Article: Discretization of simulator, filter, and PID controller

Finn Haugen
TechTeach

10. May 2010

Preface

This article describes how to develop discrete-time algorithms for

- simulators of dynamic systems,
- lowpass filter,
- PID controller.

The algorithms are ready for implementation in a computer program using e.g. C-code.

1 Simple discretization methods

Often the simulator model, the filter, or the controller, is originally given as a continuous-time model in the form of a differential equation or a Laplace transfer function. The obtain a corresponding differential equation (ready for implementation in a computer program), you have to discretize the continuous-time model. This approximation can be made in many ways. My experience is that in most applications it is sufficient to apply one of the following (simple) methods, which are based on approximations of the time derivatives of the differential equation:

- **(Euler's) Forward differentiation method**, which is commonly used in developing simple simulators.
- **(Euler's) Backward differentiation method**, which is commonly used in discretizing simple signal filters and industrial controllers.

The Forward differentiation method and the Backward differentiation method will be explained in detail below. But you should at least have heard about some other methods as well, see below [**?**]:

- *Zero Order Hold (ZOH) method*: It is assumed that the system has a zero order hold element on the input side of the system. This is the case when the physical system is controlled by a computer via a DA converter (digital to analog). Zero order hold means that the physical input signal to the system is held fixed between the discrete points of time. The discretization method is relatively complicated to apply, and in practice you will probably use a computer tool (e.g. MATLAB or LabVIEW) to do the job.
- *Tustin's method*: This method is based on an integral approximation where the integral is interpreted as the area between the integrand and the time axis, and this area is approximated with trapezoids. (The Euler's methods approximates this area with a rectangle.)
- *Tustin's method with frequency prewarping, or Bilinear transformation*: This is the Tustin's method but with a modification so that the frequency response of the original continuous-time system and the resulting discrete-time system have exactly the same frequency response at one or more specified frequencies.

Some times you want to go the opposite way – transform a discrete-time model into an equivalent continuous-time model. Such methods will however not be described in this document.

The Forward differentiation method is somewhat less accurate than the Backward differentiation method, but it is simpler to use. Particularly, with nonlinear models the Backward differentiation method may give problems since it results in an implicit equation for the output variable, while the Forward differentiation method always gives an explicit equation (the nonlinear case will be demonstrated in an example).

Figure 1 illustrates both the Forward and the Backward differentiation methods. The Forward differentiation method can be seen as the following approximation of the time derivative of a time-valued function which here is denoted x:

$$\text{Forward differentiation method: } \dot{x}(t_k) \approx \frac{x(t_{k+1}) - x(t_k)}{T_s} \tag{1}$$

T_s is the time step, i.e. the time interval between two subsequent points of time. The name "Forward differentiation method" stems from the $x(t_{k+1})$ term in (1).

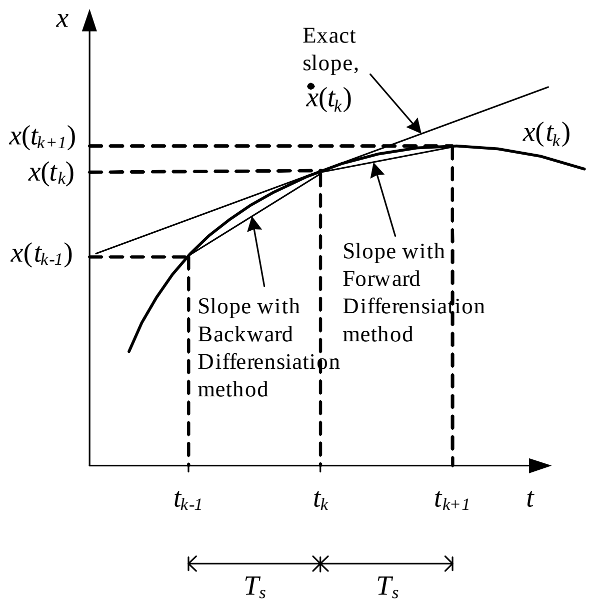
x
Exact slope, $\dot{x}(t_k)$
$x(t_{k+1})$
$x(t_k)$
$x(t_k)$
$x(t_{k-1})$
Slope with Forward Differensiation method
Slope with Backward Differensiation method
t_{k-1}
t_k
t_{k+1}
t
T_s
T_s

