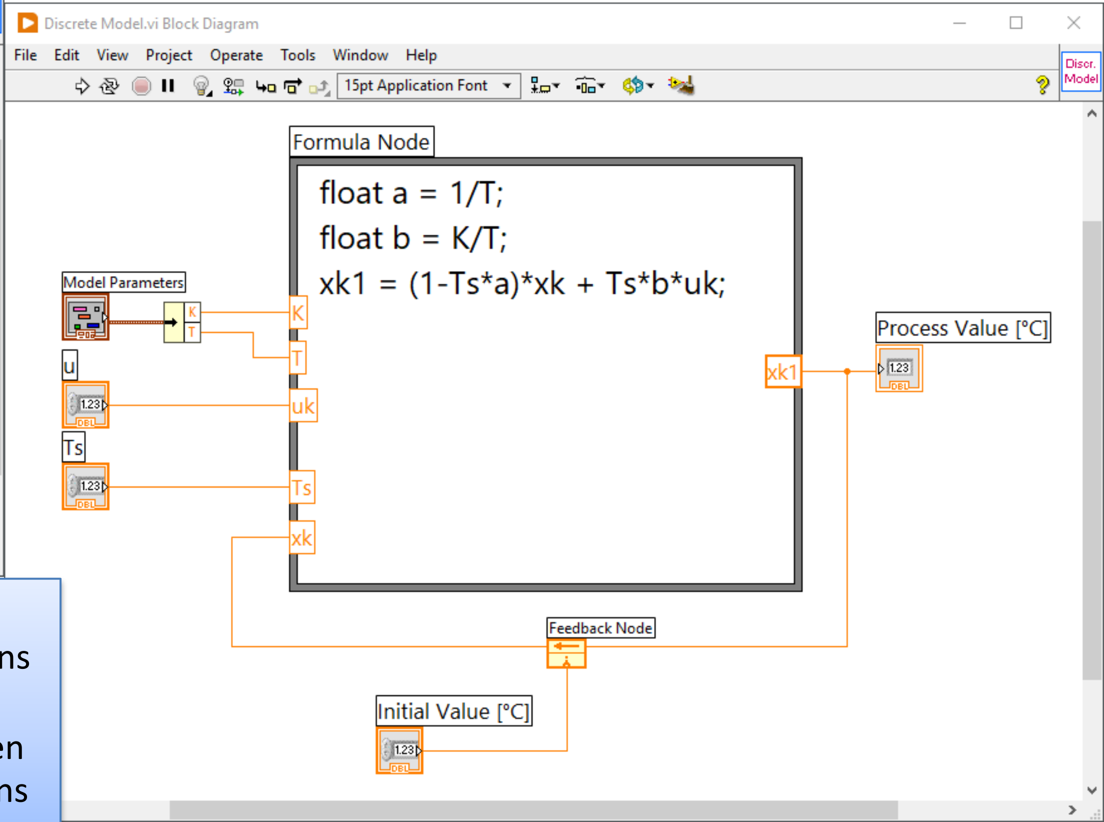
This equation can easily be implemented in any text-based programming language or the Formula Node in LabVIEW

Where $a = \frac{1}{T}$ and $b = \frac{K}{T}$

Discrete Model in LabVIEW

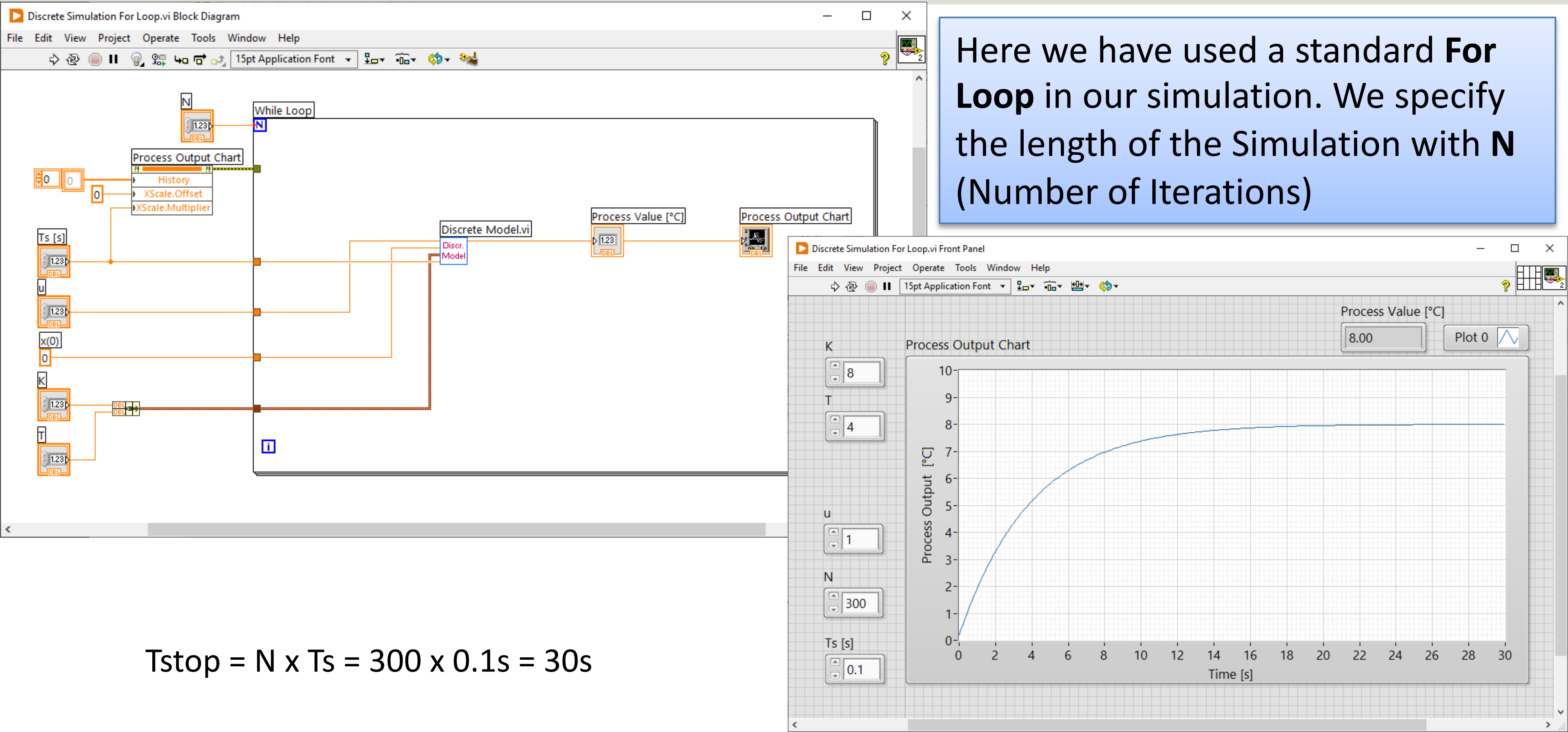
$$x(k+1) = (1 - T_s a)x(k) + T_s b u(k)$$





Here we have implemented the model inside a **Formula Node**. In that way we can write the equations as we do on paper.
We have also implemented the model as a **SubVI** then we can easily reuse the model in different applications

Simulation Example

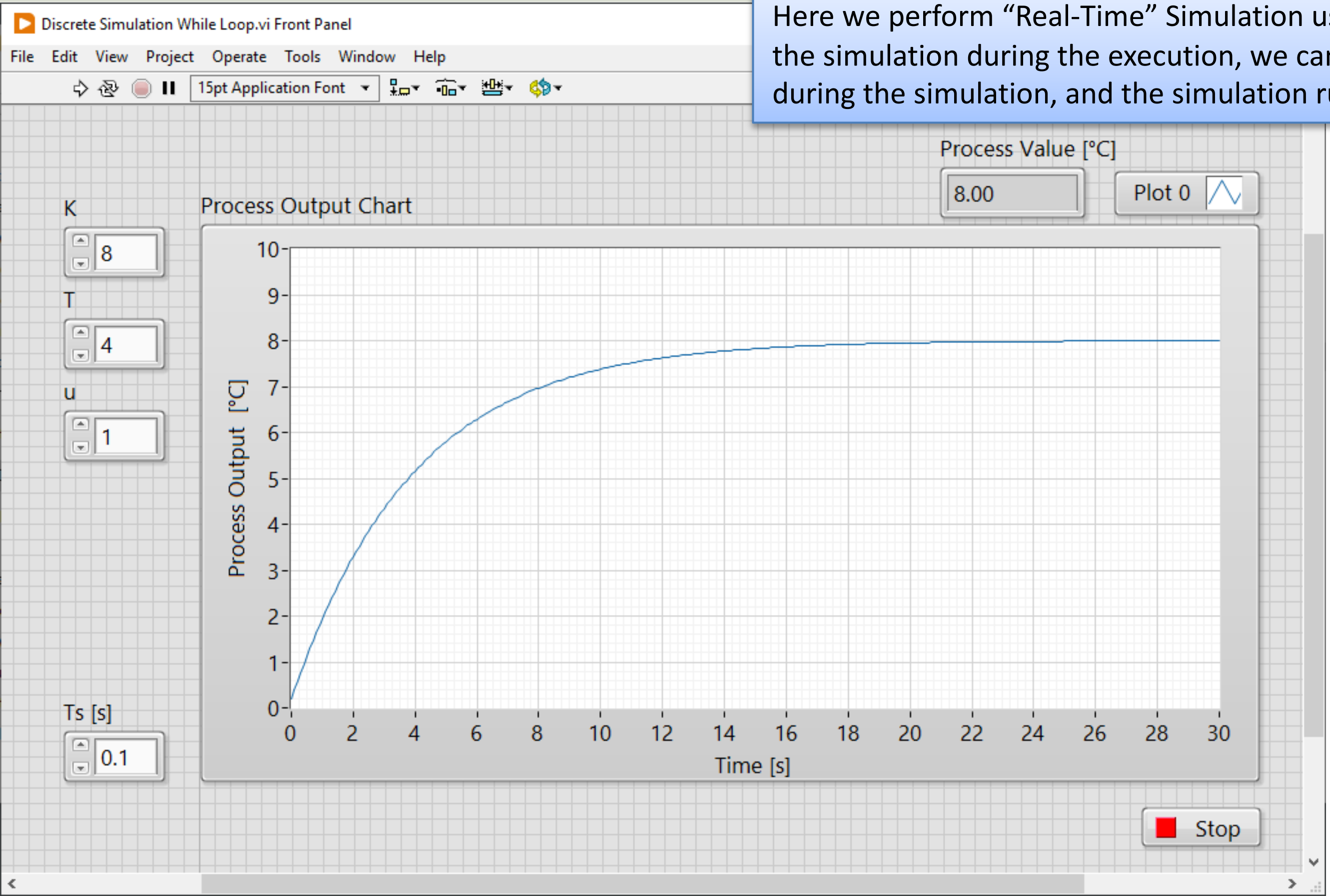


Here we have used a standard **For Loop** in our simulation. We specify the length of the Simulation with **N** (Number of Iterations)

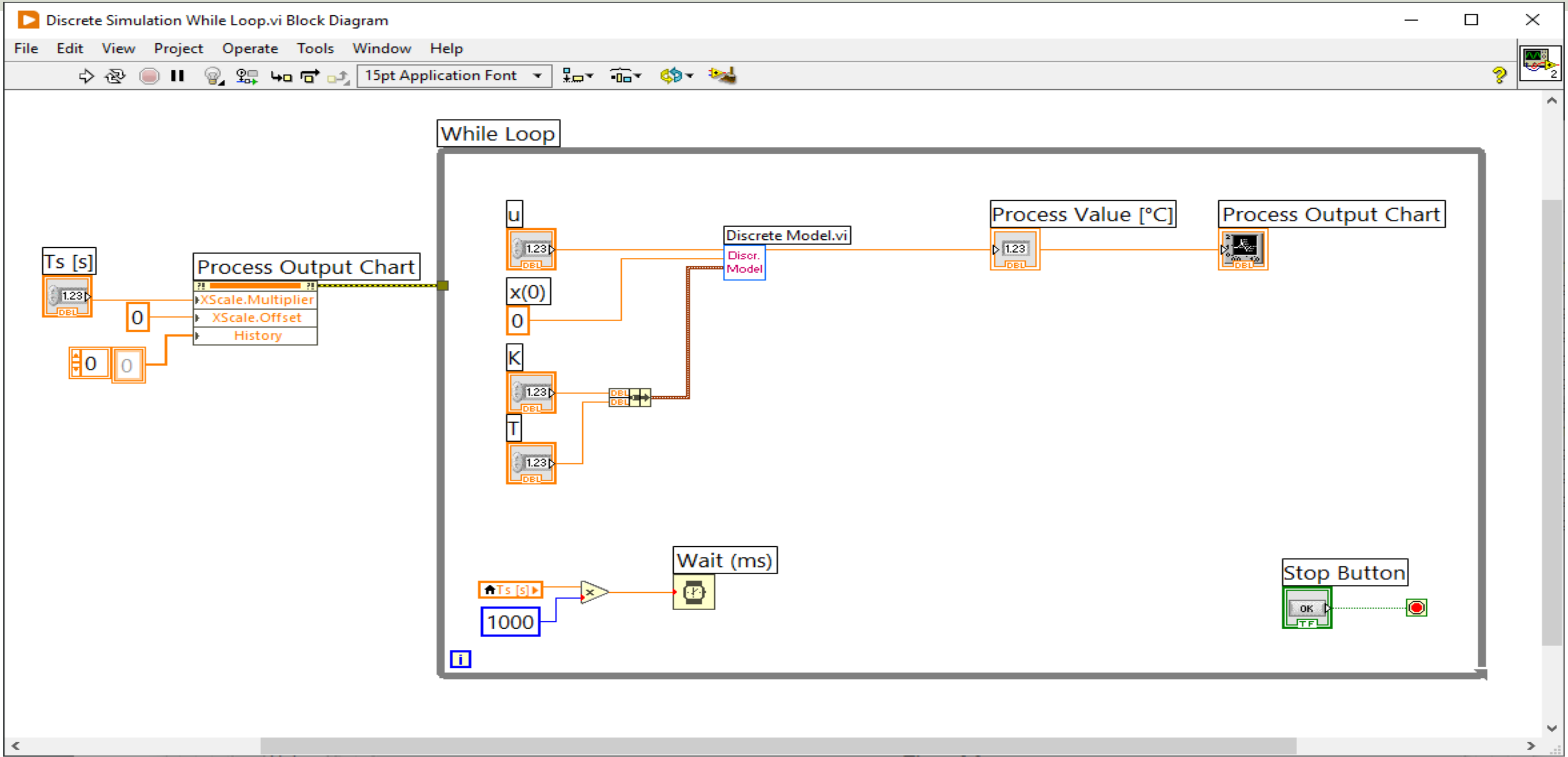
Tstop = N x Ts = 300 x 0.1s = 30s

“Real-Time” Simulation in LabVIEW

Here we perform “Real-Time” Simulation using a **While Loop**. Her we can interact with the simulation during the execution, we can change simulation parameters any time during the simulation, and the simulation runs forever until we click the Stop button.



LabVIEW Code



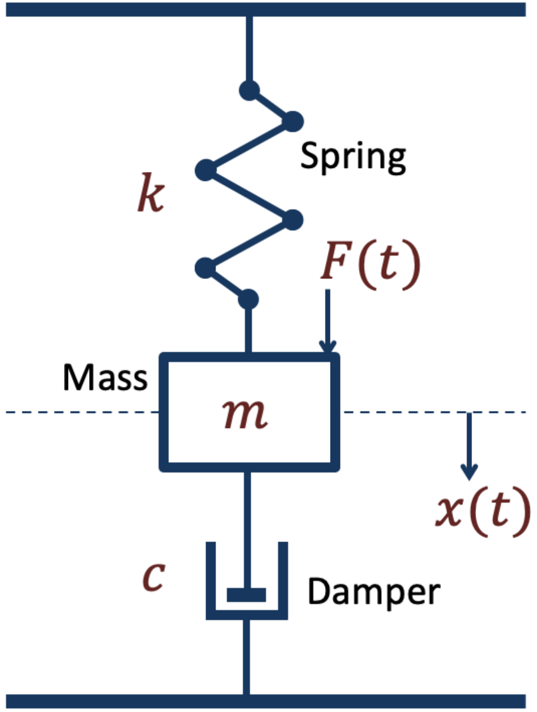
Simulation Examples 2.order System

Mass-Spring-Damper System

Table of Contents

Hans-Petter Halvorsen

Mass-Spring-Damper System



The “Mass-Spring-Damper“ System is typical system used to demonstrate and illustrate Modelling and Simulation Applications

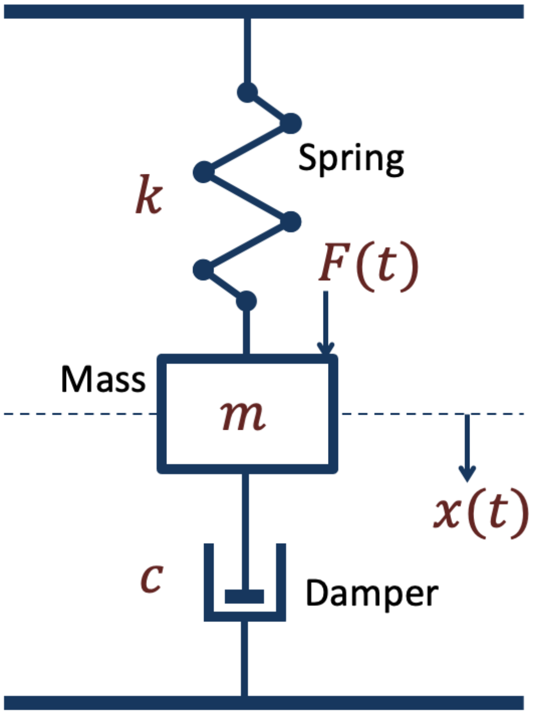
$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t)$$

Force - $F[N]$, Spring constant - $k[N/m]$, Damping coefficient - $c[kg/s]$, Position - $x[m/s]$, Velocity - $\dot{x}[m/s^2]$, Acceleration - $\dot{x}[m/s^2]$, Mass - $m[kg]$

Mass-Spring-Damper System

Given a so-called "Mass-Spring-Damper" system

Newtons 2.law: $\sum F = ma$



The system can be described by the following equation:

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t)$$

Where t is the time, $F(t)$ is an external force applied to the system, c is the damping constant, k is the stiffness of the spring, m is a mass.

