Controller Implementation

Recall we have shown

$$U(s) = H(s)E(s) = \left[c_0 + \sum_i \frac{c_i}{s-a_i} \right] E(s)$$

where a_i are poles of $H(s)$, and $c_0, c_1, c_2 \ldots$ are PFE coefs, with $c_0 = \emptyset$ if $\rho(H) > \emptyset$ ($H(s)$ has more poles than zeros).

Let $$X_i(s) = \left[\frac{1}{(s-a_i)} \right] E(s)$$

then $$u(t) = c_0 e(t) + \sum c_i x_i(t)$$

where $e(t) = y_d(t) - y(t)$, and $x_i(t)$ are sol'ns of

$$\dot{x}_i(t) = a_i x_i(t) + e(t)$$

The discrete time stepping under which the computer and sensor/actuator electronics operate mean that $u(t)$ will be computed only at integer multiples of the sample interval, T_s.

i.e. at $t_K = KT_s$, for $K = \emptyset, 1, 2, \ldots$

Let $u_K = u(t_K) = u(KT_s)$, and $e_K = e(t_K) = e(KT_s)$

From above:

$$u_K = C_0 e_K + \sum c_i x_i(t_K)$$

We need to know how to evaluate $x_i(t_K)$, i.e. sol'n of

$$\dot{x}_i(t) = a_i x_i(t) + e(t) \quad \text{evaluated at } t = t_K$$

Focus on just a single one of these eq'ns, since they are identical except for coefs a_i:

$$\dot{x}(t) = a x(t) + \varepsilon(t)$$

(Let $\varepsilon(t) = e(t)$ here, to avoid confusion with e^{at})

Assume $\varepsilon(t)$ is a step of size ε_0, and $x(0) = x_0$

Then $$X(s) = \frac{x_0}{s-a} + \frac{\varepsilon_0}{s(s-a)}$$

$$\Rightarrow x(t) = e^{at} x_0 + \frac{1}{(-a)}\left(1 - e^{at}\right)\varepsilon_0$$

Note: $e(t)$ will Not generally be a step, even if $y_d(t)$ is!

But, the above is a useful intermediate result, as we will see next.

Here

$$x(t) = e^{at}x_0 + \left(\frac{-1}{a}\right)\left(1-e^{at}\right)\varepsilon_0$$

So $x(T_s) = e^{aT_s}x_0 + \left(\frac{-1}{a}\right)\left(1-e^{aT_s}\right)\varepsilon_0$

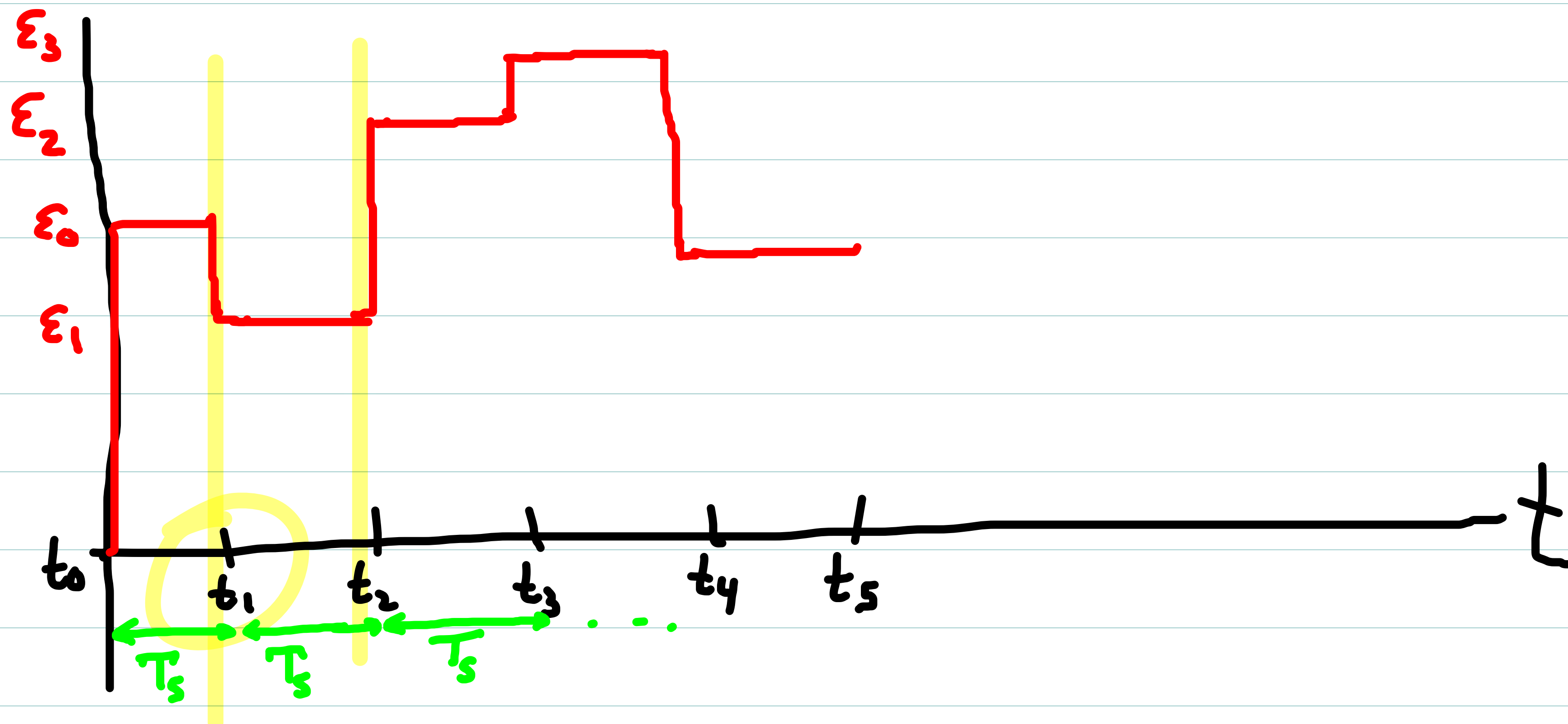
Let $\boxed{\alpha = e^{aT_s}}$, $\boxed{\beta = (1-\alpha)/(-a)}$