Figure 1: The Forward differentiation method and the Backward differentiation method

The Backward differentiation method is based on the following approximation of the time derivative:

$$\text{Backward differentiation method: } \dot{x}(t_k) \approx \frac{x(t_k) - x(t_{k-1})}{T_s} \tag{2}$$

The name "Backward differentiation method" stems from the $x(t_{k-1})$ term in (2), see Figure 1.

The examples in the subsequent sections demonstrate the application of the Forward differentiation method and the Backward differentiation method. It is also demonstrated how to get an equivalent differential equation from an original transfer function model or an integral equation (the time derivatives of the differential equation is then approximated with the Forward differentiation method or the Backward differentiation method).

2 Discretization of a process model to create a simulator

A simulator of a dynamic system, e.g. a motor, liquid tank, a ship etc., must of course be based on the mathematical model of the system.

Typically, the model is in the form of a nonlinear differential equation model. The Forward differentiation method may be used to discretize such nonlinear models.

As an example, let us discretize the following nonlinear model:

$$\dot{x}(t) = -K_1\sqrt{x(t)} + K_2 u(t) \tag{3}$$

where u is the input, x is the output, and K_1 and K_2 are parameters.[1] We will now derive a simulator algorithm or formula for $x(t_k)$. Let us first try applying the Backward differentiation method with time step h to the time derivative in (3):

$$\frac{x(t_k) - x(t_{k-1})}{T_s} = -K_1\sqrt{x(t_k)} + K_2 u(t_k) \tag{4}$$

$x(t_k)$ appears on both sides of (4). We say that $x(t_k)$ is given implicitly – not explicitly – by (4). Solving for for $x(t_k)$ in this implicit equation is possible, but a little difficult because of the nonlinear function (the square root). (If the difference equation was linear, it would be much easier to solve for $x(t_k)$.) In other cases, nonlinear functions may cause big problems in solving for the output variable, here $x(t_k)$.

Since we got some problems with the Backward differentiation method in this case, let us in stead apply the Forward differentiation method to the time derivative of (3):

$$\frac{x(t_{k+1}) - x(t_k)}{T_s} = -K_1\sqrt{x(t_k)} + K_2 u(t_k) \tag{5}$$

Solving for $x(t_{k+1})$ is easy:

$$x(t_{k+1}) = x(t_k) + T_s\left[-K_1\sqrt{x(t_k)} + K_2 u(t_k)\right] \tag{6}$$

Reducing each time index by one and using the simplifying notation $x(t_k) = x(k)$ etc. finally gives the simulation algorithm:

$$x(k) = x(k-1) + T_s\left[-K_1\sqrt{x(k-1)} + K_2 u(k-1)\right] \tag{7}$$

In general it is important that the time-step T_s of the discrete-time function is relatively small, so that the discrete-time function behaves approximately similar to the original continuous-time system. For the Forward differentiation method a (too) large time-step may even result in

[1] This can be the model of a liquid tank with pump inflow and valve outflow. x is the level. u is the pump control signal. The square root stems from the valve.

an unstable discrete-time system! For simulators the time-step T_s should be selected according to

$$T_s \leq \frac{0.1}{|\lambda|_{\max}} \tag{8}$$

Here $|\lambda|_{\max}$ is the largest of the absolute values of the eigenvalues of the model, which is the eigenvalues of the system matrix A in the state-space model $\dot{\underline{x}} = A\underline{x} + B\underline{u}$. For transfer function models you can consider the poles in stead of the eigenvalues (the poles and the eigenvalues are equal for most systems not having pol-zero cancellations). If the model is nonlinear, it must be linearized before calculating eigenvalues or poles.