$x(t)$ is the position of the object (m)

$\dot{x}(t)$ is the first derivative of the position, which equals the velocity/speed of the object (m)

$\ddot{x}(t)$ is the second derivative of the position, which equals the acceleration of the object (m)

Mass-Spring-Damper System

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t)$$

Higher order differential equations can typically be reformulated into a system of first order differential equations

$$m\ddot{x} = F - c\dot{x} - kx$$

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx)$$

x_1 = Position
x_2 = Velocity/Speed

We set
$x = x_1$
$\dot{x} = x_2$

This gives:
$\dot{x}_1 = x_2$
$\dot{x}_2 = \ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) = \frac{1}{m}(F - cx_2 - kx_1)$

Finally:

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) \Rightarrow \boxed{\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= \frac{1}{m}(F - cx_2 - kx_1)\end{aligned}}$$

Discretization

Given:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$$

Using Euler:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

Then we get:

$$\frac{x_1(k+1) - x_1(k)}{T_s} = x_2(k)$$
$$\frac{x_2(k+1) - x_2(k)}{T_s} = \frac{1}{m}[F(k) - cx_2(k) - kx_1(k)]$$

This gives:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = x_2(k) + T_s \frac{1}{m}[F(k) - cx_2(k) - kx_1(k)]$$

Then we get:

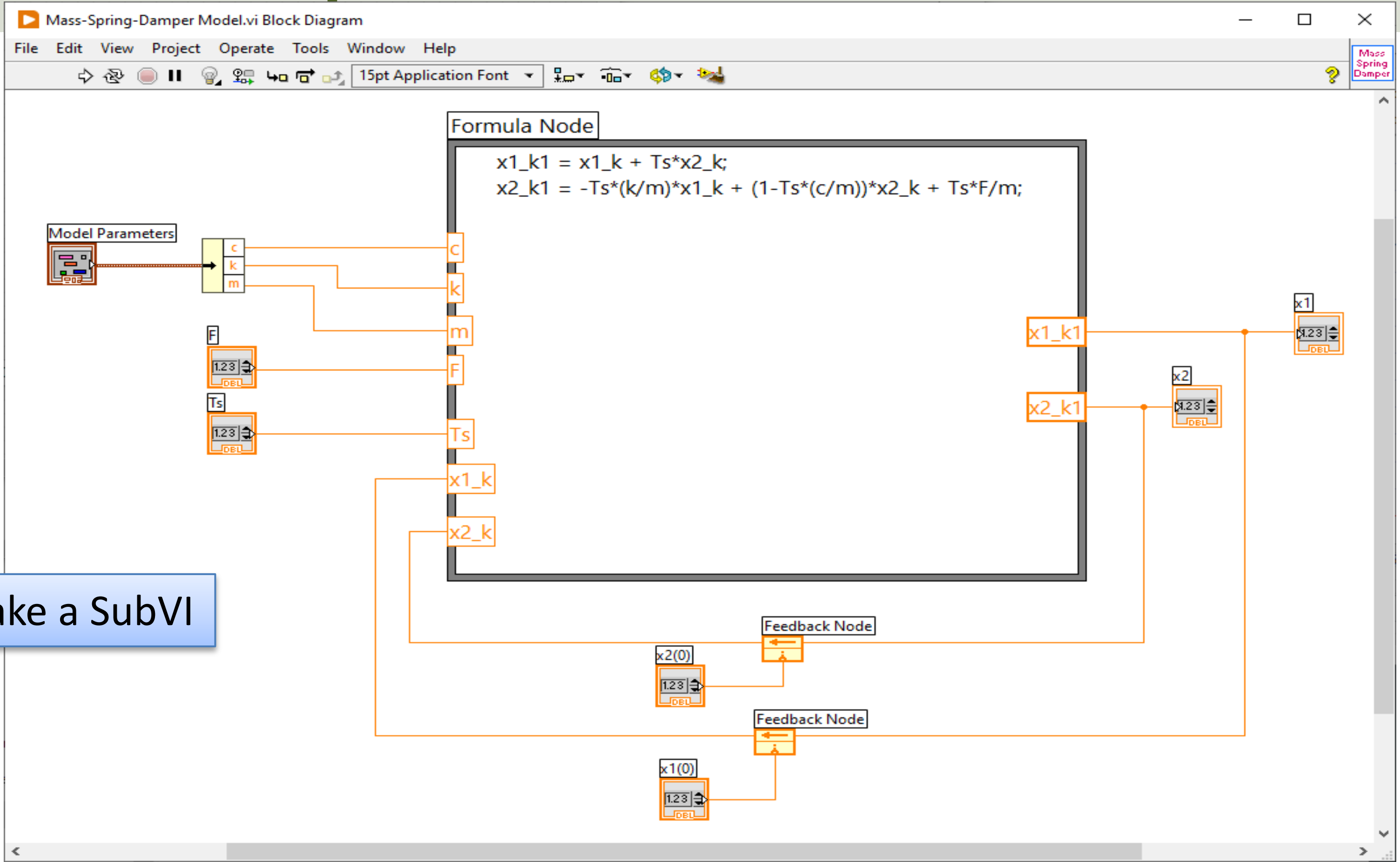
$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + x_2(k) - T_s \frac{c}{m} x_2(k) + T_s \frac{1}{m} F(k)$$

Finally:

$$\boxed{\begin{aligned} x_1(k+1) &= x_1(k) + T_s x_2(k) \\ x_2(k+1) &= -T_s \frac{k}{m} x_1(k) + (1 - T_s \frac{c}{m}) x_2(k) + T_s \frac{1}{m} F(k) \end{aligned}}$$

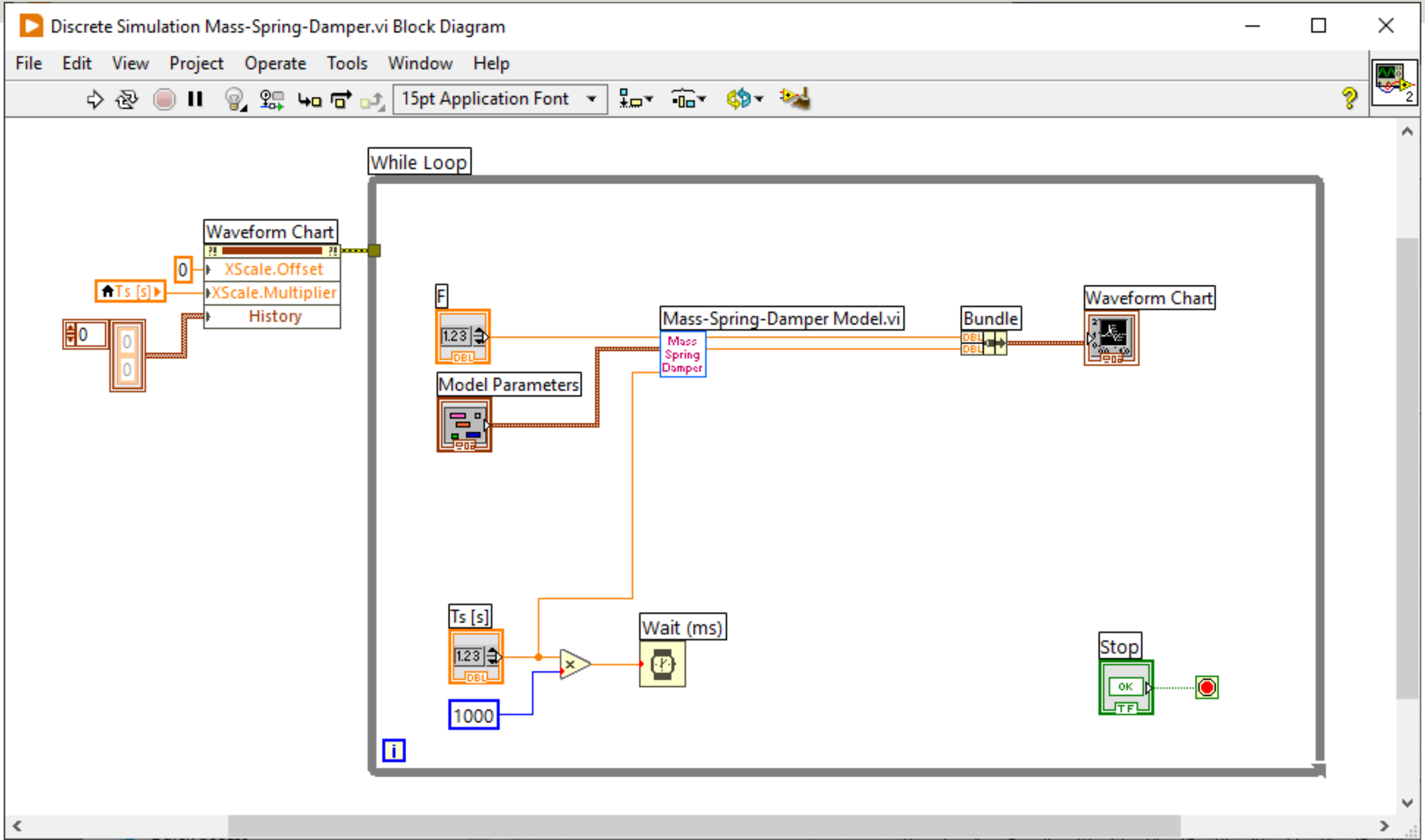
This can be implemented in LabVIEW

Model Implementation in LabVIEW

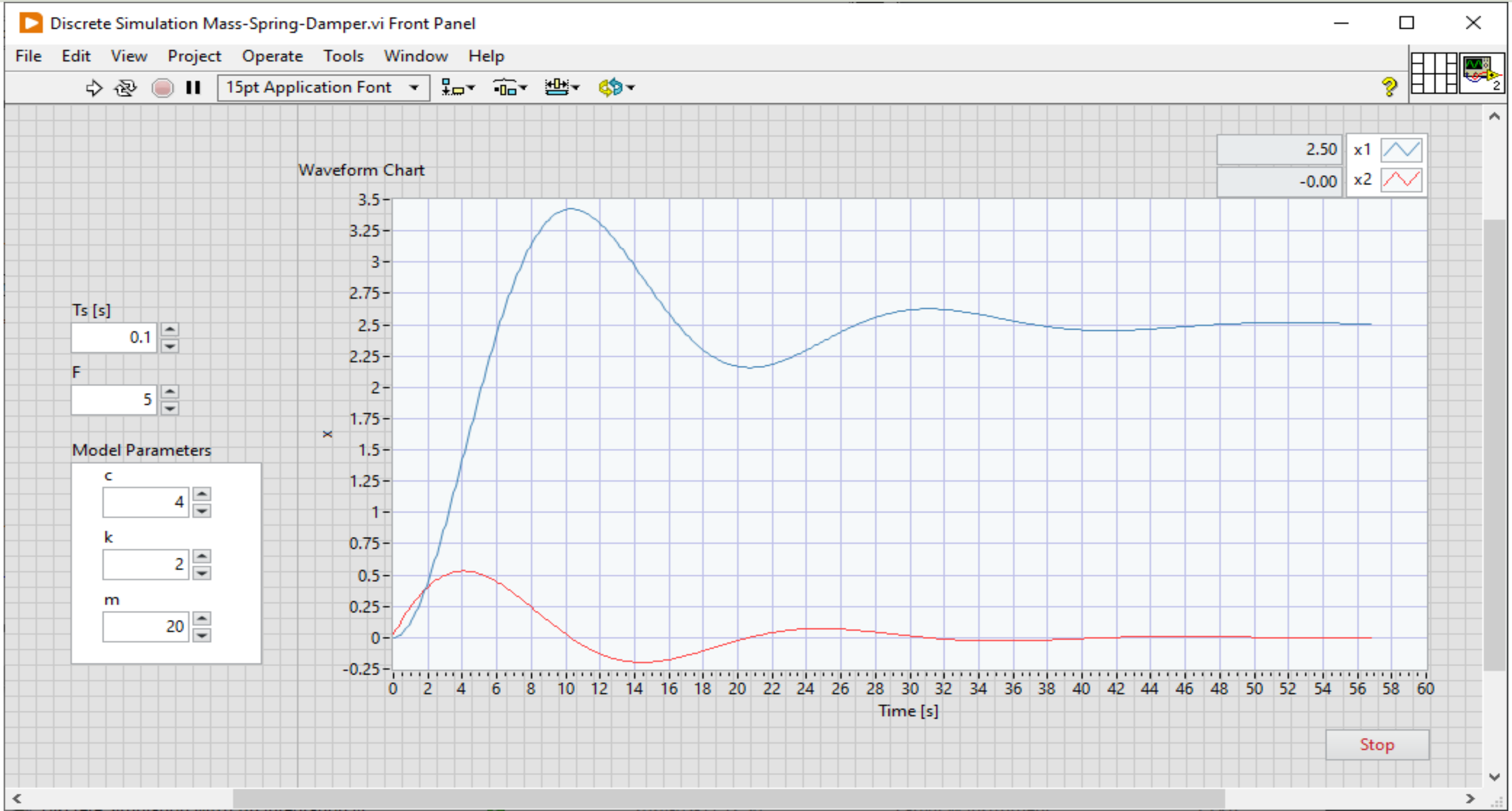


We make a SubVI

Simulation in LabVIEW



LabVIEW



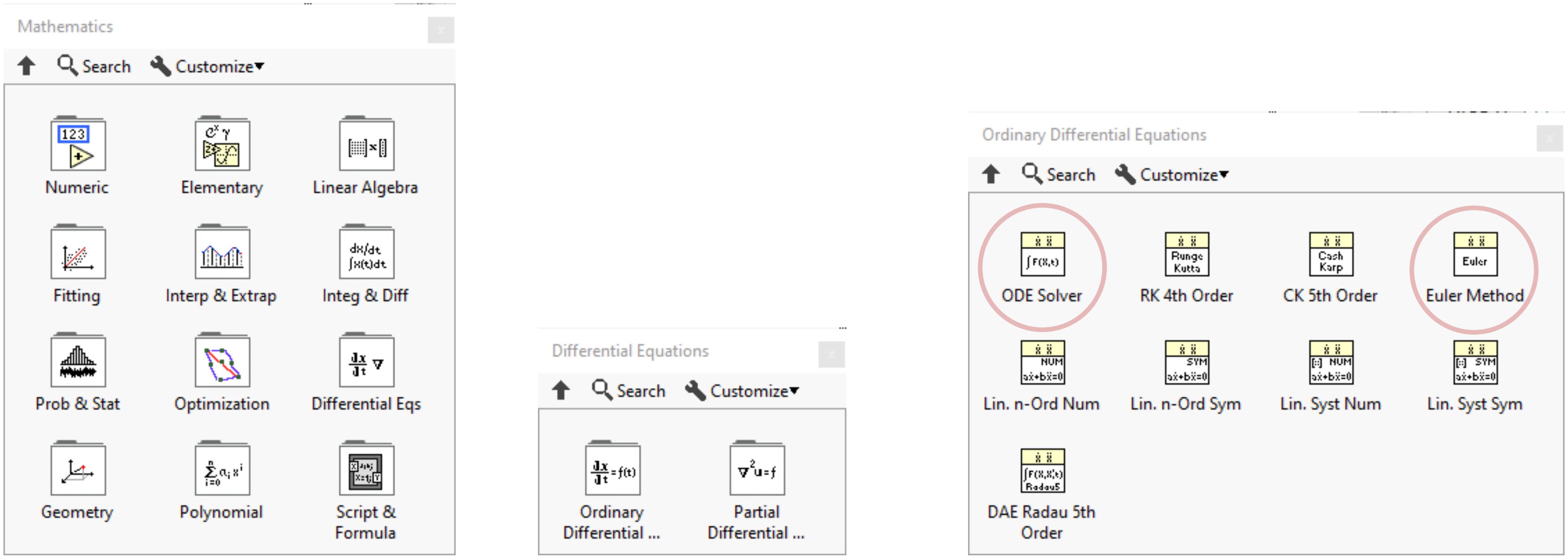
Discrete Simulation Mass-Spring-Damper.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Ts [s]
0.1
F
5
Model Parameters
c
4
k
2
m
20
Waveform Chart
2.50
-0.00
x1
x2
x
3.5
3.25
3
2.75
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
-0.25
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60
Time [s]
Stop

ODE Functions in LabVIEW

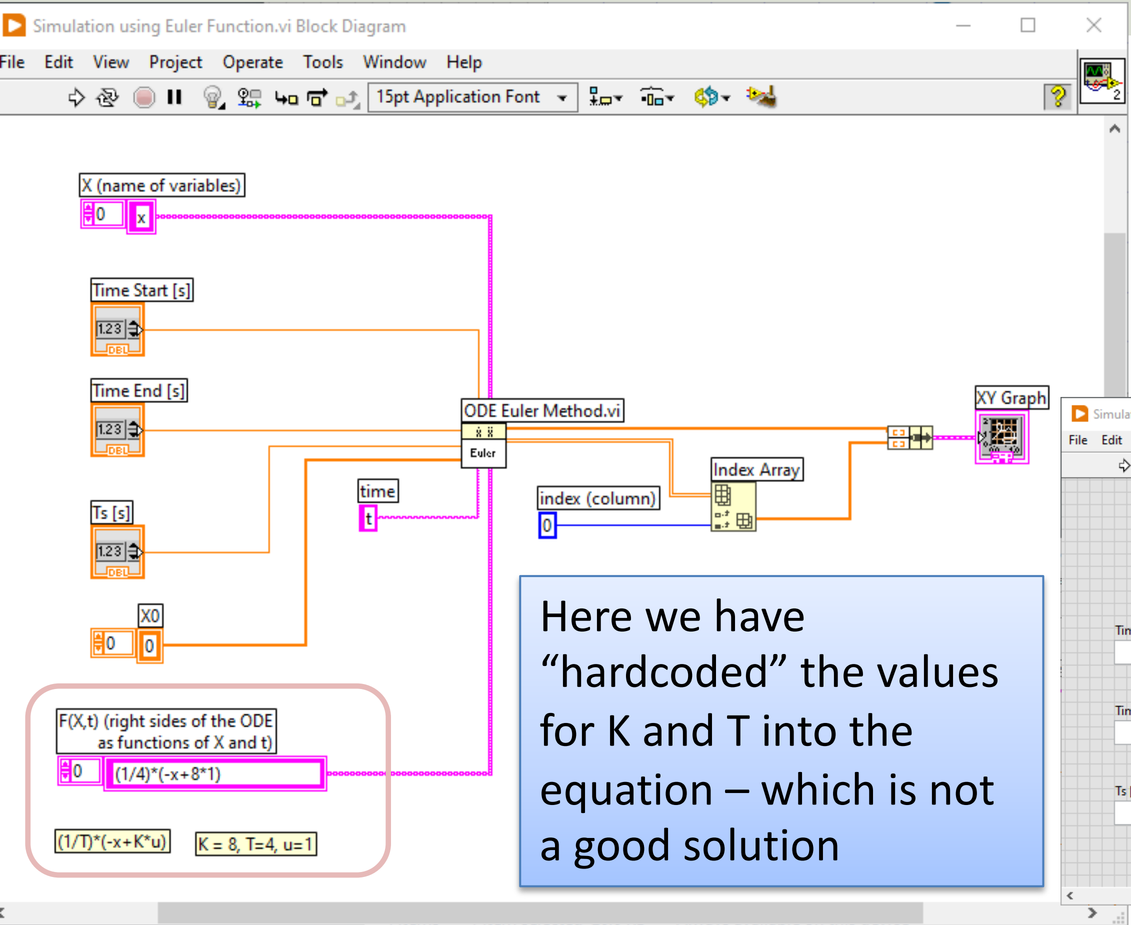
Hans-Petter Halvorsen

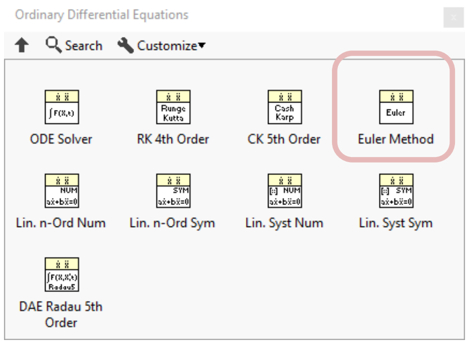
Table of Contents

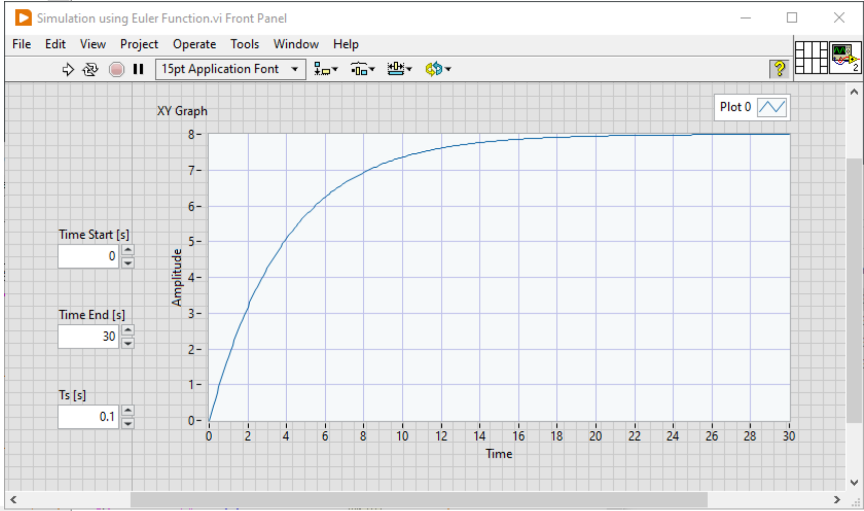
ODE Functions in LabVIEW



ODE Euler Method.vi

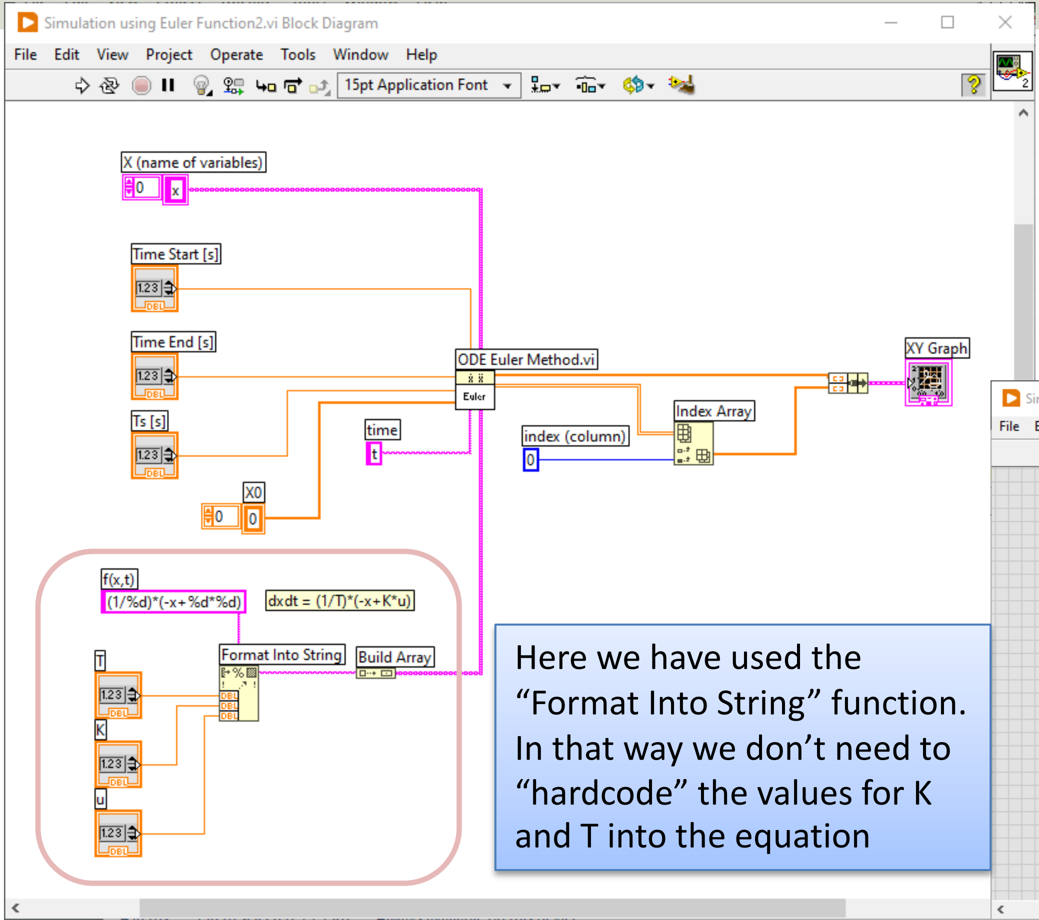


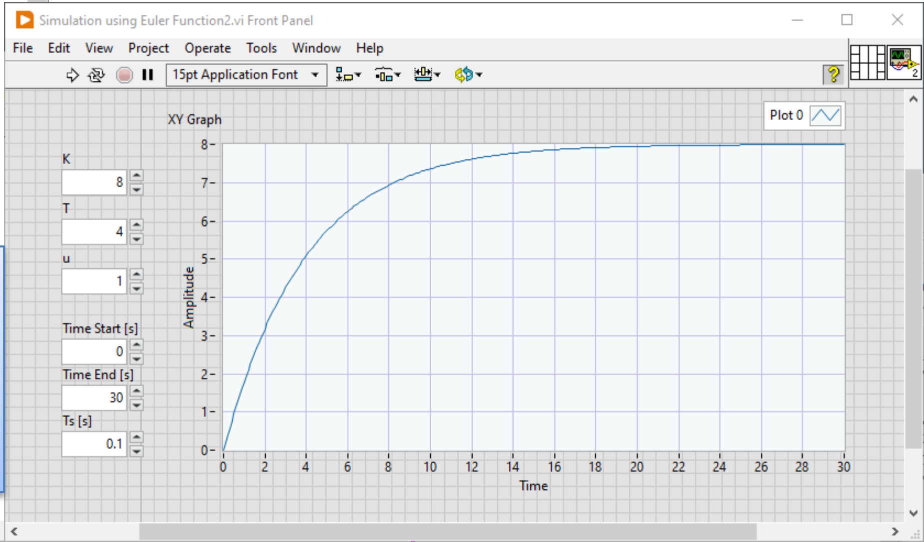




Here we have “hardcoded” the values for K and T into the equation – which is not a good solution

