Real-time implementation

$$u(t_K) = u_K = C_0 e(t_K) + \sum C_i x_i(t_K) \quad , \quad t_K \text{ is } K^{th} \text{ update time}$$

$t_K = K T_s$

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)
(secs) (Hz)

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

$\Rightarrow C_0, C_1, C_2 \ldots$ are PFE coefs of $H(s)$

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

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

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

persistent x

if isempty(x)
   x=0;
end

% 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
```

"zoh" zero order hold

initalize x first time

Works!

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

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

```
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
```

precomputed from $\alpha = \exp(p_c * T_s)$

$\beta = \frac{1}{p_c}(\alpha - 1)$

$p_c = -9$, $T_s = 0.1$

$-\frac{1}{9}$

Implementation of pole at origin

If $p_c = 0$ (comp pole at origin), then clearly

$$\alpha = \exp[0 \cdot T_s] = 1$$

in the implementation eq'n. However $\beta = \frac{(1-1)}{0}$ is indeterminate.

If we look more carefully at $\lim_{p_c \to 0} \left[\frac{1 - \exp[p_c T_s]}{-p_c}\right]$

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

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

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

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

$$H(s) = K\frac{(s-z_{c1})(s-z_{c2})}{(s-p_{c1})(s-p_{c2})} = C_0 + \frac{C_1}{s-p_{c1}} + \frac{C_2}{s-p_{c2}}$$

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

persistent x1 x2

if isempty(x1)
   x1=0;
   x2=0;
end

% define constants
% ...
%

% compute u
e = yd-y;
u = c0*e+c1*x1+c2*x2;

% update x
x1 = alpha1*x1+beta1*e;
x2 = alpha2*x2+beta2*e;

end
```

An implementation with 2 poles in $H(s)$, hence 2 Diff. eq'ns which need to be solved ($x_1(t_k), x_2(t_k)$)

$$H(s) = K\frac{(s-z_{c1})(s-z_{c2})}{(s-p_{c1})(s-p_{c2})} = C_0 + \frac{C_1}{s-p_{c1}} + \frac{C_2}{s-p_{c2}}$$

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

persistent x

if isempty(x)
   x=[0;0];
end

% define constants
% ...
%

% compute u
e = yd-y;
u = c0*e+c1*x(1)+c2*x(2);

% update x
x(1) = alpha1*x(1)+beta1*e;
x(2) = alpha2*x(2)+beta2*e;

end
```

Alternate implementation using Matlab arrays

a(1)
a(2)

b(1)
b(2)

(Can put α, β coefs into arrays also)

Extention to 3 or more poles in H(s) straightforward following same general pattern.

Note that, with two or more poles in $H(s)$, we could write the implementation equations in Matlab in the general form:

$$u = \underline{C} * \underline{x} + D * e$$

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}$$

Matlab ".*" array mult.

$$\underline{\dot{x}} = \underline{a} .* \underline{x} + \underline{b} * e$$

Where $\underline{x}$ is vector with all the different x_i variables

$$\underline{a} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_M \end{bmatrix} \qquad \underline{b} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_M \end{bmatrix}$$

M = # poles in $H(s)$

$$\underline{C} = [C_1 \; C_2 \cdots C_M] \qquad D = C_0$$

Even more generally:

$$u = \underline{C} * x + D * e$$

$$\dot{\underline{x}} = A * \underline{x} + \underline{b} * e$$

"State space" representation of (discretized) Controller dynamics

where

$$A = \text{diag}\{\alpha_1, \alpha_2, \cdots, \alpha_M\}$$

Special structure for A matrix

$$= \begin{bmatrix} \alpha_1 & & \emptyset \\ & \alpha_2 & \\ \emptyset & & \ddots \\ & & & \alpha_M \end{bmatrix}$$

$M \times M$ matrix

(M = # poles in $H(s)$)

```
function u = control(yd,y)
% Stub to illustrate most general form
% of discretized implementation equations

% Note: more efficient to define numerical
% components once, when we initialize x
  persistent x A B C D
  if isempty(x)
    A = []; % square matrix
    B = []; % column vector
    C = []; % row vector
    D = []; % scalar
        x = zeros(size(A,1),1);
        % Note: one state (xi variable) for each row of A
        %  (equivalently, for each pole in H(s))
  end

% Do the actual calculations
  e = yd - y;
  u = C*x + D*e;
  x = A*x + B*e;