Then $x(T_s) = \alpha x_0 + \beta\varepsilon_0$

Let $x(T_s) = x_1 \Rightarrow \boxed{x_1 = \alpha x_0 + \beta\varepsilon_0 = x(t_1)}$

Now, how does this help generally?

Sampling of output at discrete times $t_K = KT_s$ means that error $e(t)$ will have a staircase graph



i.e. $e(t)$ will be constant with level ε_K on the interval $t_K \leq t < t_{K+1}$.

Note that at t_0, $e(t)$ does look like a step.

So, it is true for first sample interval that

$$X(t_1) = x_1 = \alpha X_0 + \beta \varepsilon_0 \quad \text{(as above)}$$

But what about subsequent time steps??

Exploit time invariance: when solving constant coeff DE's, the time called zero is <u>arbitrary</u>. All that matters is the initial cond'n and the time <u>elapsed</u> since initial time.

So, to get sol'n for next sample time t_2, we can "reset" the zero time to t_1, and use $X(t_1)$ as new initial cond'n.

Then, from new zero time $t = t_1$, error $e(t)$ looks like a step of height (initially) ε_1, so:

$$X(t_2) = X_2 = \alpha X_1 + \beta \varepsilon_1$$ by same logic as above

We can repeat this trick for all subsequent t_k:

$$\boxed{x_{k+1} = \alpha x_k + \beta \varepsilon_k}$$

where $x_k = x(t_k) = x(kT_s)$

$$\boxed{\alpha = e^{aT_s}, \quad \beta = (1-\alpha)/(-a)}$$

$\varepsilon_k = e(t_k)$ (error at k^{th} sample time)

We have thus shown that:

$$\boxed{x(t_{k+1}) = \alpha\, x(t_k) + \beta\, e(t_k)}$$

is an <u>iterative</u> algorithm for generating the <u>exact</u> sol'n for $x(t)$ at each of the sample times t_k, given the staircase structure of $e(t)$.

So generally:

$$u(t_K) = u_K = C_0 e(t_K) + \sum C_i x_i(t_K)$$

Where each $x_i(t_K)$ is computed iteratively using

$$x_i(t_{K+1}) = \alpha_i x_i(t_K) + \beta_i e(t_K)$$

where $\alpha_i = \exp[a_i T_s]$, $\beta_i = \left[\frac{1-\alpha_i}{(-a_i)}\right]$

and T_s is the sample interval.