In stead of, or as a supplementary using However, you may also use a trial-and-error method for choosing T_s (or f_s): *Reduce h until there is a negligible change of the response of the system if* T_s *is further reduced.* If possible, you should use a simulator of your system to test the importance of the value of T_s before implementation.

3 Discretization of a signal filter

A lowpass filter is used to smooth out high frequent or random noise in a measurement signal. A very common lowpass filter in computer-based control systems is the discretized first order filter – or time-constant filter. You can derive such a filter by discretizing the Laplace transfer function of the filter. A common discretization method in control applications is the Backward differentiation method. We will now derive a discrete-time filter using this method.

The Laplace transform transfer function – also denoted the continuous-time transfer function – of a first order lowpass filter is

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{T_f s + 1} = \frac{1}{\frac{s}{\omega_b} + 1} = \frac{1}{\frac{s}{2\pi f_b} + 1} \tag{9}$$

Here, u is the filter input, and y is the filter output. T_f [s] is the time-constant. ω_b is the filter bandwidth in rad/s, and f_b is the filter bandwidth in Hz. (In the following, the time-constant will be used as the filter parameter since this is the parameter typically used in filter implementations for control systems.)

Cross-multiplying in (9) gives

$$(T_f s + 1)\, y(s) = u(s) \tag{10}$$

Resolving the parenthesis gives

$$T_f s y(s) + y(s) = u(s) \tag{11}$$

Taking the inverse Laplace transform of both sides of this equation gives the following differential equation (because multiplying by s means time-differentiation in the time-domain):

$$T_f \dot{y}(t) + y(t) = u(t) \tag{12}$$

Let us use t_k to represent the present point of time – or discrete time:

$$T_f \dot{y}(t_k) + y(t_k) = u(t_k) \tag{13}$$

Substituting the time derivative by the Backward differentiation approximation gives

$$T_f \frac{y(t_k) - y(t_{k-1})}{T_s} + y(t_k) = u(t_k) \tag{14}$$

Solving for $y(t_k)$ gives

$$y(t_k) = \frac{T_f}{T_f + T_s} y(t_{k-1}) + \frac{T_s}{T_f + T_s} u(t_k) \tag{15}$$

which is commonly written as

$$y(t_k) = (1 - a)\, y(t_{k-1}) + a u(t_k) \tag{16}$$

with filter parameter

$$a = \frac{T_s}{T_f + T_s} \tag{17}$$

which has a given value once you have specified the filter time-constant T_f and the time-step T_s is given. (16) is the formula for the filter output. It is ready for being programmed. This filter is denoted the *exponentially weighted moving average (EWMA) filter*, but we can simply denote it a first order lowpass filter.

It is important that the filter time-step T_s is considerably smaller than the filter time-constant T_f, otherwise the filter may behave quite differently from the original continuous-time filter (9) from which it is derived. A rule of thumb for the upper limit of T_s is

$$T_s \leq \frac{T_f}{5} \tag{18}$$

4 Discretization of a PID controller

4.1 Computer based control loop

Figure 2 shows a control loop where controller is implemented in a computer. The computer registers the process measurement signal via an AD converter (from analog to digital). The AD converter produces a numerical value which represents the measurement. As indicated in the block diagram this value may also be scaled, for example from volts to percent. The resulting digital signal, $y(t_k)$, is used in the control function, which is in the form of a computer algorithm or program calculating the value of the control signal, $u(t_k)$.