end
```

fill in numbers for A, B, C, D

x = 0 initially

Matrix-vector version of Controller calculations

=

Different control strategies correspond to Different numerical values for matrices A, B, C, D

"State Space" Models

The diagonal state space form of the discretized controller equations is mirrored in the form of the continuous implementation eqns:

i.e. $$u(t) = c_0 e(t) + \sum_{i=1}^{M} c_i x_i(t)$$

$$\dot{x}_i(t) = a_i x_i(t) + e(t) \qquad i = 1, \ldots, M$$

$$\Rightarrow \quad u(t) = C\underline{x}(t) + De(t)$$
$$\dot{\underline{x}}(t) = A\underline{x}(t) + Be(t)$$

where $C = [c_1\ c_2 \ \ldots\ c_M] \qquad D = c_0$

$$A = \text{diag}\{a_1, a_2, \ldots, a_m\} \quad \text{and}$$

$$B = [1\ 1\ 1\ \ldots\ 1]^T \quad \text{(column vector)}$$

- This diagonal form for the state-space equations for $H(s)$ (and its discretization) is known as the "modal form".
- However, it is not the only possible state-space form we could use for $H(s)$ and its discretization.
- We have already seen that there are many (infinitely!) different state-space models for a given transfer f'n.
- One other form we have examined is the "companion form", where the values in A, B, C, D come from coefs of numerator+denom. polys of $H(s)$, instead of pole locations and residues.

Companion form (review)

$$H(s) = \frac{b_{n-1}s^{n-1} + \cdots + b_1 s + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 1 \\ -a_0 & -a_1 & \cdots & \cdots & \cdots & -a_{n-1} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} b_0 & b_1 & \cdots & b_{n-1} \end{bmatrix}$$

=

Note that $A_1 = A^T$, $B_1 = C^T$, $C_1 = B^T$

defines an alternate, equivalent, companion form.

General State Space Models

- Modal and companion forms are two common state-space models for a transfer function – again, there are an infinite number of other possible models
  - Discretization is easy for modal form, so why would we want to consider alternate forms?

    ⇒ When H(s) has repeated or complex poles, the diagonal modal form is not always possible

    ⇒ Thus we need to consider discretization of more general state-space models for H(s).

Suppose

$$\dot{\underline{x}} = A_H \underline{x} + B_H \underline{e}$$
$$\underline{u} = C_H \underline{x} + D_H \underline{e}$$

Arbitrary state-space model for $H(s)$.

is any state-space model for $H(s)$. Then the corresponding discrete equivalent is

$$\underline{u}_k = C_d \underline{x}_k + D_d \underline{e}_k$$

$$\underline{x}_{k+1} = A_d \underline{x}_k + B_d \underline{e}_k$$

For ZOH discretization with sample interval T_s.

$$A_d = e^{A_H T_s} \qquad C_d = C_H$$

$$B_d = A_H^{-1}(A_d - I)B_H \qquad D_d = D_H$$

(compare with $\alpha = e^{aT_s}$, $\beta = \left(\frac{1}{a}\right)(\alpha - 1)$ in scalar case)

$$A_d = e^{A_H T_s} \qquad B_d = A_H^{-1}\left[A_d - \mathrm{II}\right]B_H$$

recall

e^{At} is the "matrix exponential" function

$$e^{A_H t} = \mathcal{L}^{-1}\left\{(s\mathrm{II} - A_H)^{-1}\right\}$$

Matlab function "expm"

which is an $n \times n$ matrix whose entries are scalar exponential functions determined by the eigenvalues of A_H (which are the same as the poles of $H(s)$)

=

Note: if A_H is singular, then A_H^{-1} DNE and B_d must be calculated using a limiting process similar to scalar case (we'll use Matlab for this).

Example I

(1)

$$H(s) = \frac{5(s+1)^2}{(s+5)^2} = 5 - \frac{40s+120}{s^2+10s+25}$$

Can take

a) $A_H = \begin{bmatrix} 0 & 1 \\ -25 & -10 \end{bmatrix}$ $B_H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $C_H = [-120 \ -40]$ $D_H = 5$

companion form

for $T_s = 0.1$

$$A_d = \text{expm}[.1 A_H] = \begin{bmatrix} 0.91 & 0.061 \\ -1.52 & 0.30 \end{bmatrix}$$

$$B_d = A_H^{-1}[A_d - I]B_H = \begin{bmatrix} .0036 \\ .061 \end{bmatrix}$$

$$C_D = [-120 \ -40] \qquad D_D = 5$$

Example I, cont

(1)

$$H(s) = \frac{5(s+1)^2}{(s+5)^2} = C_H(sI - A_H)^{-1}B_H$$

b) Suppose we use instead

$$A_H = \begin{bmatrix} -5 & 1 \\ 0 & -5 \end{bmatrix} \quad B_H = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad C_H = \begin{bmatrix} 80 & -40 \end{bmatrix} \quad D_H = 5$$

[This is a "block modal" form for A_H]

Then here

$$A_d = \begin{bmatrix} 0.61 & 0.061 \\ 0 & 0.61 \end{bmatrix} \quad B_d = \begin{bmatrix} .0036 \\ .079 \end{bmatrix}$$

$$C_D = \begin{bmatrix} 80 & -40 \end{bmatrix} \quad D_D = 5$$

And this will yield exactly same result as a) above.

Example II:

2.) $H(s) = \frac{5(s+1)^2}{s^2+10s+100} = 5 - \frac{40s+495}{s^2+10s+100}$

a.) Can take

$$A_H = \begin{bmatrix} 0 & 1 \\ -100 & -10 \end{bmatrix} \quad B_H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

companion form

$$C_H = \begin{bmatrix} -495 & -40 \end{bmatrix} \quad D_H = 5$$

then

$$A_D = \begin{bmatrix} 0.66 & 0.053 \\ -5.33 & 0.13 \end{bmatrix} \quad B_D = \begin{bmatrix} .0034 \\ .0534 \end{bmatrix}$$

$C_D = C_H$, $D_D = D_H$

Example II, cont

2) $H(s) = \dfrac{5(s+1)^2}{s^2 + 10s + 100}$ ~ poles at $-5 \pm \frac{10\sqrt{3}}{2} j$

b) alternately cont'n

$$A_H = \begin{bmatrix} -5 & \frac{10\sqrt{3}}{2} \\ \frac{-10\sqrt{3}}{2} & -5 \end{bmatrix} \quad B_H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Complex block modal form →

$$C_H = [-34 \quad -40] \quad D_H = 5$$

and then

$$A_d = \begin{bmatrix} .393 & .462 \\ -.462 & .393 \end{bmatrix} \quad B_d = \begin{bmatrix} .0295 \\ .0704 \end{bmatrix}$$

$$C_d = C_H \qquad D_D = D_H$$

and again, this would be completely equivalent to the model in a) above

Which form to choose?

- The ultimate question is which form is going to yield the best accuracy given finite precision of computer calculation
  - Matlab's "c2d" function does attempt to generate matrices with this goal in mind
  - We will consider uses of this function, and other techniques to improve the accuracy of the discretization, in the next lecture.