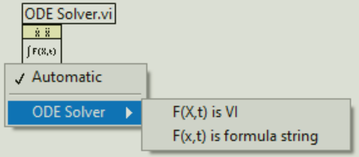
ODE Euler Method.vi – Alt2

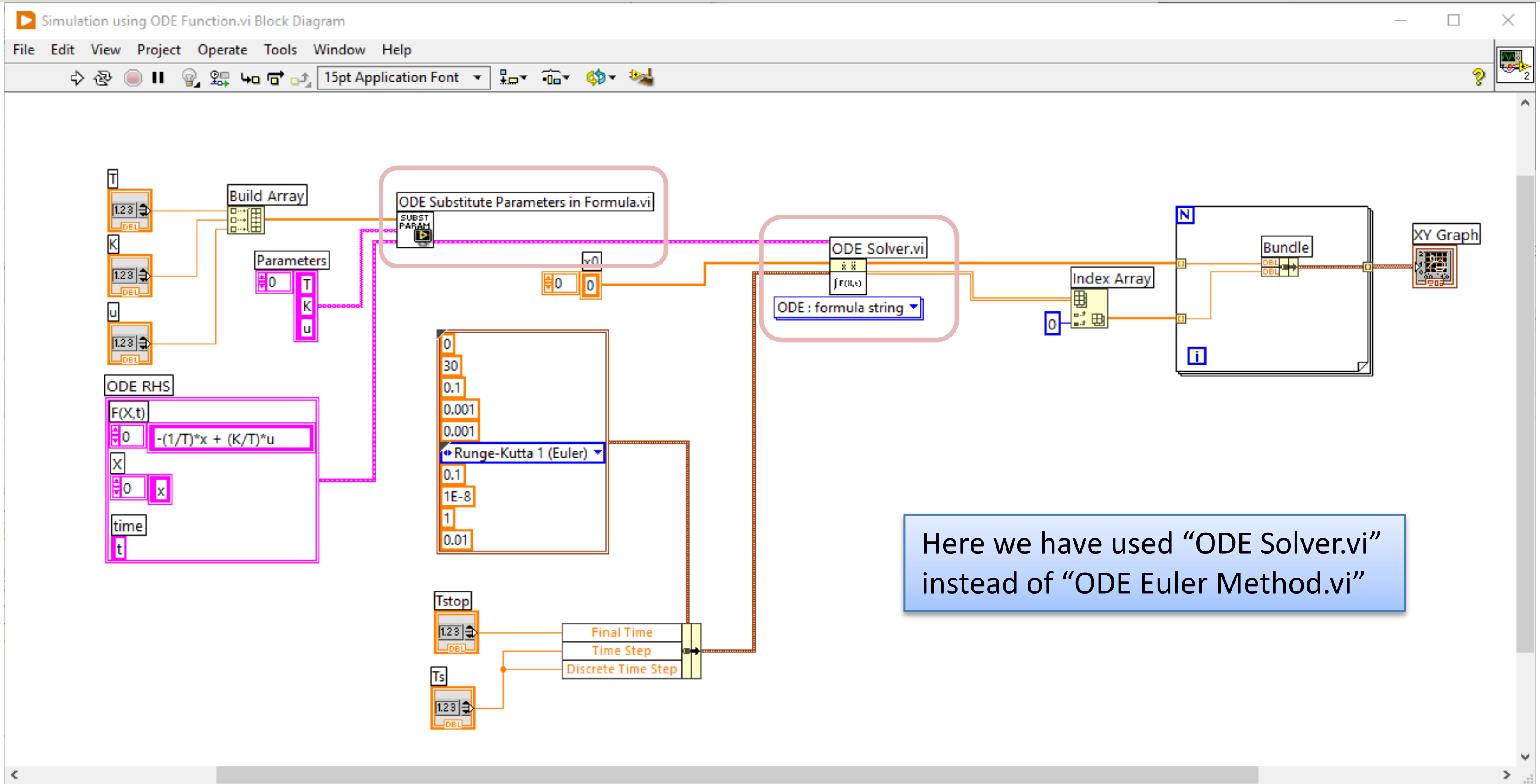




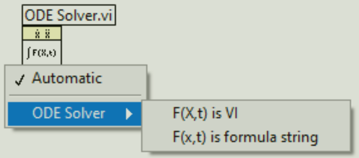
Here we have used the "Format Into String" function. In that way we don't need to "hardcode" the values for K and T into the equation

ODE Solver.vi

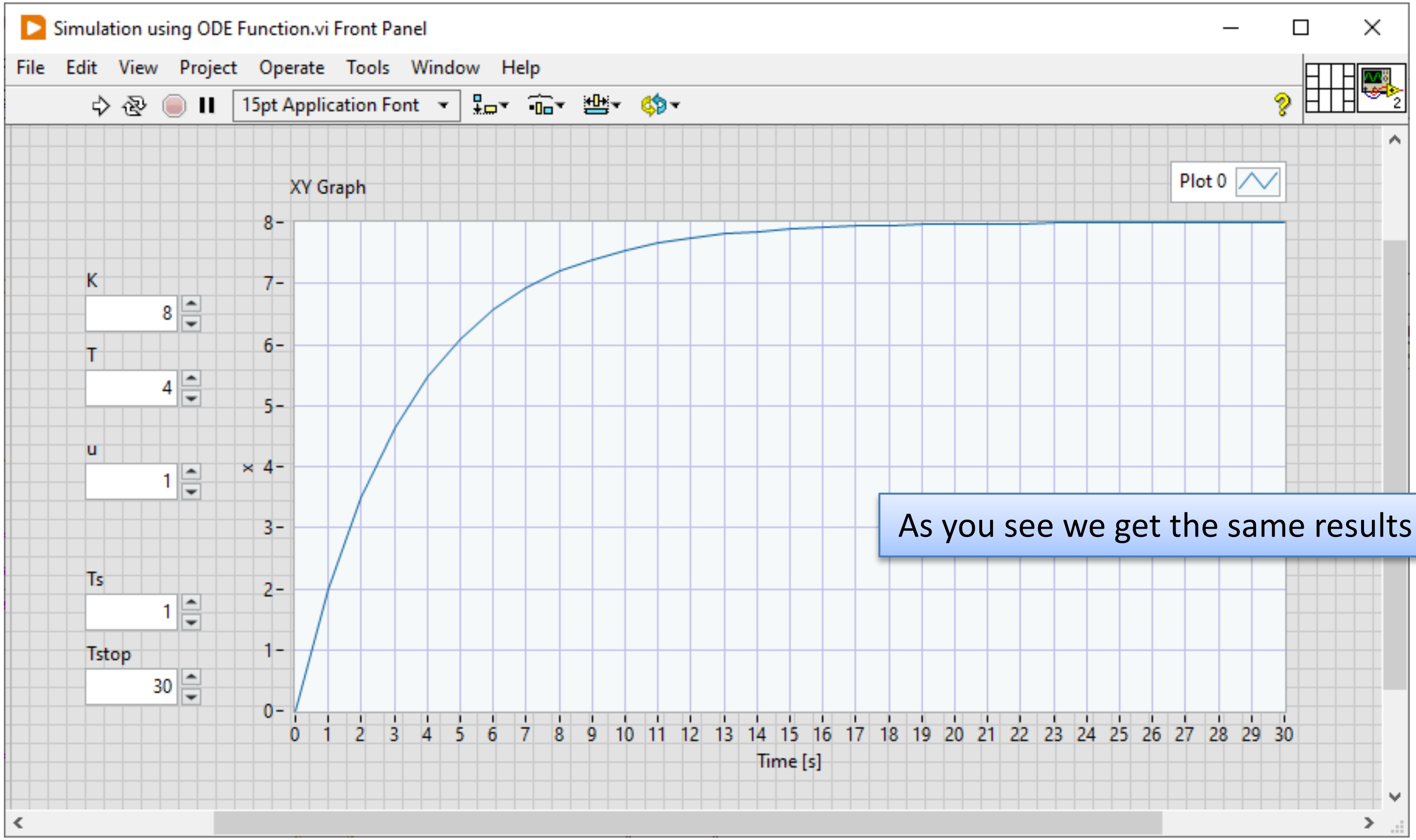




Here we have used "ODE Solver.vi" instead of "ODE Euler Method.vi"



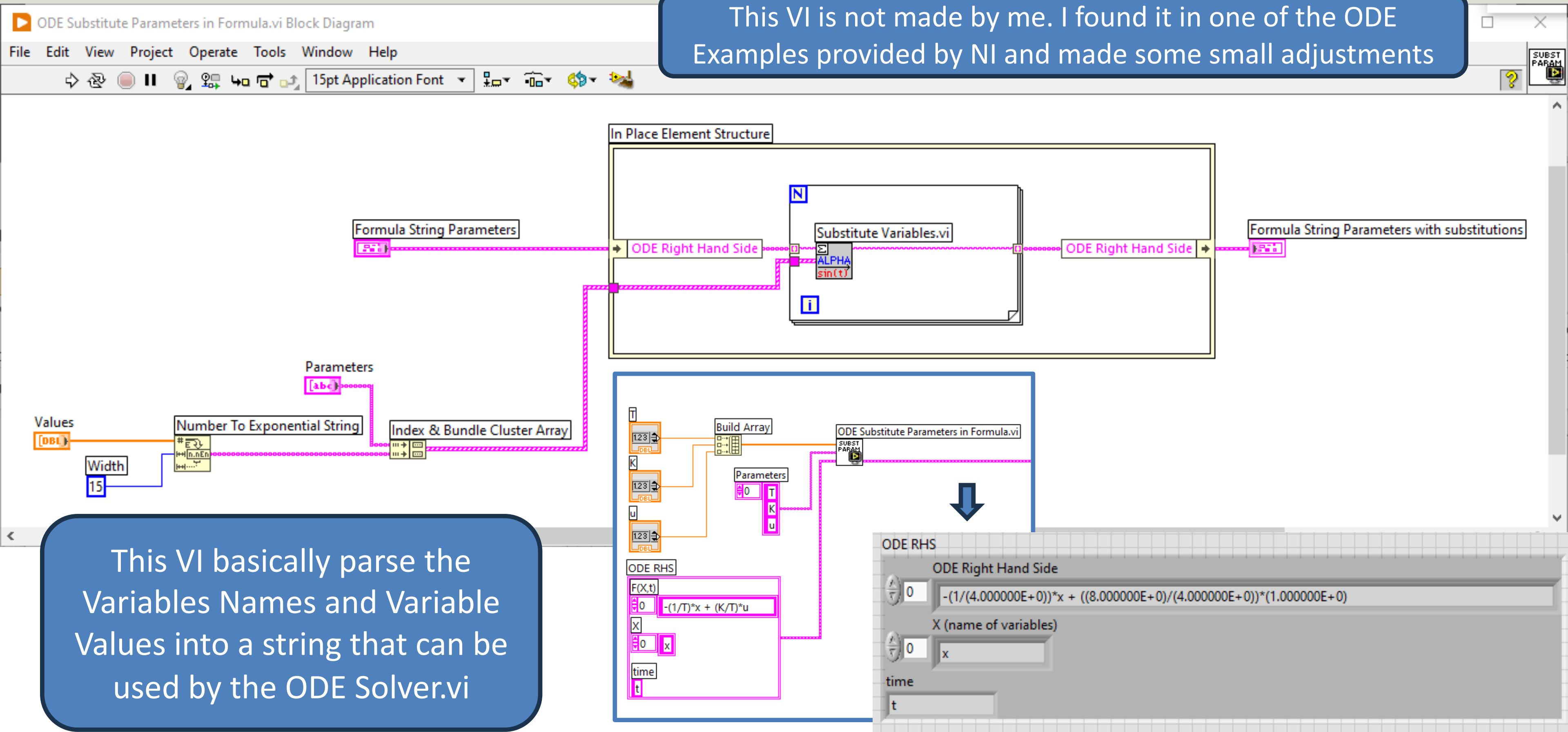
ODE Solver.vi



As you see we get the same results

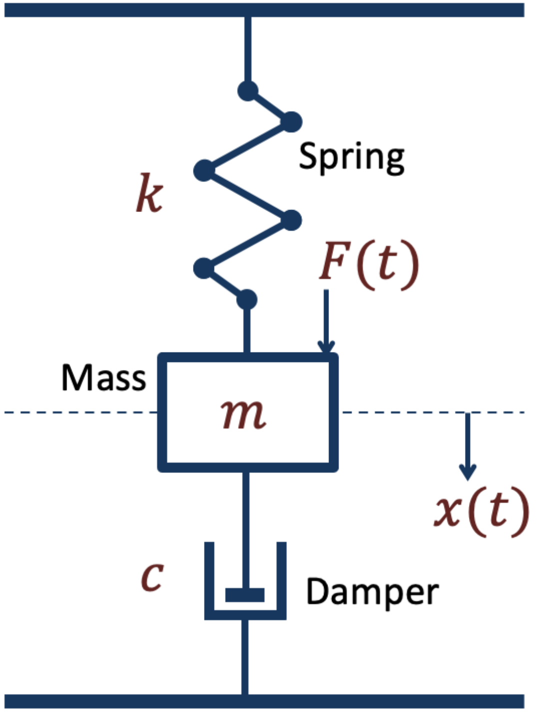
ODE Substitute Parameters in Formula.vi

This VI is not made by me. I found it in one of the ODE Examples provided by NI and made some small adjustments



This VI basically parse the Variables Names and Variable Values into a string that can be used by the ODE Solver.vi

Mass-Spring Damper System



See if you can use “ODE Solver.vi” for implementing and simulating the Mass-Spring Damper System as well (I leave that to you)

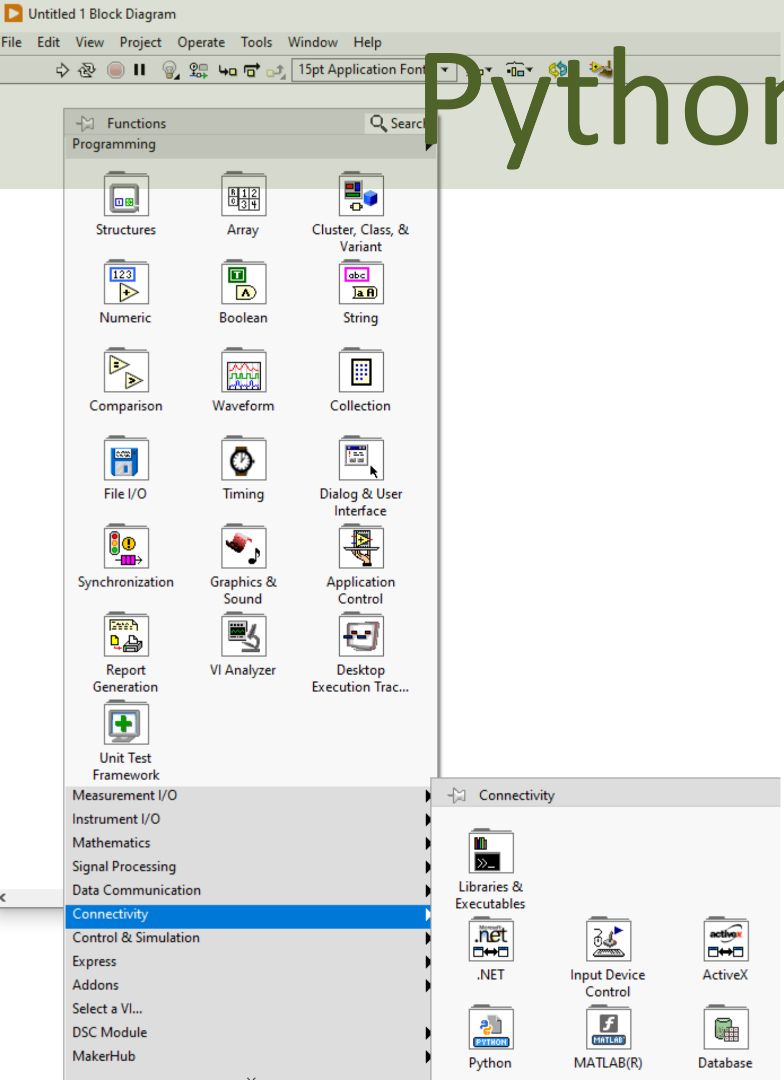
$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$$

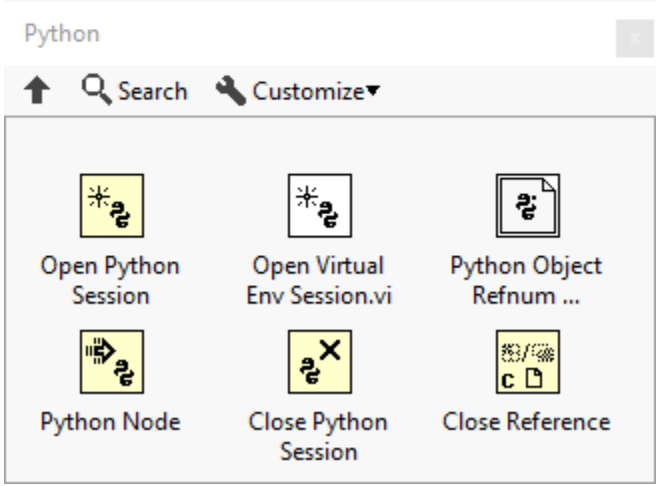
Python Integration

Table of Contents

Hans-Petter Halvorsen

Python Integration





Python Integration Example

We start by making the Python code using Spyder or another Python Editor

```
def c2f(Tc):
    Tf = (Tc * 9/5) + 32
    return Tf

def f2c(Tf):
    Tc = (Tf - 32)*(5/9)
    return Tc
```

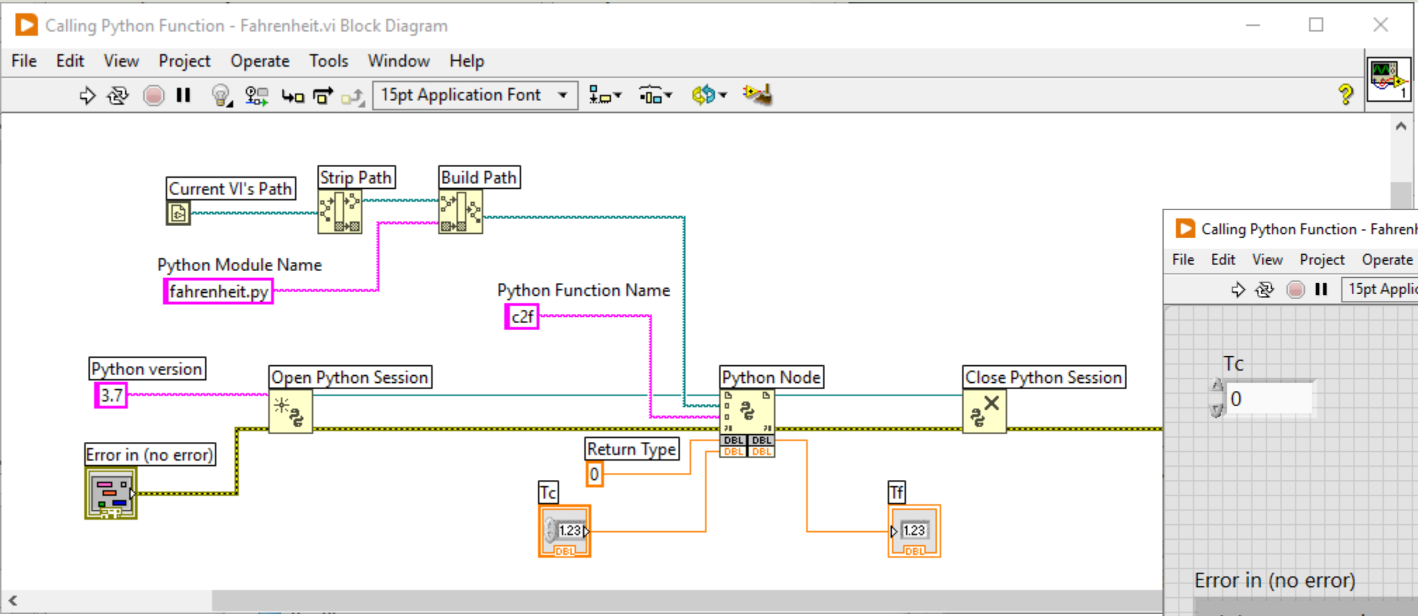
fahrenheit.py

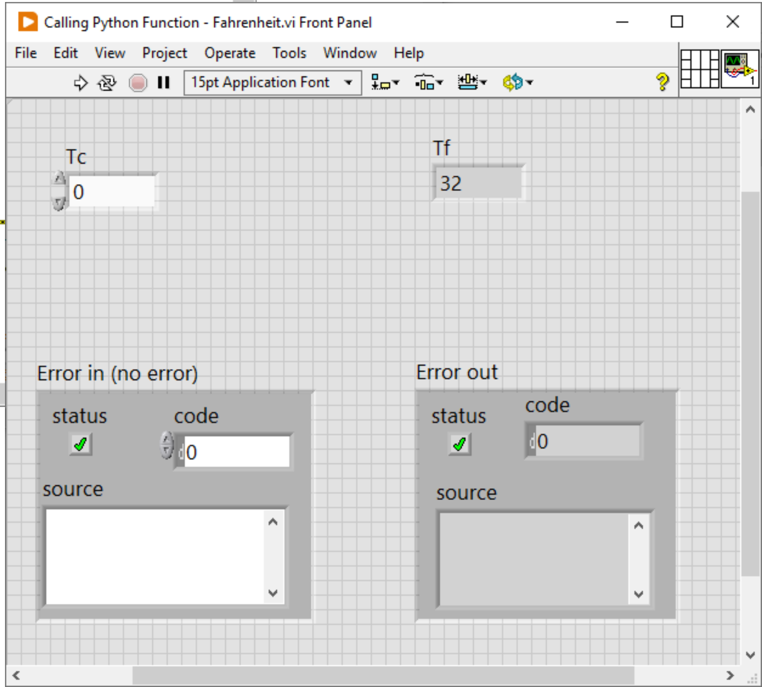
We make a Python Module with 2 Functions, one that converts from Celsius to Fahrenheit and another that converts from Fahrenheit to Celsius