Real-time implementation

$$u(t_K) = u_K = c_0 e(t_K) + \sum c_i x_i(t_K) \quad , \quad t_K \text{ is } K^{th} \text{ update time}$$

Where Each $x_i(t_K)$ is computed iteratively using

$$x_i(t_{K+1}) = \alpha_i x_i(t_K) + \beta_i e(t_K)$$

and $\Rightarrow$ $\alpha_i = \exp[a_i T_s]$, $\beta_i = \left[\frac{1-\alpha_i}{(-a_i)}\right]$

$\Rightarrow$ T_s is the sample interval (inverse of sample rate)

$\Rightarrow$ a_i are poles of $H(s)$

$\Rightarrow$ $c_0, c_1, c_2 \ldots$ are PFE coefs of $H(s)$

$\Rightarrow$ $e(t_K) = y_d(t_K) - y(t_K)$

Matlab code

Simple case: $H(s) = K \Rightarrow u(t) = Ke(t)$

```
function u=control(yd,y)

% define K (number!)
K=...

% compute u
e = yd-y;
u = K*e;

end
```

$H(s)$ with 1 pole

$$H(s) = K\frac{(s-z_c)}{(s-p_c)} = c_0 + \frac{c_1}{(s-p_c)}$$

```
function u=control(yd,y)

% define c0, c1, alpha, beta
```
(as numbers!)

```
c0=...
c1=...
alpha=...
beta=...
```

alpha = exp[p_c*T] here, and

Beta = (1-alpha)/(-p_c)

```
% compute u
e = yd-y;
u = c0*e+c1*x;

% update x
x = alpha*x+beta*e;

end
```

H(s) with 1 pole

$$H(s) = K\frac{(s-z_c)}{(s-p_c)} = c_0 + \frac{c_1}{(s-p_c)}$$

```
function u=control(yd,y)

% define c0, c1, alpha, beta
c0=...
c1=...
alpha=...
beta=...

% compute u
e = yd-y;
u = c0*e+c1*x;

% update x
x = alpha*x+beta*e;

end
```

Unfortunately, won't work as written!

We need the function to "remember" the values of x between calls.

Remember: Matlab functions (like C/C++ functions) have their own, private workspace (storage) for their variables, which is separate from the main workspace (main function).

Local variables in functions are <u>cleared</u> after the function runs.

Can prevent this clearing by declaring the variable to be "persistent" in Matlab ("static" in C/C++).

$$H(s) = K\frac{(s-z_c)}{(s-p_c)} = c_0 + \frac{c_1}{(s-p_c)}$$

```
function u=control(yd,y)

persistent x

% define c0, c1, alpha, beta
c0=...
c1=...
alpha=...
beta=...

% compute u
e = yd-y;
u = c0*e+c1*x;

% update x
x = alpha*x+beta*e;

end
```

$$H(s) = K\frac{(s-z_c)}{(s-p_c)} = c_0 + \frac{c_1}{(s-p_c)}$$

```
function u=control(yd,y)

persistent x

% define c0, c1, alpha, beta
c0=...
c1=...
alpha=...
beta=...

% compute u
e = yd-y;
u = c0*e+c1*x;

% update x
x = alpha*x+beta*e;

end
```

Still won't work!

x needs to be initialized, but only the 1^{st} time the function is called.

Matlab initializes a persistent variable as an empty array the first time the function is run

We can test for this, and set initial value of x to our pleasing: "isempty" function for test

Note: simplest to initialize x to zero, unless there is a compelling reason not to (very rare)

$$H(s) = K\frac{(s-z_c)}{(s-p_c)} = c_0 + \frac{c_1}{(s-p_c)}$$

"ZOH" zero order hold

```
function u=control(yd,y)

persistent x

if isempty(x)
   x=0;
end
```

initialize x first time

```
% define c0, c1, alpha, beta
c0=...
c1=...
alpha=...
beta=...

% compute u
e = yd-y;
u = c0*e+c1*x;

% update x
x = alpha*x+beta*e;

end
```

Works!

All our mathematical analysis ultimately boils down to 4 "magic numbers" that we plug into this standard template.

$$H(s) = 30\left[\frac{s+3}{s+9}\right] = 30 - \frac{180}{s+9} \quad , T_s = 0.1 \ (10 Hz)$$

```
function u=control(yd,y)

persistent x

if isempty(x)
   x=0;
end

% define c0, c1, alpha, beta
c0 = 30;
c1 = -180;
alpha = 0.4066;
beta = 0.0659;

% compute u
e = yd-y;
u = c0*e+c1*x;

% update x
x = alpha*x+beta*e;

end
```

Implementation of pole at origin

If $p_c = \phi$ (comp pole at origin), then clearly

$$\alpha = \exp[\phi T_s] = 1$$

in the implementation eq'n. However $\beta = \frac{(1-1)}{\phi}$ is indeterminate.

If we look more carefully at $\lim_{p_c \to \phi} \left[\frac{1 - \exp[p_c T_s]}{-p_c} \right]$

This yields the correct value $\beta = T_s$ for this case.

Thus for $\quad \dot{x}(t) = e(t)$

we have $\quad x(t_{k+1}) = x(t_k) + T_s e(t_k)$

i.e. $\quad x_{k+1} = x_k + T_s e_k$