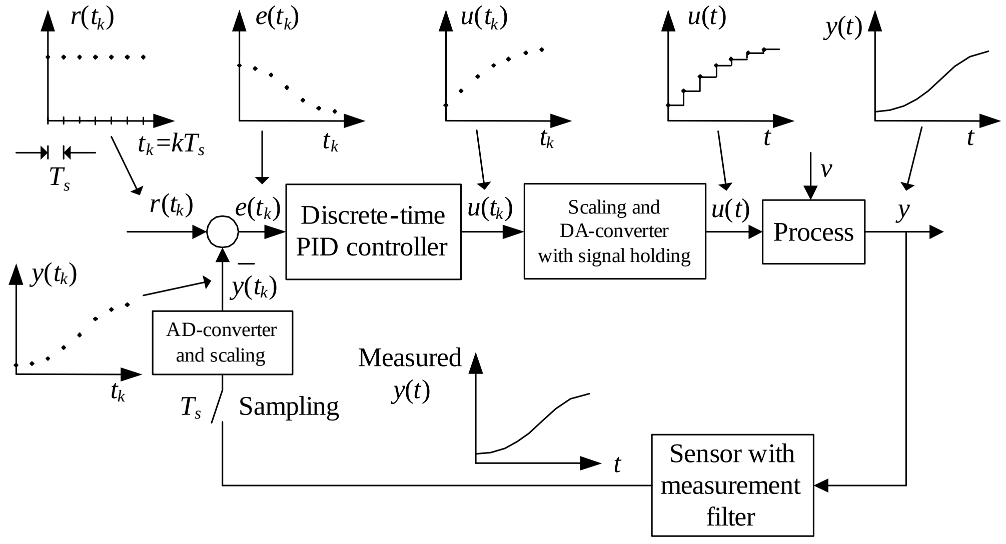

Figure 2: Control loop where the controller function is implemented in a computer

The control signal is scaled, for example from percent to milliamperes, and sent to the DA converter (from digital to analog) where it is held constant during the present time step. Consequently the control signal becomes a staircase signal. The time step or the sampling interval, T_s [s], is usually small compared to the time constant of the actuator (e.g. a valve) so the actuator does not feel the staircase form of the control signal. A typical value of T_s in commercial controllers is 0.1 s.

4.2 Development of discrete-time PID controller

The starting point of deriving the discrete-time PID controller is the continuous-time PID (proportional + integral + derivate) controller:

$$u(t) = u_0 + K_p e(t) + \frac{K_p}{T_i} \int_0^t e \, d\tau + K_p T_d \dot{e}(t) \tag{19}$$

where u_0 is the control bias or manual control value (to be adjusted by the operator when the controller is in manual mode), u is the controller output (the control variable), e is the control error:

$$e(t) = r(t) - y(t) \tag{20}$$

where r is the reference or setpoint, and y is the process measurement.

We will now derive a discrete-time formula for $u(t_k)$, the value of the control signal for the present time step. The discretization can be performed in a number of ways. Probably the simplest way is as follows: Differentiating both sides of (19) gives[2]

$$\dot{u}(t) = \dot{u}_0 + K_p \dot{e}(t) + \frac{K_p}{T_i} e(t) + K_p T_d \ddot{e}(t) \tag{21}$$

Applying the Backward differentiation method (2) to $\dot{u}$, $\dot{e}$, and $\ddot{e}$ gives

$$\begin{aligned}
\frac{u(t_k) - u(t_{k-1})}{T_s} &= \frac{u_0(t_k) - u_0(t_{k-1})}{T_s} && (22)\\
&\quad + K_p \frac{e(t_k) - e(t_{k-1})}{T_s} && (23)\\
&\quad + \frac{K_p}{T_i} e(t_k) && (24)\\
&\quad + K_p T_d \frac{\dot{e}(t_k) - \dot{e}(t_{k-1})}{T_s} && (25)
\end{aligned}$$

Applying the Backward differentiation method on $\dot{e}_f(t_k)$ and $\dot{e}_f(t_{k-1})$ in (22) gives

$$\begin{aligned}
\frac{u(t_k) - u(t_{k-1})}{T_s} &= \frac{u_0(t_k) - u_0(t_{k-1})}{T_s} && (26)\\
&\quad + K_p \frac{e(t_k) - e(t_{k-1})}{T_s} && (27)\\
&\quad + \frac{K_p}{T_i} e(t_k) && (28)\\
&\quad + K_p T_d \frac{\frac{e(t_k) - e(t_{k-1})}{T_s} - \frac{e(t_{k-1}) - e(t_{k-2})}{T_s}}{T_s} && (29)
\end{aligned}$$

[2]The time derivative of an integral is the integrand.