We test if it works:

```
from fahrenheit import c2f, f2c

Tc = 0

Tf = c2f(Tc)

print("Fahrenheit: " + str(Tf))

Tf = 32

Tc = f2c(Tf)

print("Celsius: " + str(Tc))
```

Python Integration Example





Note! LabVIEW and Python needs to match. If you use a 32bit LabVIEW version, you also need to use a 32bit Python version

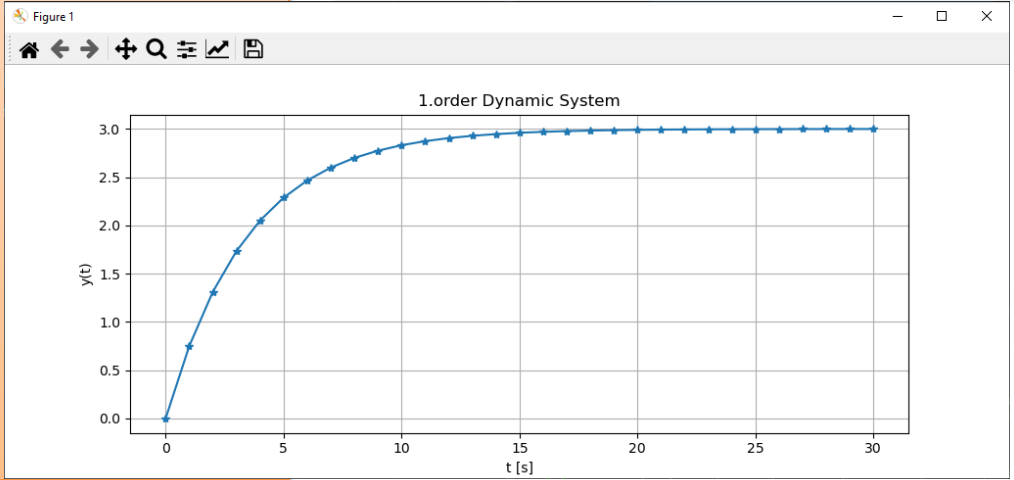
Simulation

```
import numpy as np

def sim_ex():
    # Model Parameters
    K = 3
    T = 4
    a = -1/T
    b = K/T

    #Simulation Parameters
    yk = 0
    uk = 1
    Tstop = 30
    Ts = 1
    N = int(Tstop/Ts) # Simulation length

    data = []
    data.append(yk)

    # Simulation
    for k in range(N):
        #Model Implementation
        yk1 = (1 + a*Ts) * yk + Ts*b*uk

        yk = yk1
        data.append(yk1)

    t = np.arange(0,Tstop+Ts,Ts)
    return t, data
```

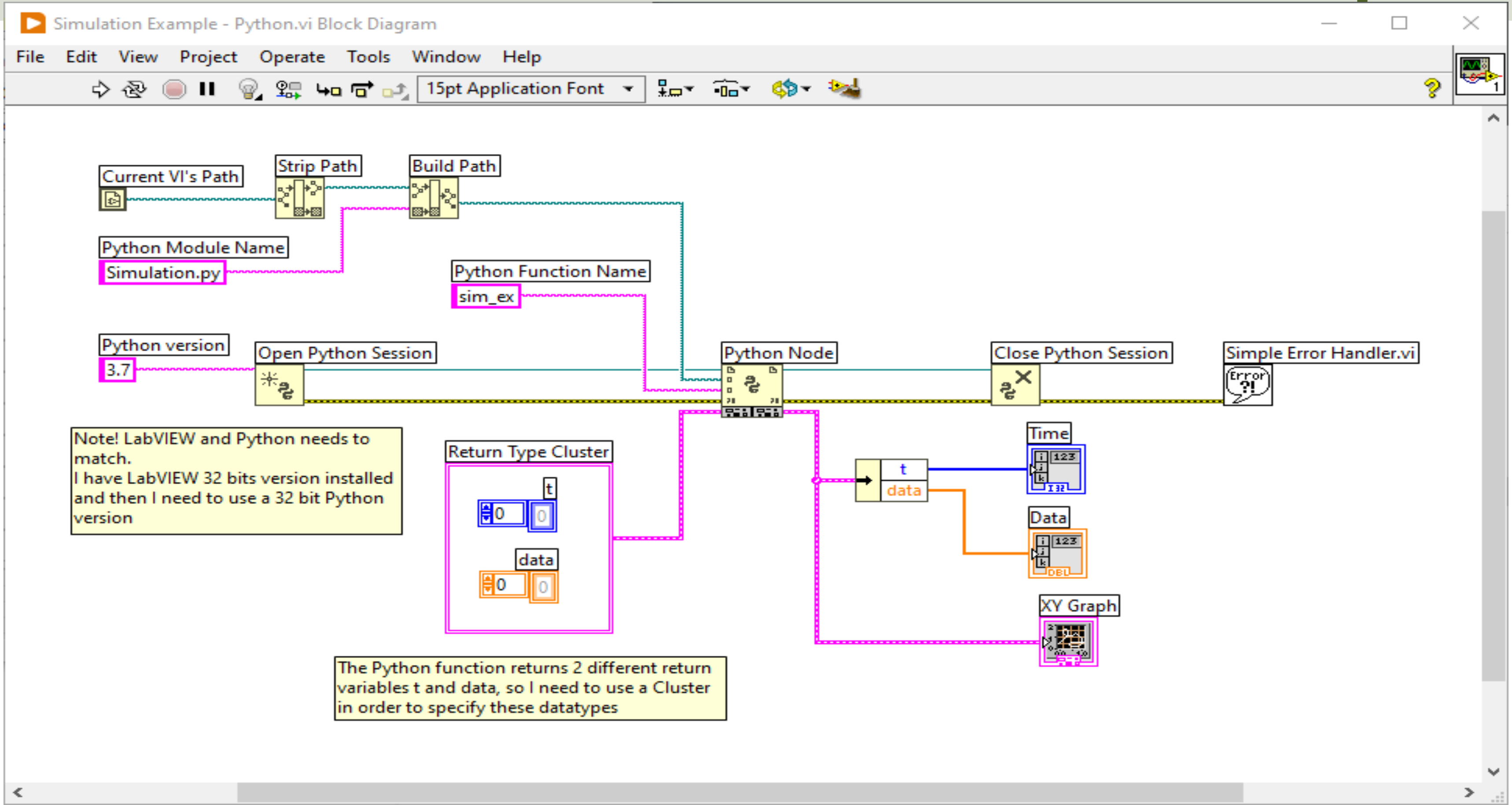
Here we make a discrete simulation example in Python using our 1.order model from previous examples

Simulation.py

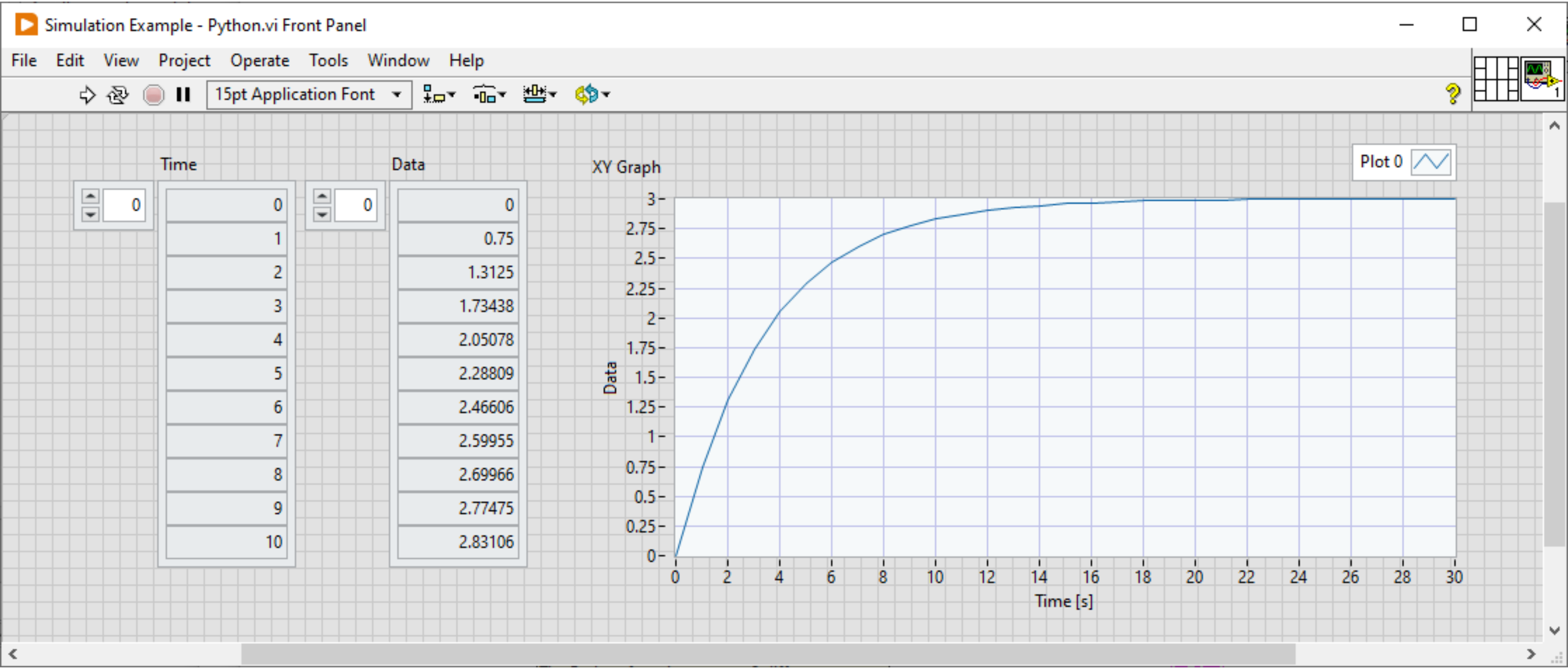
```
import matplotlib.pyplot as plt
from Simulation import sim_ex

#Run Simulation
t, data = sim_ex()

# Plot the Simulation Results
plt.plot(t,data,'-*')
plt.title('1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```



LabVIEW Simulation Example



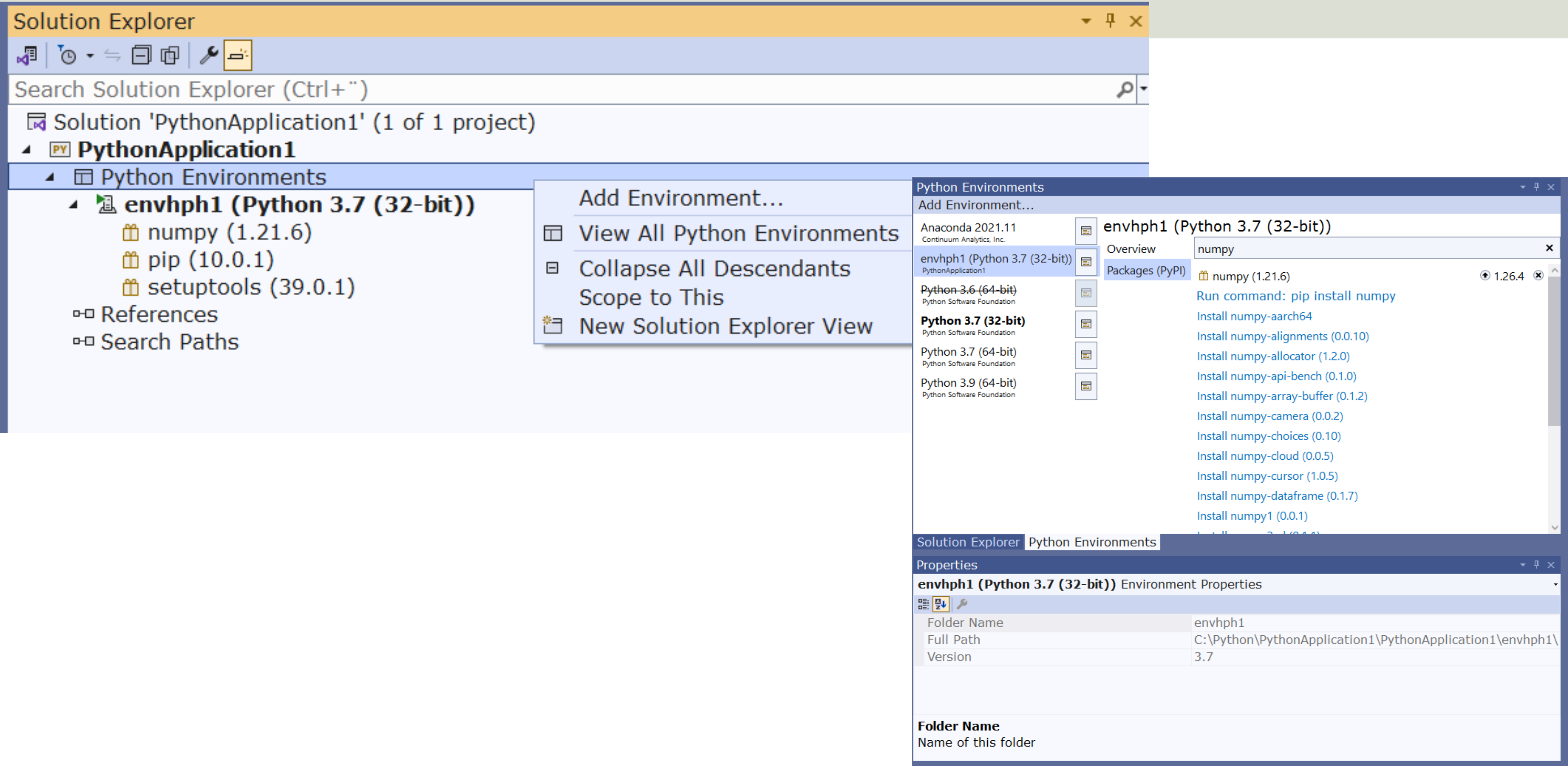
LabVIEW Simulation Example



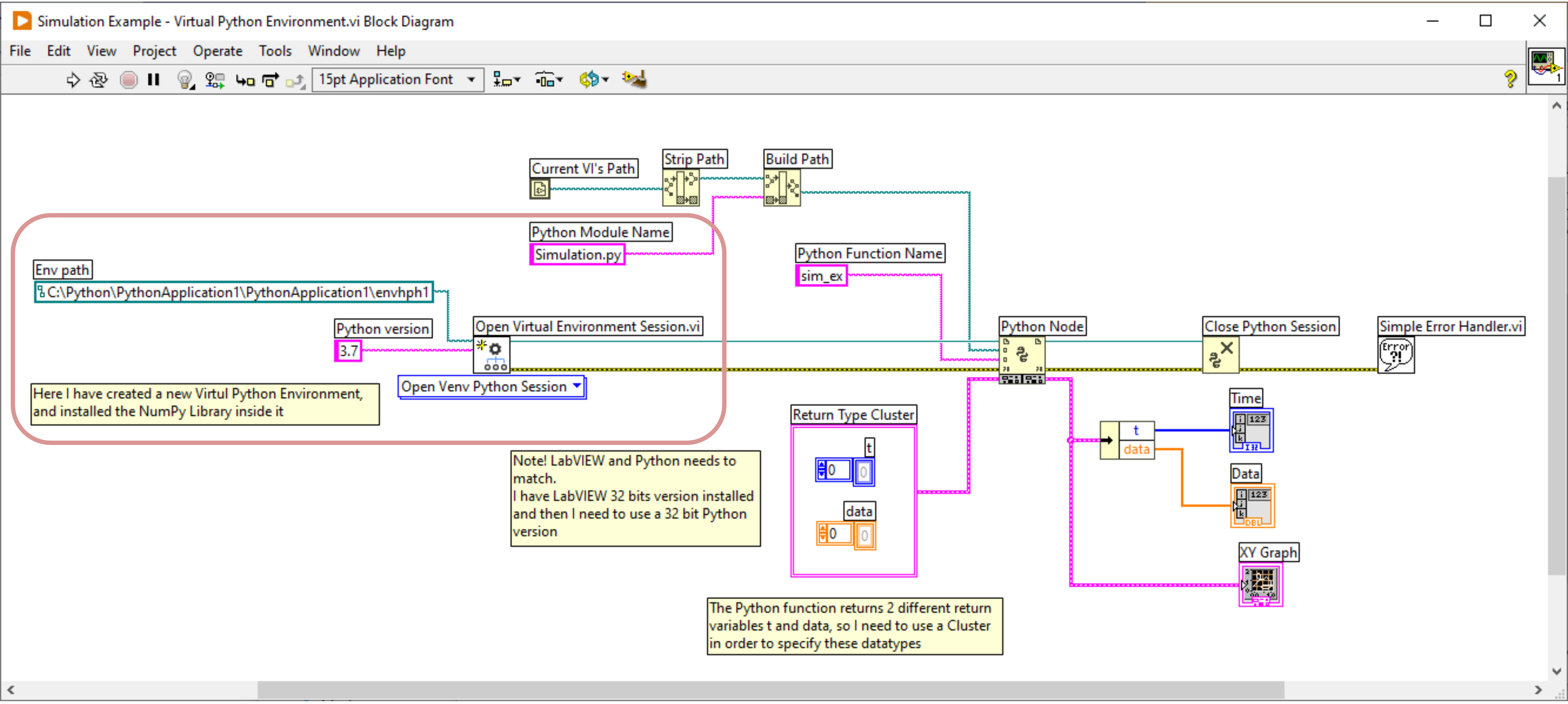
Virtual Python Environment

- With Python you can create Virtual Environments.
- Here you can install your independent set of Python packages.
- In that way you can create an isolated environment where you can run your Python Applications/Scripts without destroying for other Applications/Scripts using other versions of different Python packages.
- You can create a Virtual Python Environment using **venv** command:
  python -m venv /path/to/new/virtual/environment
- You can also use tools like Visual Studio, VenviPy, etc. to do this from a user interface.

Virtual Environment with Visual Studio



Virtual Python Environment



https://www.halvorsen.blog

MATLAB Integration

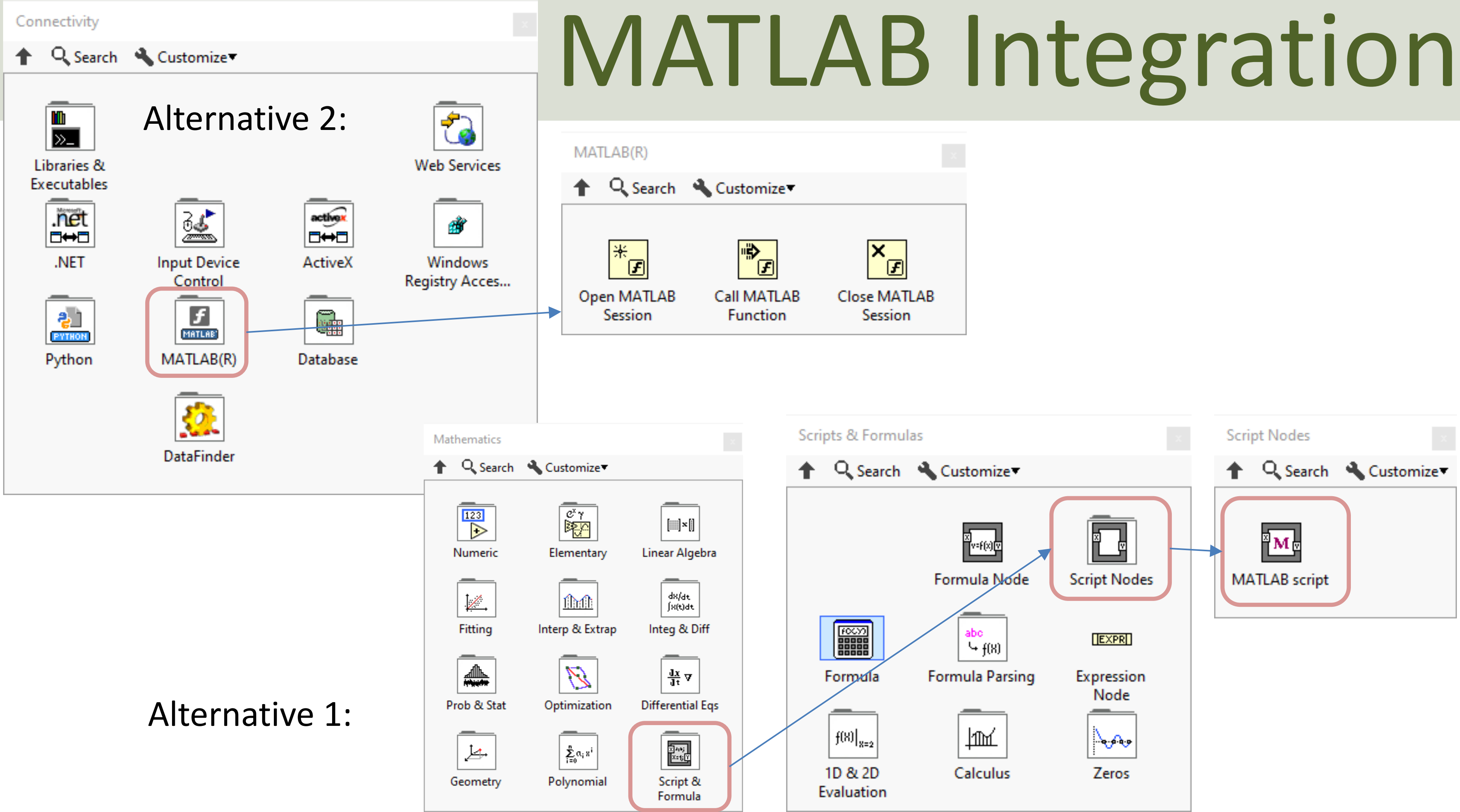
Hans-Petter Halvorsen

Table of Contents

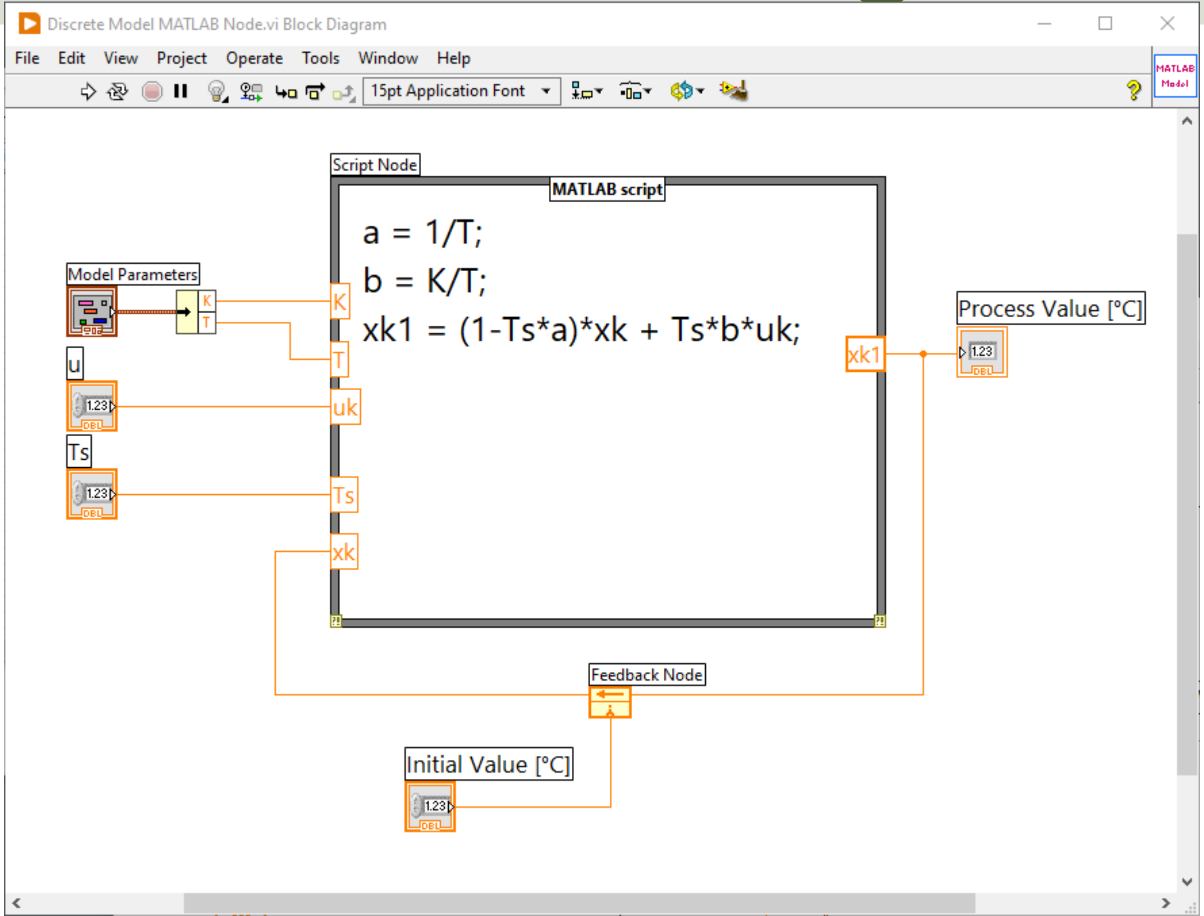
MATLAB Integration

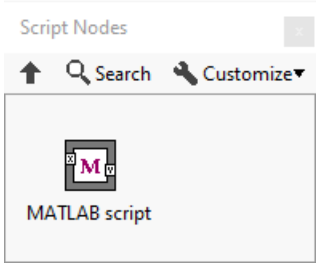
Alternative 2:

Alternative 1:

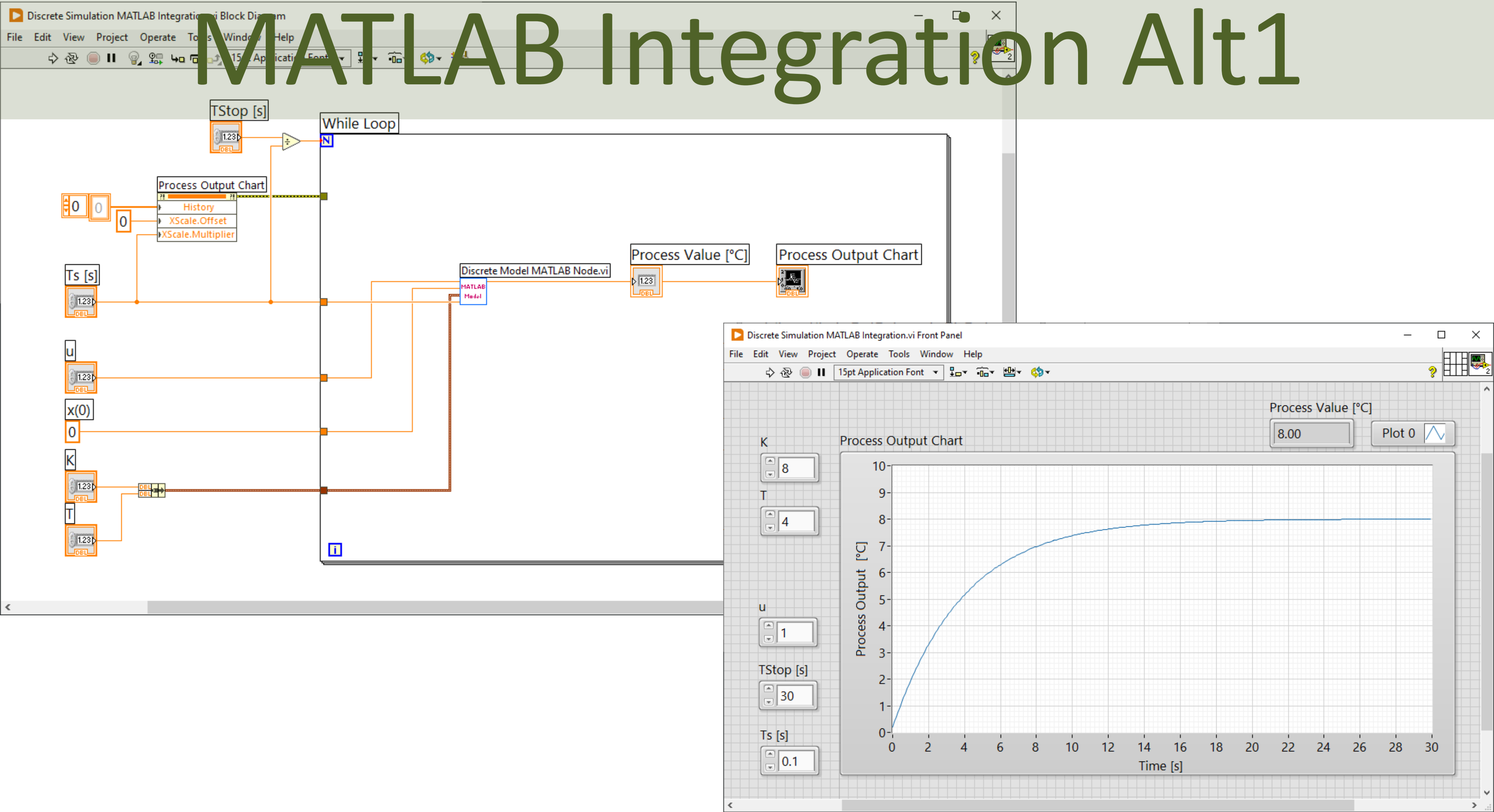


MATLAB Integration Alt1



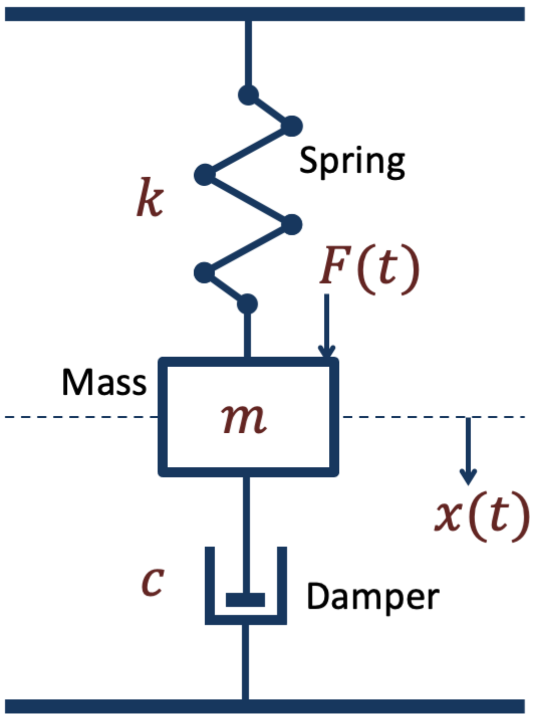


MATLAB Integration Alt1

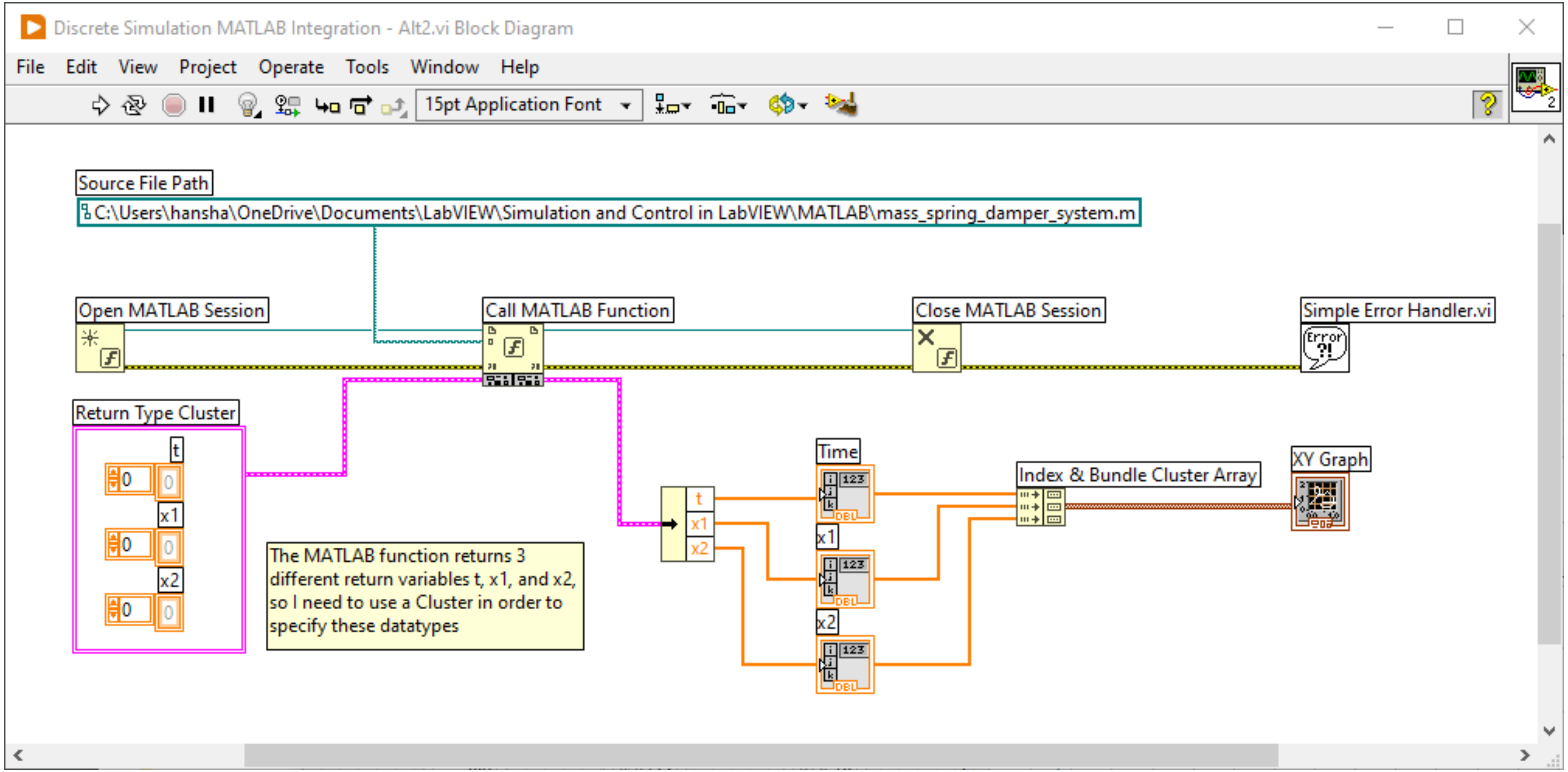


MATLAB Integration Alt2

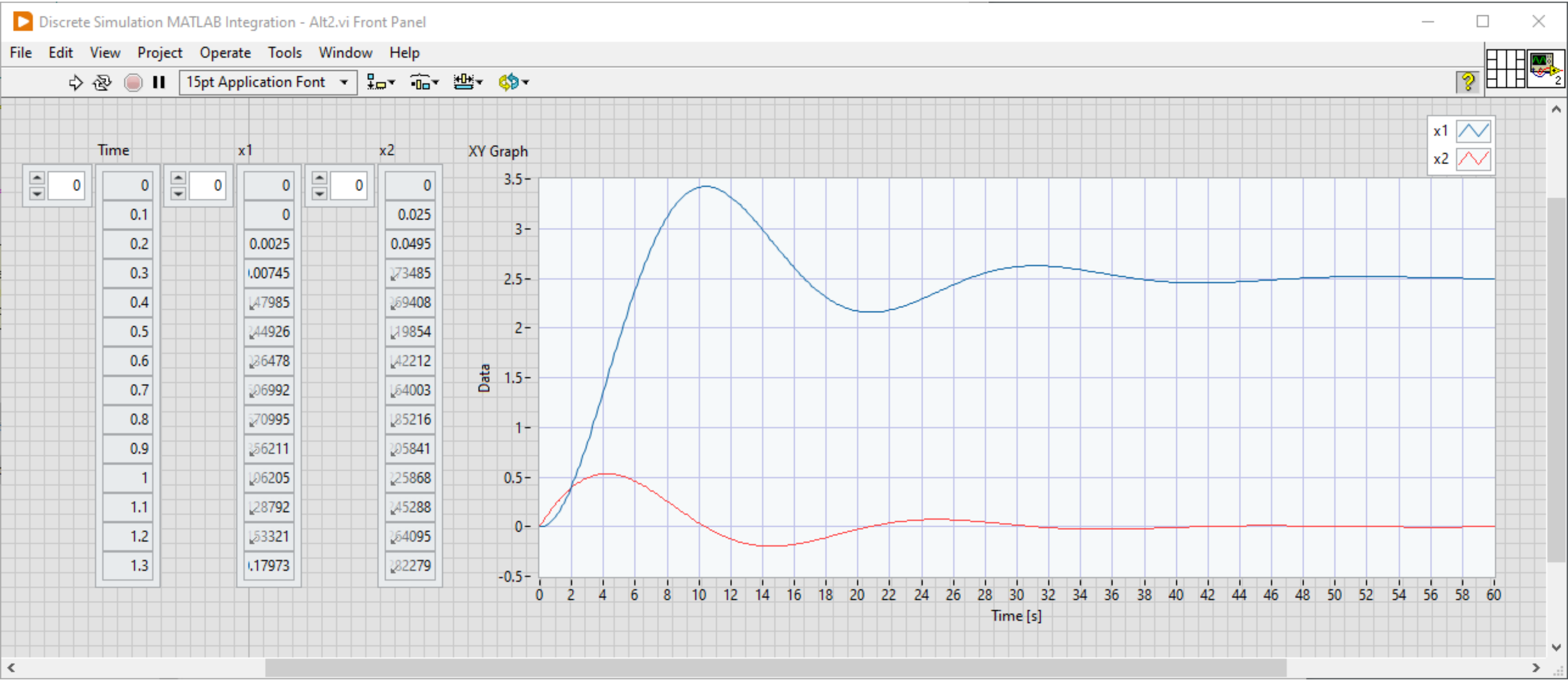
```
function [t, x1, x2] = mass_spring_damper_system()

% Simulation of Mass-Spring-Damper System
clear
clc

% Model Parameters
c = 4; % Damping constant
k_stiff = 2; % Stiffness of the spring
m = 20; % Mass
F = 5; % Force

% Simulation Parameters
Ts = 0.1;
Tstart = 0;
Tstop = 60;
N = (Tstop-Tstart)/Ts; % Simulation length
t = Tstart : Ts : Tstop;
x1 = zeros(N,1);
x2 = zeros(N,1);
x1(1) = 0; % Initial Position
x2(1) = 0; % Initial Speed

% Simulation
for k=1:N
   x1(k+1) = x1(k) + Ts * x2(k);
   x2(k+1) = (-(Ts*k_stiff)/m) * x1(k) + (1 - (Ts*c)/m) * x2(k) + (Ts/m) * F;
end
```



MATLAB Integration Alt2



MATLAB Integration Alt2



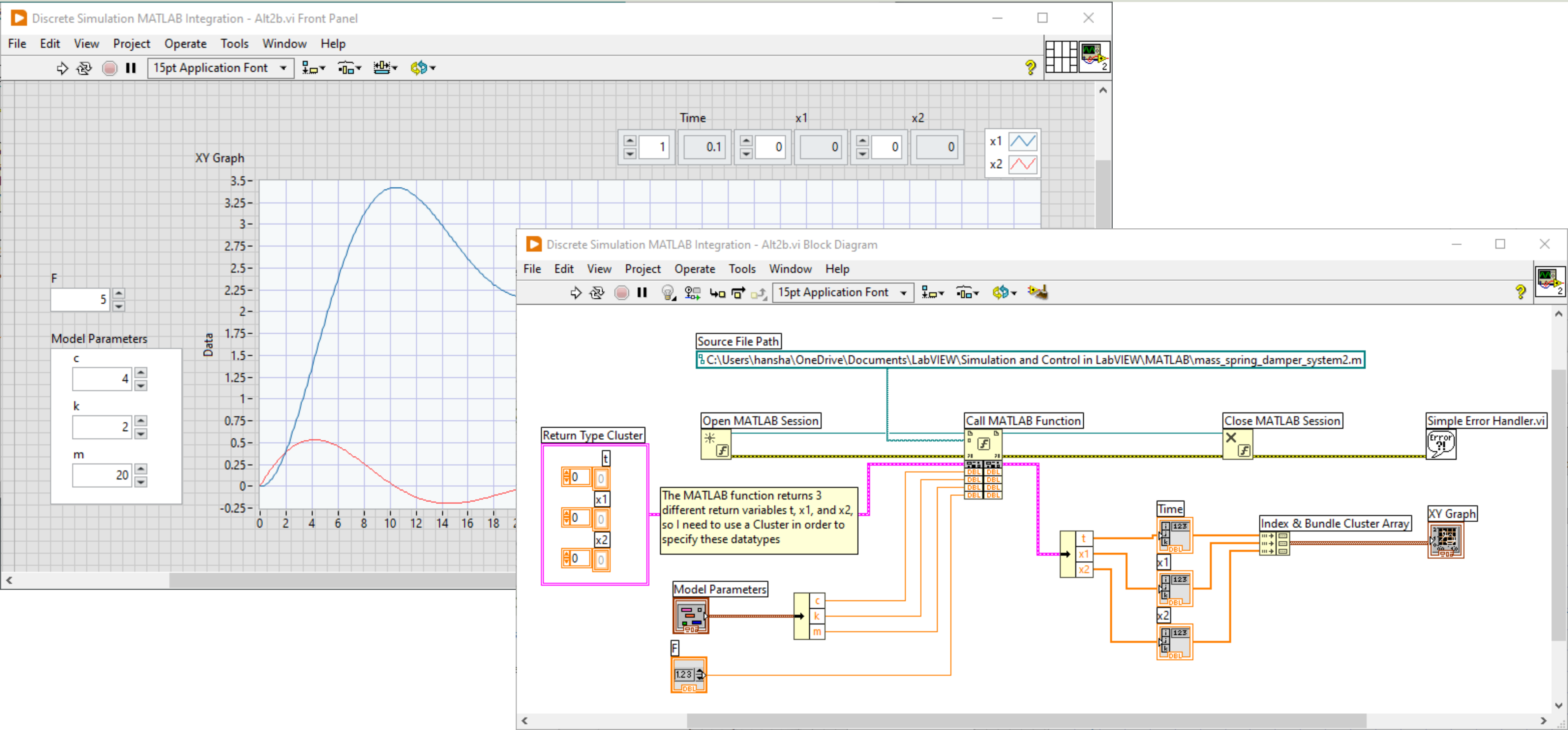
MATLAB Alt2 - Improved

- In the Alt2 Example we just got the results from the Simulation
- Let's also make it possible to set the Model Parameters, etc. from LabVIEW
- This will then be sent as arguments to the MATLAB Function

MATLAB Alt2 - Improved