Solving for $u(t_k)$ finally gives the discrete-time PID controller:

$$\begin{aligned}
u(t_k) &= u(t_{k-1}) + [u_0(t_k) - u_0(t_{k-1})] && (30)\\
&\quad + K_p \left[e(t_k) - e(t_{k-1})\right] && (31)\\
&\quad + \frac{K_p T_s}{T_i} e(t_k) && (32)\\
&\quad + \frac{K_p T_d}{T_s} \left[e(t_k) - 2e(t_{k-1}) + e(t_{k-2})\right] && (33)
\end{aligned}$$

The discrete-time PID controller algorithm (30) is denoted the *absolute* or *positional* algorithm. Automation devices typically implements the *incremental* or *velocity* algorithm. because it has some benefits. The incremental algorithm is based on splitting the calculation of the control value into two steps:

1. First the *incremental control value* $\Delta u(t_k)$ is calculated:

$$\begin{aligned}
\Delta u(t_k) &= [u_0(t_k) - u_0(t_{k-1})] && (34)\\
&\quad + K_p \left[e(t_k) - e(t_{k-1})\right] && (35)\\
&\quad + \frac{K_p T_s}{T_i} e(t_k) && (36)\\
&\quad + \frac{K_p T_d}{T_s} \left[e(t_k) - 2e(t_{k-1}) + e(t_{k-2})\right] && (37)
\end{aligned}$$

2. Then the *total or absolute control value* is calculated with

$$u(t_k) = u(t_{k-1}) + \Delta u(t_k) \quad (38)$$

The summation (38) implements the (numerical) integral action of the PID controller.

The incremental PID control function is particularly useful if the actuator is controlled by an incremental signal. A step-motor is such an actuator. The motor itself implements the numerical integration (38). It is (only) $\Delta u(t_k)$ that is sent to the motor.

4.3 Implementing integrator anti windup

Large excitations of the control system, typically large disturbances or large setpoint changes, may cause the control signal to reach its maximum or minimum limits with the control error being different from zero. The summation in (38), which is actually a numerical integration, will then cause u to increase (or descrease) steadily – this is denoted *integral windup* – so that u may get a very high (or low) value. When the excitations are back to normal values, it may take a very long time before the large value of u is integrated back to a normal value (i.e. within 0 – 100%), causing the process output to deviate largely from the setpoint.

Preventing the windup is (not surprisingly) denoted *anti windup*, and it can realized as follows:

1. Calculate an intermediate value of the control variable $u(t_k)$ according to (38), but do not send this value to the DA (Digital-to-Analog) converter.
2. Check if this intermediate value is greater than the maximum value $u_{\max}$ (typically 100%) or less than the minimum value $u_{\min}$ (typically 0%). If it exceeds one of these limits, set $\Delta u(t_k)$ in (38) to zero.
3. Write $u(t_k)$ to the DA converter.

4.4 Implementing bumpless transfer between auto/manual modes

Suppose the controller is switched from automatic to manual mode, or from manual to automatic mode (this will happen during maintenance, for example). The transfer between modes should be bumpless, ideally. Bumpless transfer can be realized as follows:

- **Bumpless transfer from automatic to manual mode:** In manual mode it is only the manual (or nominal) control signal u_0 – adjusted by the operator – that controls the process. Manual mode is equivalent to setting the controller gain to zero (or multiplying the gain by zero). We assume here that the control signal $u(t_k)$ in (38) has a proper value, say u_{good}, so that the control error is small, immediately before the switch to manual mode. To implement bumpless transfer, set $u(t_{k-1})$ in (38) equal to u_{good} immediately after the switching moment.
- **Bumpless transfer from manual to automatic mode:** Nothing special has to be done during the switching except activting all term in (34) – (37).