```
function [t, x1, x2] = mass_spring_damper_system2(c, k_stiff, m, F)
% Simulation of Mass-Spring-Damper System

% Simulation Parameters
Ts = 0.1;
Tstart = 0;
Tstop = 60;
N = (Tstop-Tstart)/Ts; % Simulation length
t = Tstart : Ts : Tstop;
x1 = zeros(N,1);
x2 = zeros(N,1);
x1(1) = 0; % Initial Position
x2(1) = 0; % Initial Speed

% Simulation
for k=1:N
    x1(k+1) = x1(k) + Ts * x2(k);
    x2(k+1) = (-(Ts*k_stiff)/m) * x1(k) + (1 - (Ts*c)/m) * x2(k) + (Ts/m) * F;
end
```

MATLAB Alt2 - Improved

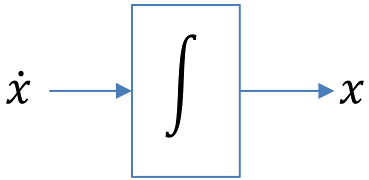


“Discrete Integrator”

Hans-Petter Halvorsen

“Discrete Integrator”

- In previous examples we needed to first find a discrete version of our differential equation(s) using Euler or other discretization methods
- In can be time-consuming and cumbersome to find these discrete differential equations
- So, we may want to create a “Discrete Integrator” and in that way we don’t need to solve or find discrete versions of the differential equation(s)



Integrator

Assume a general Differential Equation:

$$\dot{x} = f(t, x)$$

The purpose is to find x.

So, to find x we can Integrate $f(t, x)$:

$$x = \int f(t, x)$$

Discrete Integrator

Given

$$\dot{x} = f(t, x)$$

We use Euler to find a discrete version

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

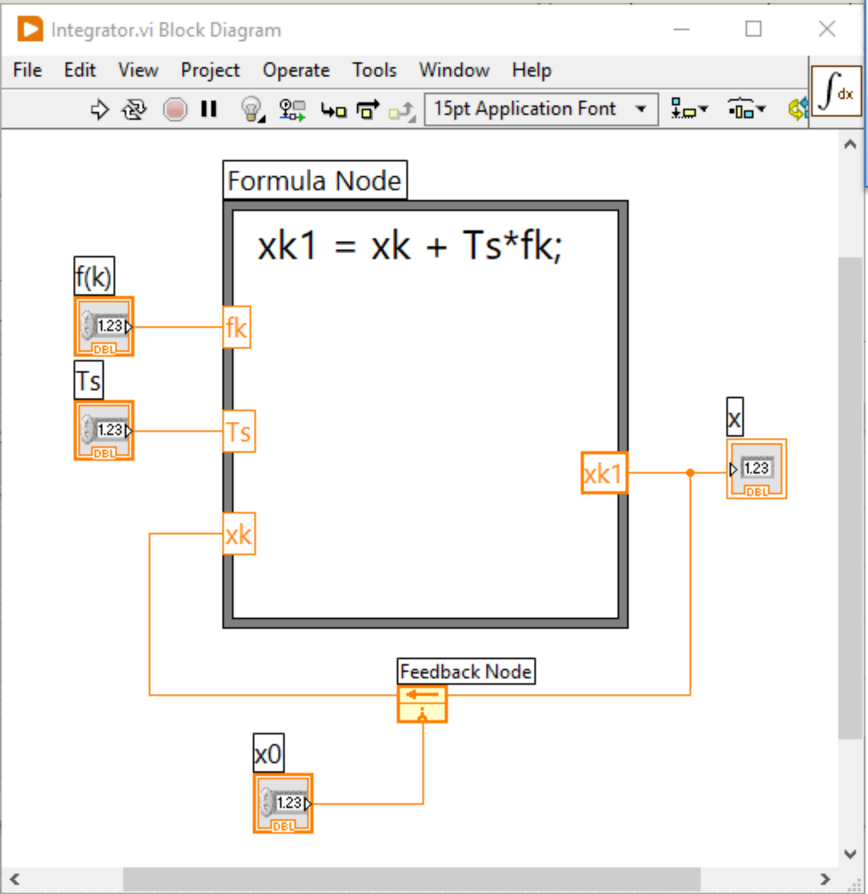
Then we get:

$$\frac{x(k+1) - x(k)}{T_s} = f(k)$$

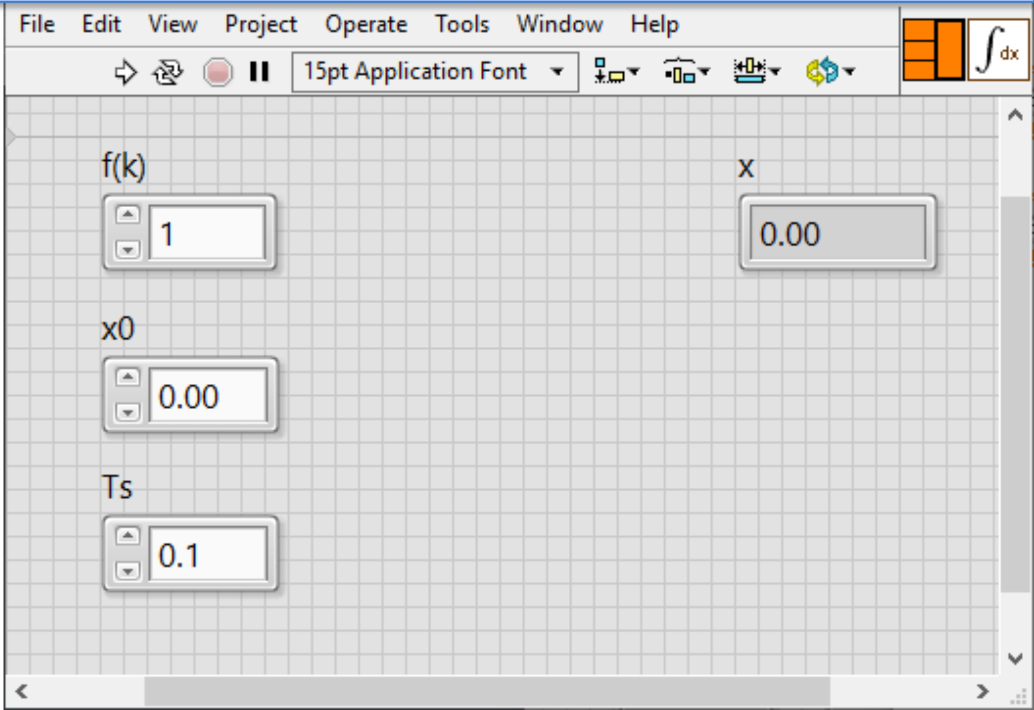
Which gives:

$$x(k+1) = x(k) + T_s f(k)$$

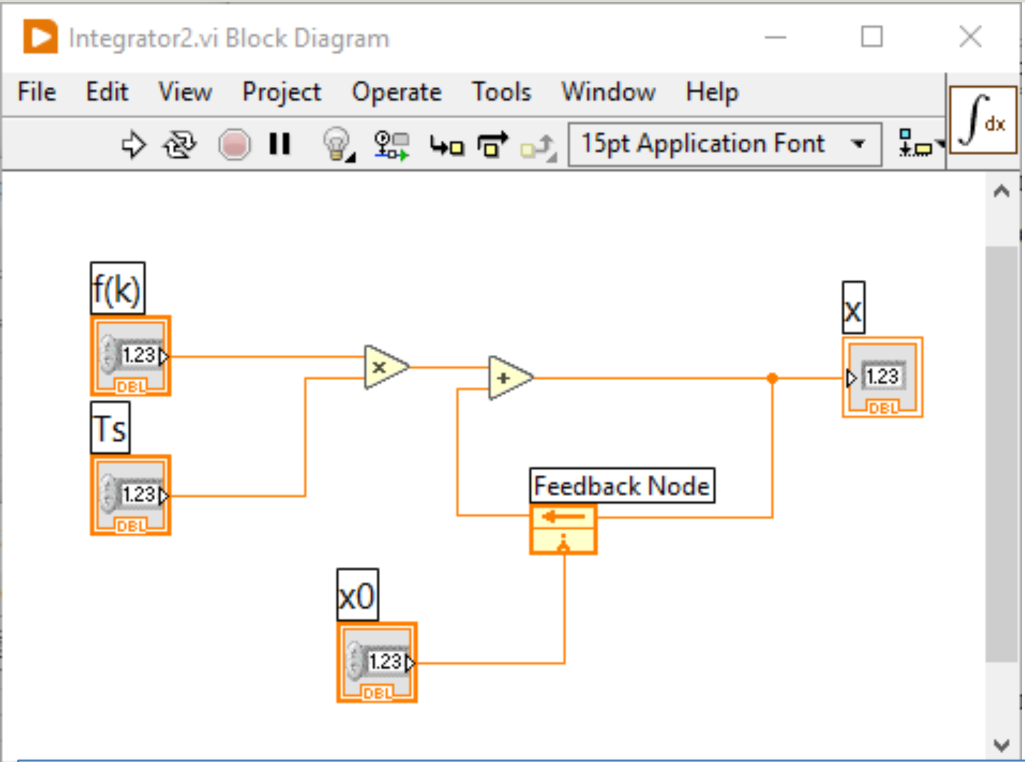
LabVIEW Integrator



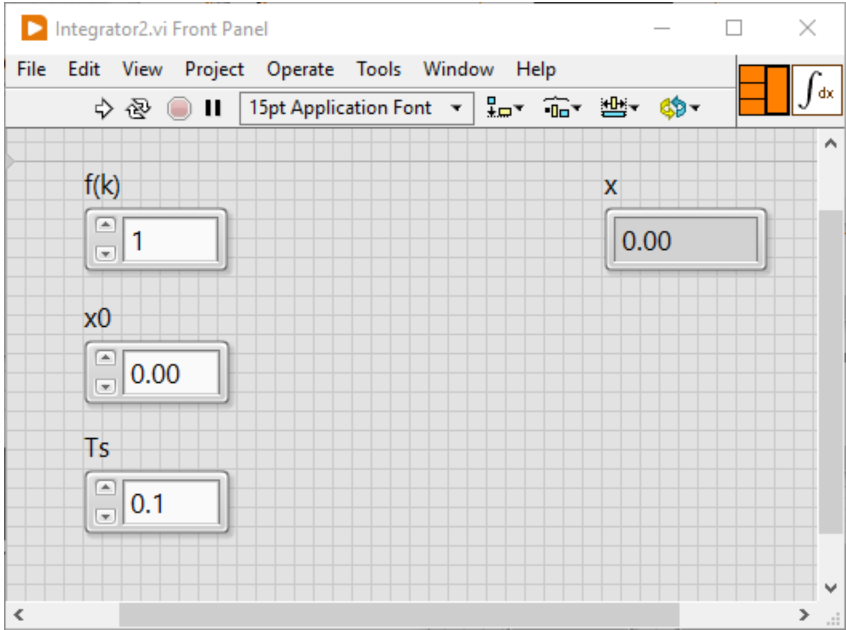
Her is the LabVIEW implementation of our Integrator. Basically, we can use this for all kind of differential systems, either we have one or many differential equations. Here is a Formula Node used, but you could have used pure LabVIEW code as well



LabVIEW Integrator – Alt2



Here we have used pure LabVIEW code instead of a Formula Node



We can also improve the Integrator block by implementing Saturation, i.e., include inputs for Lower Limit and Upper Limit

Simulation Example

We have the general differential equation:

$$\dot{x} = f(t, x)$$

Then

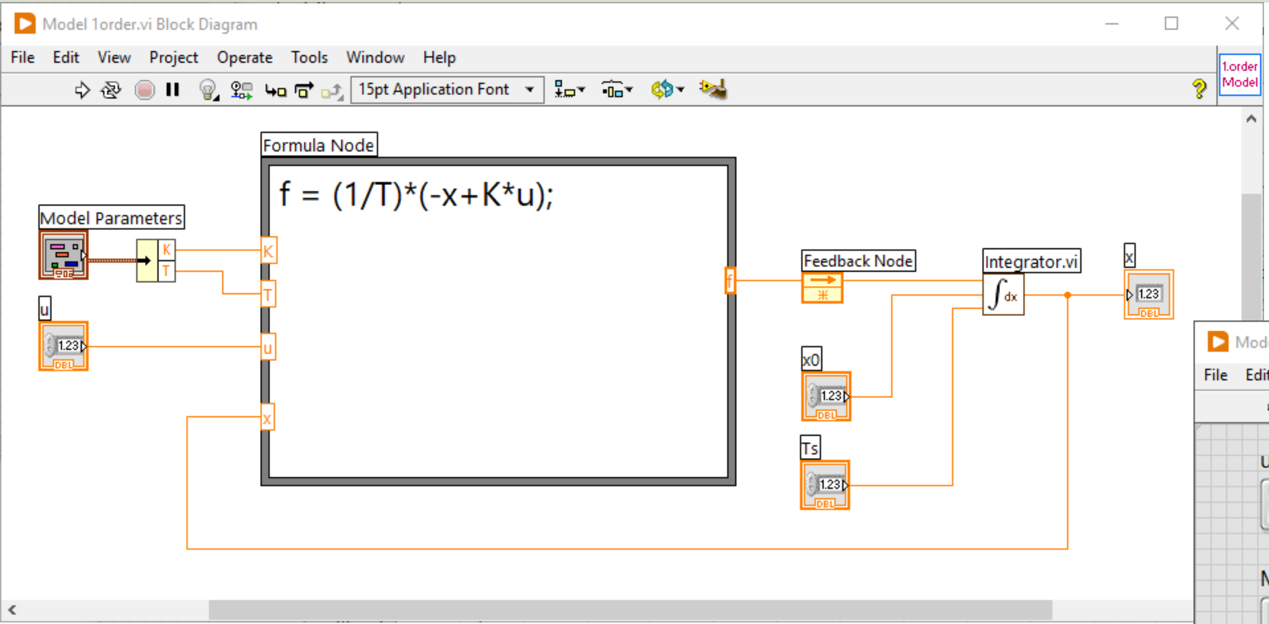
$$x = \int f(t, x)$$

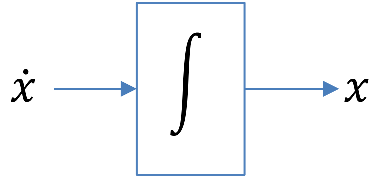
Let's test out this discrete Integrator on our standard 1.order system:

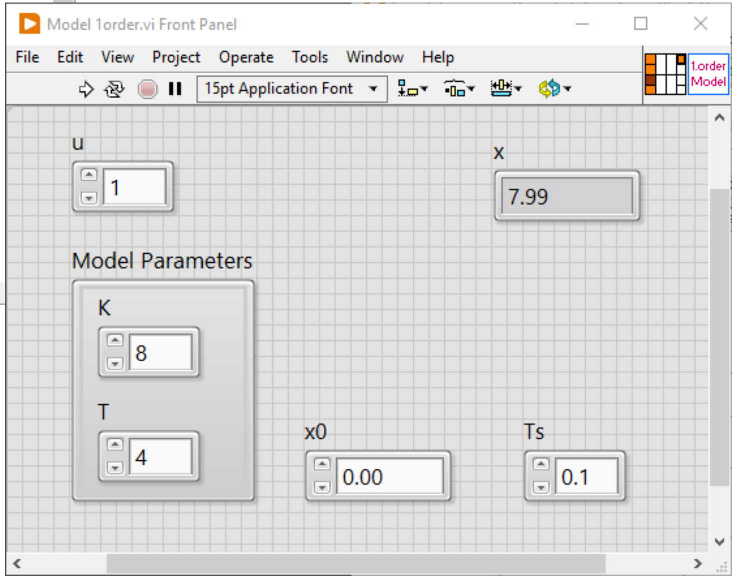
$$\dot{x} = \frac{1}{T}(-x + Ku)$$

$$\dot{x} = \frac{1}{T}(-x + Ku)$$

LabVIEW

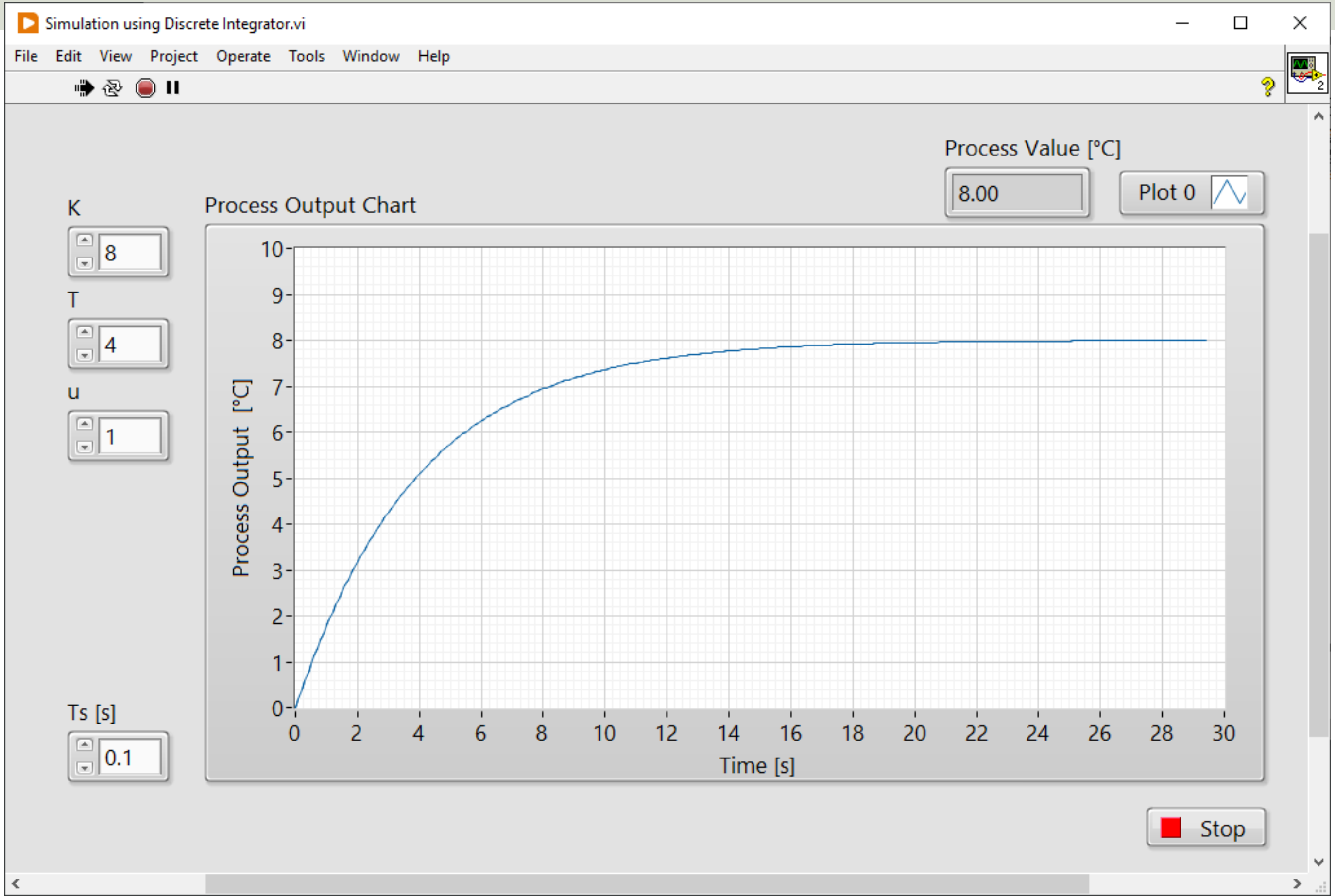






Now we only need to implement the right side of the differential equation. Then we use our new “Discrete Integrator” to find the solution in each iteration

LabVIEW

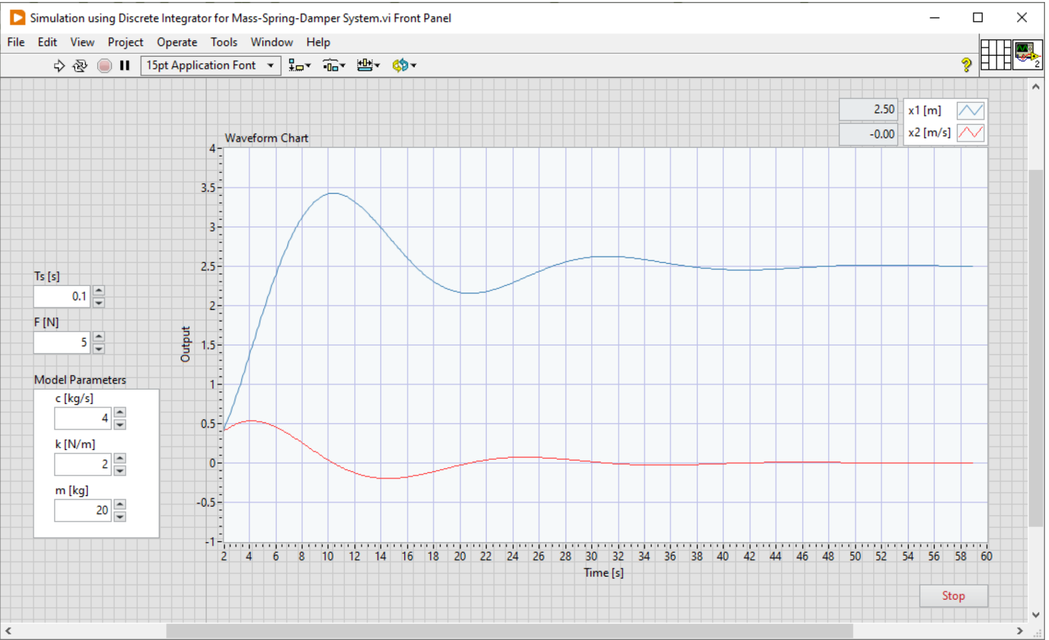


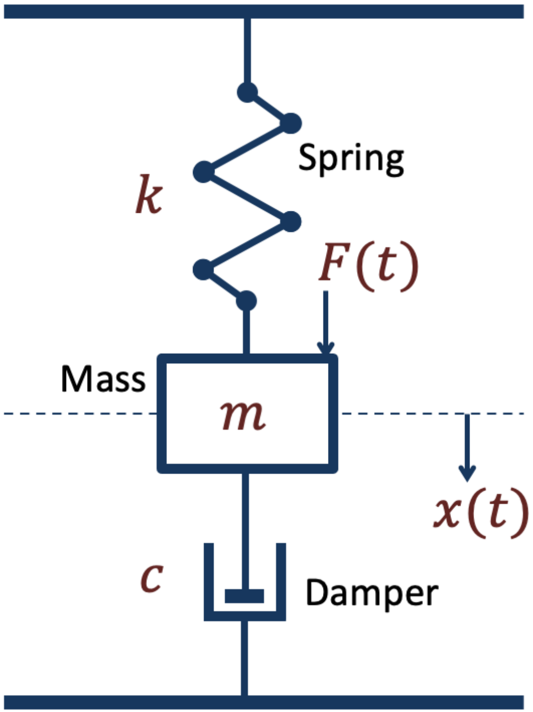
Simulation using Discrete Integrator.vi
File
Edit
View
Project
Operate
Tools
Window
Help
Process Value [°C]
8.00
Plot 0
K
8
T
4
u
1
Ts [s]
0.1
Process Output Chart
Process Output [°C]
Time [s]
0 1 2 3 4 5 6 7 8 9 10
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
Stop

2.order System

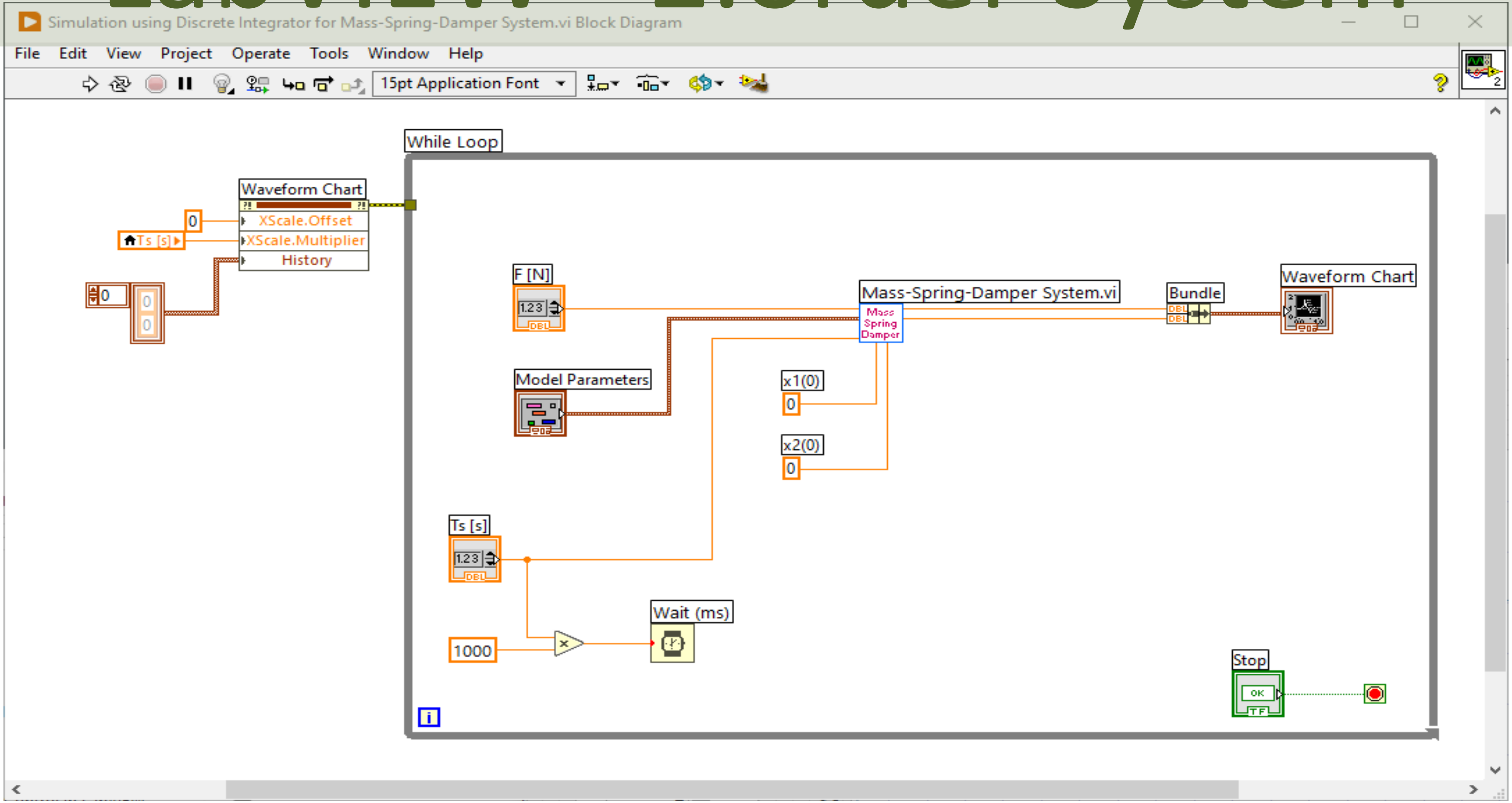
Let's test out new Integrator block on a 2.order system. We can use the previous Mass-Spring-Damper System:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$$

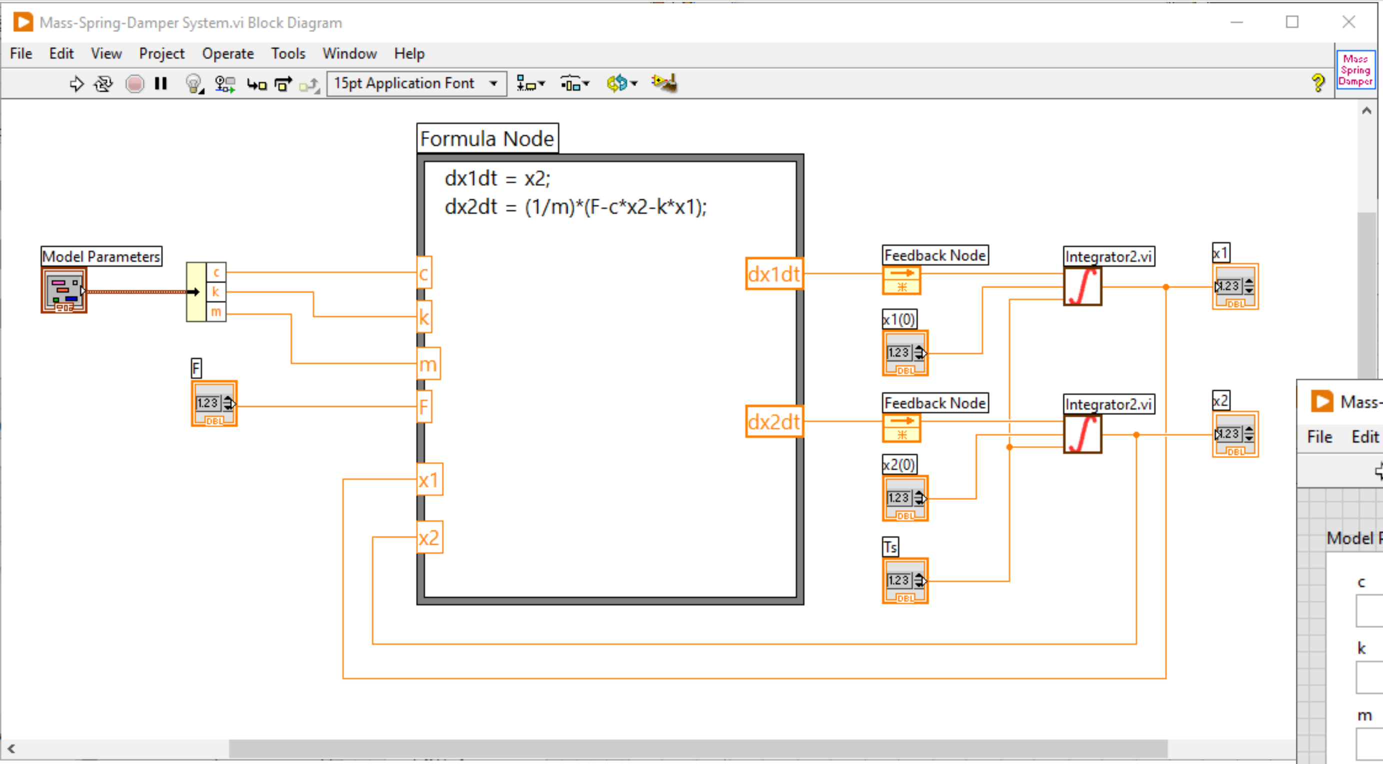


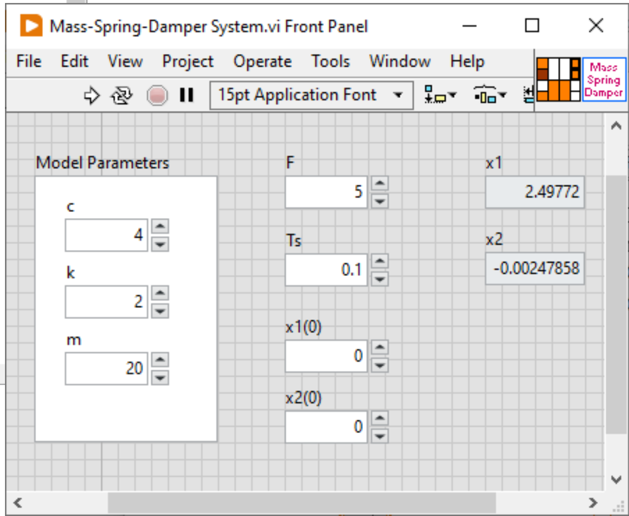


LabVIEW - 2.order System



LabVIEW - 2.order System





It is also possible to implement this in a more general way, i.e., create our own “ODE Solver.vi” that we used in one of the previous examples

1.order System with Time Delay

Hans-Petter Halvorsen

Time Delay

The equation for a Time Delay can be written as:

$$y = u(t - \tau)$$

Laplace:

$$y(s) = u(s)e^{-\tau s}$$

This gives the following Transfer function:

$$H(s) = \frac{y(s)}{u(s)} = e^{-\tau s}$$

Laplace Transformation pairs:

$$\dot{x} \Leftrightarrow sx(s)$$

$$u(t - \tau) \Leftrightarrow u(s)e^{-\tau s}$$

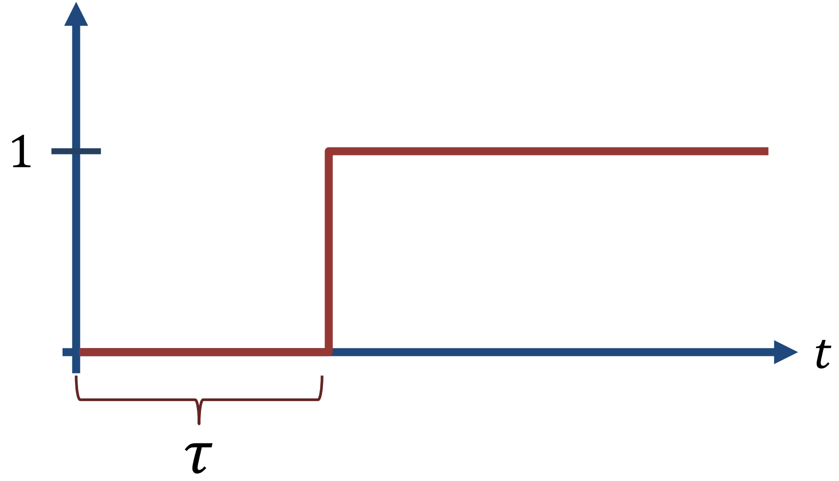
τ is the Time Delay

Time Delay

Transfer Function for Time Delay:

$$H(s) = e^{-\tau s}$$

Step Response for a Time Delay:



τ is the Time Delay in seconds

1.order System with Time Delay

A general 1. order System with Time Delay can be written as:

$$\dot{x} = -ax + bu(t - \tau)$$

Where $a = \frac{1}{T}$ and $b = \frac{K}{T}$

It can also be written like this:

$$\dot{x} = \frac{1}{T}[-x + Ku(t - \tau)]$$

Where K is the Gain, T is the Time constant and τ is the Time Delay

Transfer Function

Transfer Function for Time Delay:

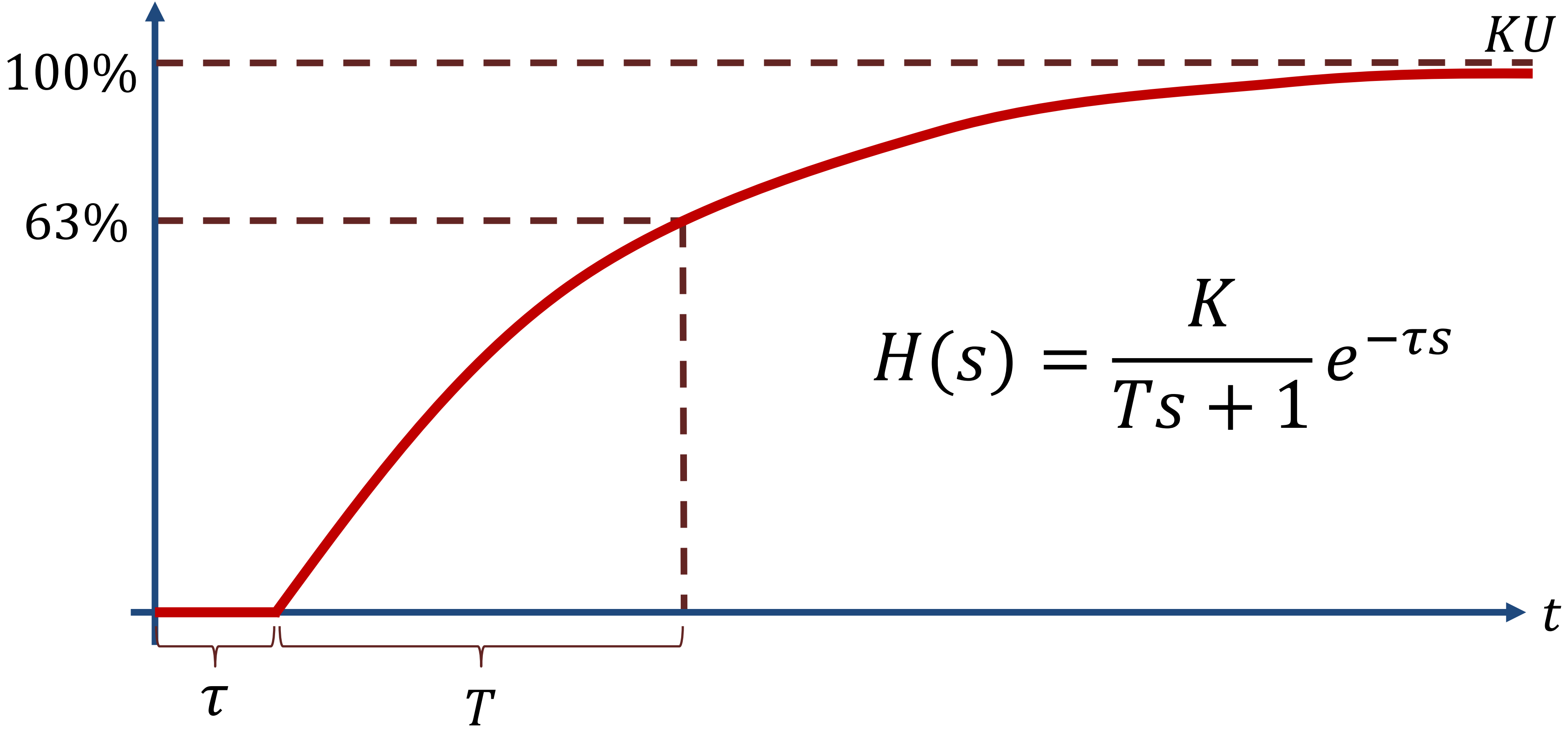
$$H(s) = \frac{y(s)}{u(s)} = e^{-\tau s}$$

1.order Transfer Function with Time Delay:

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{Ts + 1} e^{-\tau s}$$

Where K is the Gain, T is the Time constant and τ is the Time Delay

Step Response



$$H(s) = \frac{K}{Ts+1} e^{-\tau s}$$

Discrete Time Delay Function

Let's create a Discrete Time Delay Function in LabVIEW

The equation for a Time Delay can be written as:

$$y = u(t - \tau)$$

Discrete version:

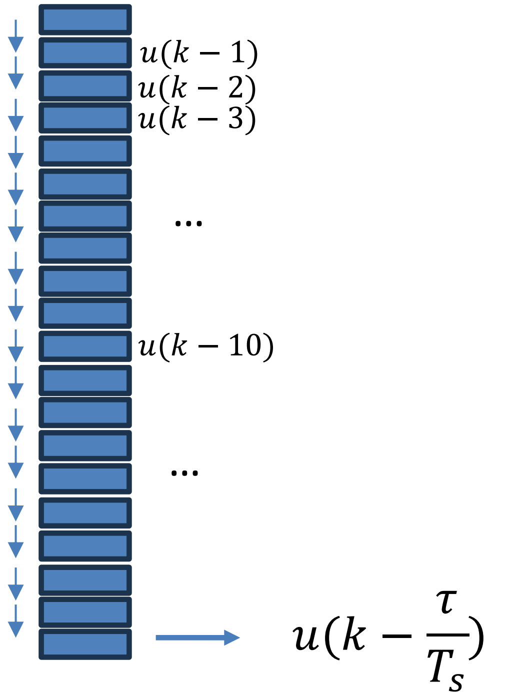
$$y(k) = u(k - \frac{\tau}{T_s})$$

Assuming, e.g., $\tau = 2s$ and $T_s = 0.1s$ we get $u(k - 20)$

This means we must remember the 20 previous samples of $u(k)$ in our calculations

Discrete Time Delay Function

$u(k)$



A Discrete Time Delay can be implemented as a FIFO queue.

FIFO – First In First Out.

The length of the queue will be

$$N = \frac{\tau}{T_s}$$

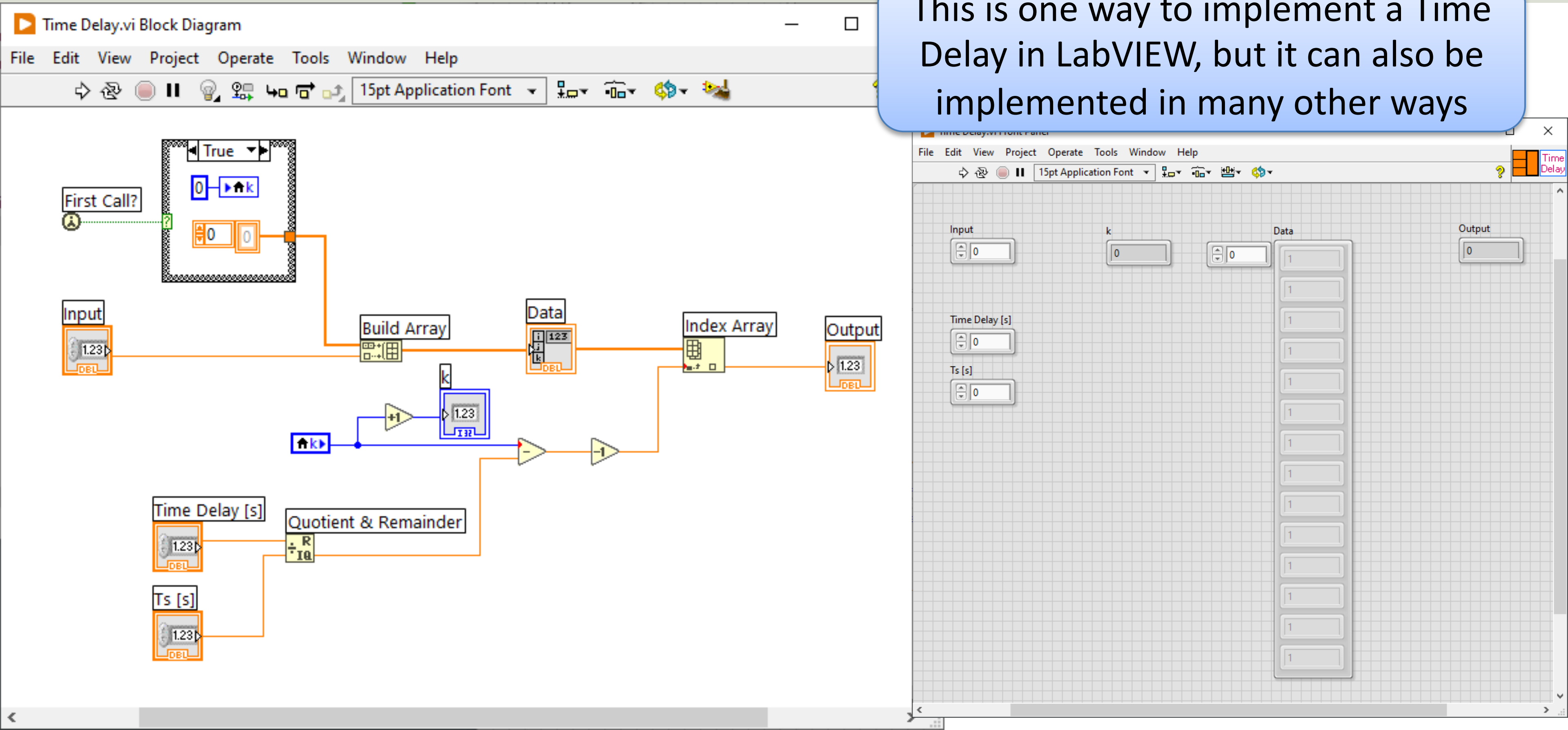
E.g.,

$\tau = 2s$ and $T_s = 0.1s$

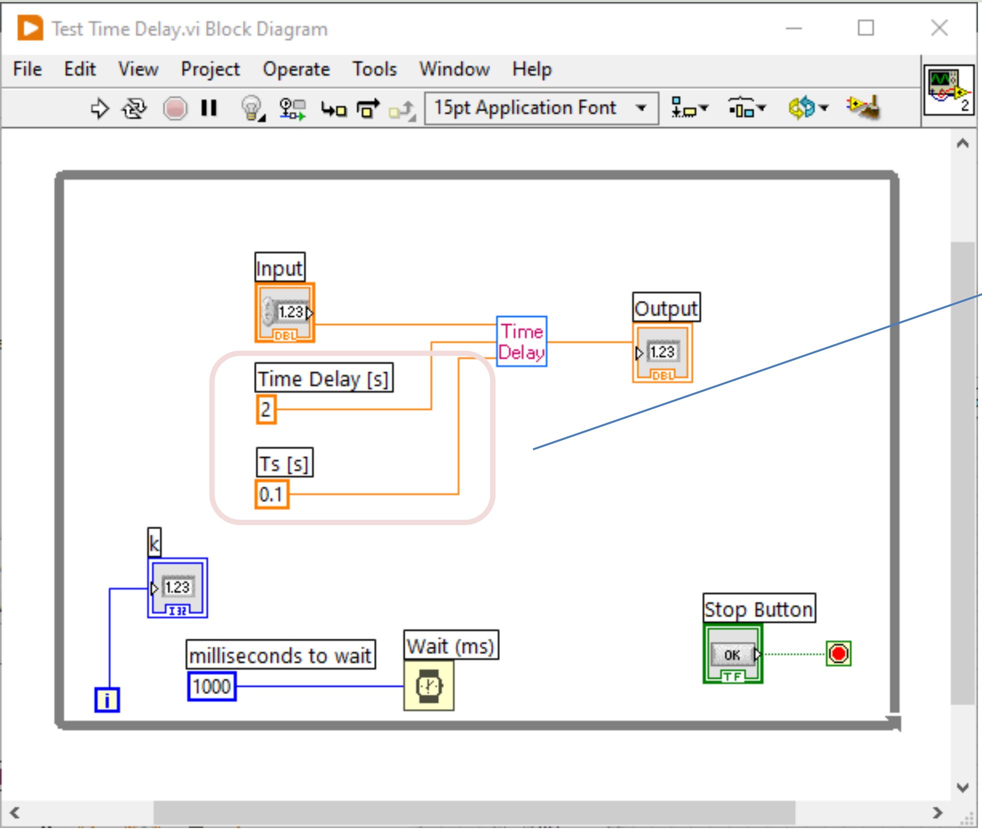
Then we get $u(k-20)$

LabVIEW Time Delay Function

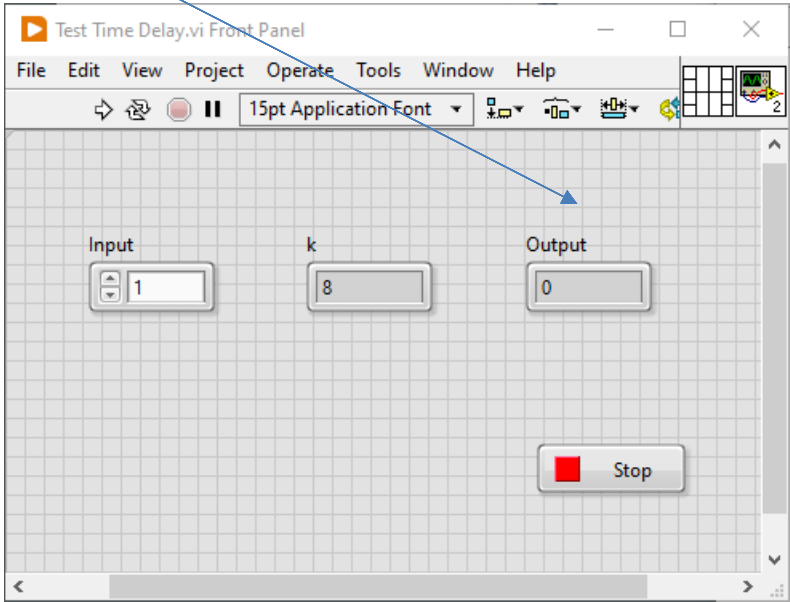
This is one way to implement a Time Delay in LabVIEW, but it can also be implemented in many other ways



Test Time Delay Function



When k=21 the Output will be 1 in this case



Discretization

We have the continuous differential equation: $\dot{x} = -ax + bu(t - \tau)$

We apply **Euler**: $\dot{x} \approx \frac{x(k+1)-x(k)}{T_s}$

Where $a = \frac{1}{T}$ and $b = \frac{K}{T}$

The discrete version of τ is $\frac{\tau}{T_s}$

Then we get:

$$\frac{x(k+1) - x(k)}{T_s} = -ax(k) + bu(k - \frac{\tau}{T_s})$$

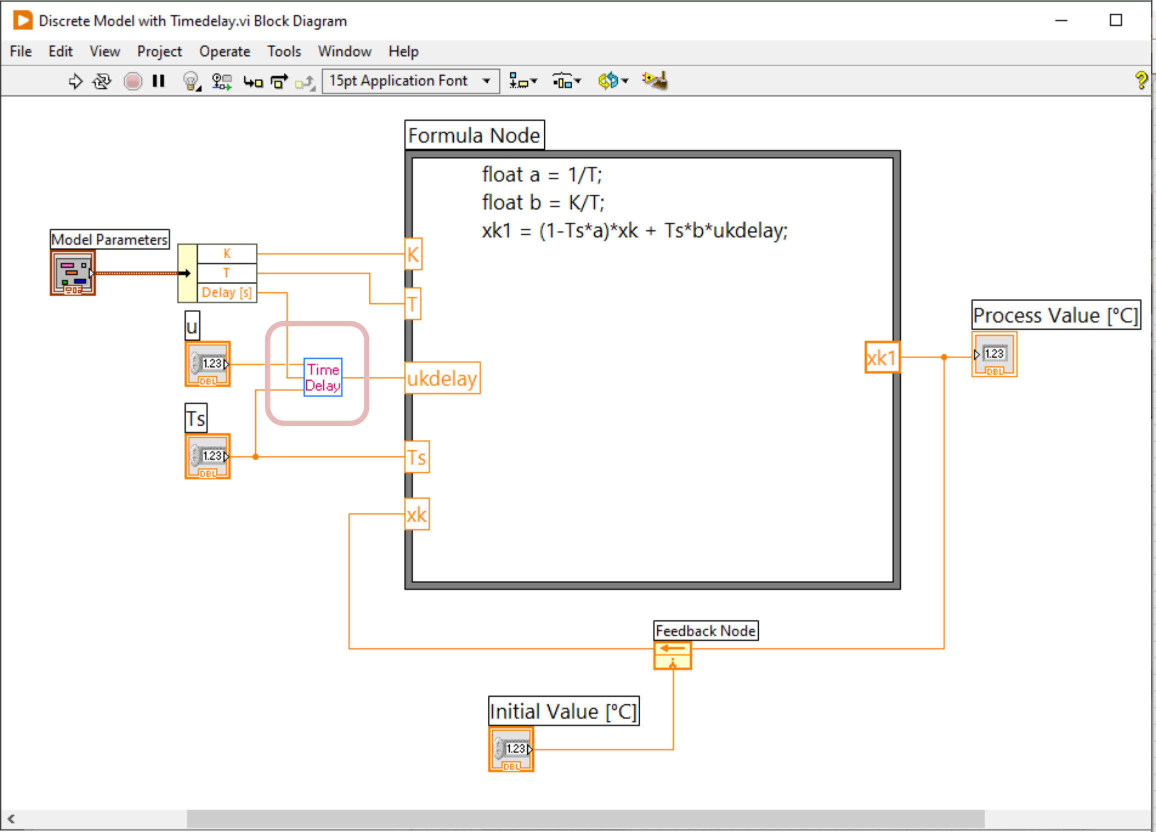
This gives the following discrete differential equation (difference equation):

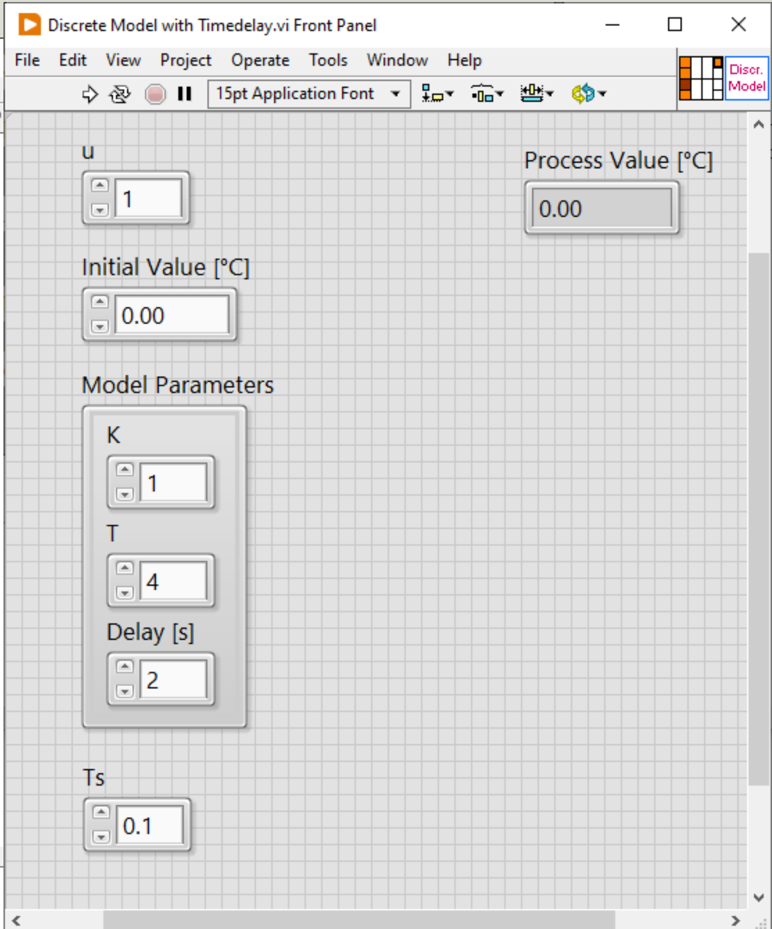
$$\boxed{x(k+1) = (1 - T_s a)x(k) + T_s bu(k - \frac{\tau}{T_s})}$$

Assuming $\tau = 2s$ and $T_s = 0.1s$ we get $u(k - 20)$

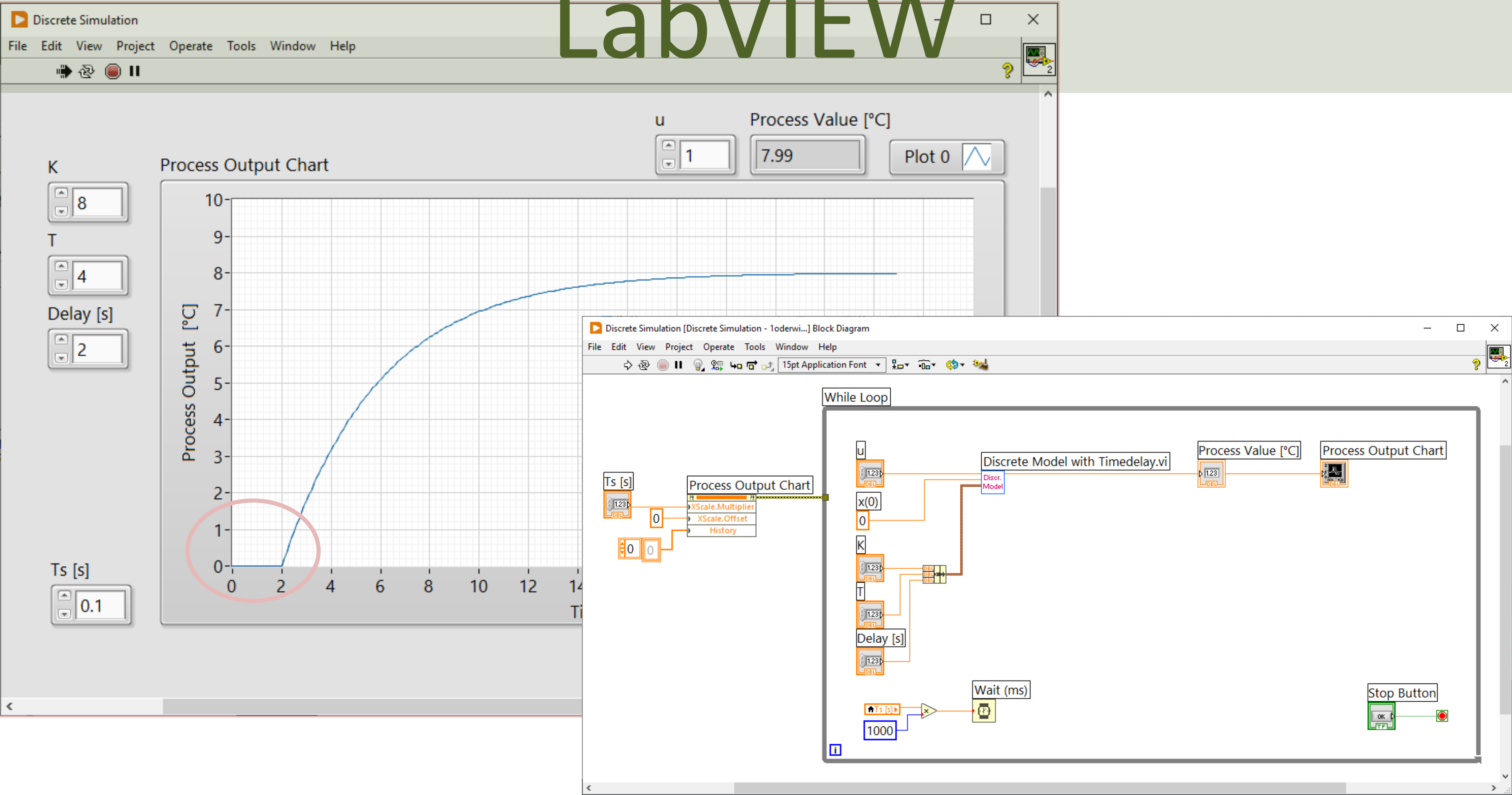
This means we must remember the 20 previous samples of $u(k)$

LabVIEW


Discrete Model with Timedelay.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Formula Node
float a = 1/T;
float b = K/T;
xk1 = (1-Ts*a)*xk + Ts*b*ukdelay;
Model Parameters
K
T
Delay [s]
u
Time Delay
ukdelay
Ts
xk
xk1
Process Value [°C]
Feedback Node
Initial Value [°C]


Discrete Model with Timedelay.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Discr. Model
u
1
Process Value [°C]
0.00
Initial Value [°C]
0.00
Model Parameters
K
1
T
4
Delay [s]
2
Ts
0.1

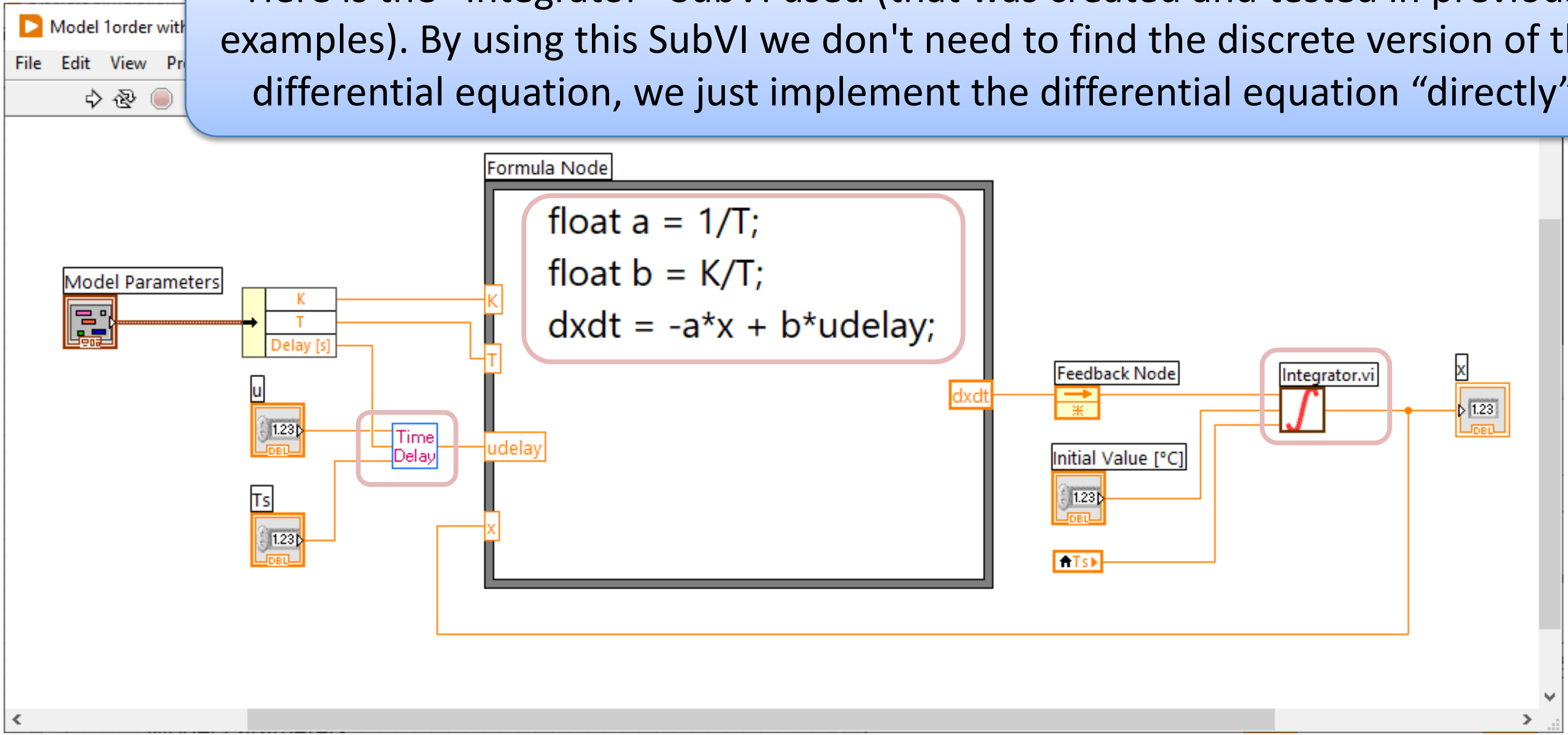
LabVIEW



Discrete Simulation
File Edit View Project Operate Tools Window Help
K
8
T
4
Delay [s]
2
Ts [s]
0.1
Process Output Chart
u
1
Process Value [°C]
7.99
Plot 0
Process Output [°C]
Discrete Simulation [Discrete Simulation - 1oderwi...] Block Diagram
15pt Application Font
While Loop
Ts [s]
Process Output Chart
XScale.Multiplier
XScale.Offset
History
u
x(0)
K
T
Delay [s]
Discrete Model with Timedelay.vi
Discr. Model
Process Value [°C]
Process Output Chart
Wait (ms)
1000
Stop Button

Alternative Model Implementation

Here is the “Integrator” SubVI used (that was created and tested in previous examples). By using this SubVI we don't need to find the discrete version of the differential equation, we just implement the differential equation “directly”



Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog