

ELE3105 Computer Controlled Systems

# Module 7 – State Variables

# State Variables

- Basic idea: The more information we have about the dynamic behaviour of a system, the better we can control it.
- Usually the information is there for the asking: eg for controlling a robot arm's angular position, we could measure motor angle, motor speed, motor current, etc.

# State Variables

- We may want to control several things — eg not just the robot arm's position, but also the velocity profile (so that we don't break things!)
- So we have several quantities, each a function of time → represent as a vector.

# State Variables

- Represent the state of a system at an instant of time as a **state vector**. It will be a function of time:

$$\mathbf{x}(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_N(t) \end{pmatrix} \quad (1)$$

- The  $x_k$ 's are the state variables.
- At any time, a system is defined by its state.

## Aside...

- This idea finds applicability elsewhere, eg *speech recognition* in signal processing.
- Consider speech as a sequence of waveforms, with the particular waveform observed over a short period of time being the current state.
- Here, the particular sequence of states that a system goes through defines a word (by analogy, consider words being made up of syllables).

# States

- So these states are a function of time.
- We could observe the states using **several A/D converters** (one for each state), say
  - A shaft encoder or potentiometer to measure position.
  - A tachogenerator to measure velocity.
  - A series (shunt) resistor to measure current.
  - etc
- The **model** then aims to define the states and the sequence of states (how we get from one state at one instant to the next state at the next time instant).

# State Variables

- In physical systems, many variables are derivatives of one another.
- eg, velocity is the derivative of position or

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

- Differential equations are important in electrical circuit theory (voltage and current relationships etc).

# Matrices & Vectors

- Because we have a **vector** of states, we need some matrix/vector theory  
...
- By convention, vectors are **column vectors**
- I will use **capital bold font** for matrices such as  $\mathbf{X}$  and  $\mathbf{\Gamma}$ , and **lower case bold font** for vectors such as  $\mathbf{u}$  and  $\phi$ .
- Compare to  $X$ ,  $\Gamma$ ,  $u$  and  $\phi$ .

# Matrix Multiply

Consider an  $M \times N$  matrix:

$$\begin{array}{c} M \text{ rows} \\ \left\{ \begin{array}{cccc} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \cdot & \cdots & \cdots & \cdot \\ x_{M1} & x_{M2} & \cdots & x_{MN} \end{array} \right. \end{array}$$

$N$  columns

# Matrix Multiply

Multiply matrix  $\mathbf{A}$  ( $M \times P$ ) by  $\mathbf{B}$  ( $P \times N$ ) to give  $\mathbf{X}$  ( $M \times N$ ):

$$\begin{pmatrix} a_{11} & \cdots & a_{1P} \\ \cdot & \cdots & \cdot \\ a_{i1} & \cdots & a_{iP} \\ \cdot & \cdots & \cdot \\ a_{M1} & \cdots & a_{MP} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1N} \\ \cdot & \cdots & \cdot & \cdots & \cdot \\ b_{P1} & \cdots & b_{Pj} & \cdots & b_{PN} \end{pmatrix}$$

$$= \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ \cdot & \cdots & \cdot \\ \cdot & x_{ij} & \cdot \\ \cdot & \cdots & \cdot \\ x_{M1} & \cdots & x_{MN} \end{pmatrix}$$

# Matrix Multiply

In scalar form this would be:

$$x_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{iP}b_{Pj} \quad (2)$$

$$= \sum_{k=1}^P a_{ik}b_{kj} \quad (3)$$

# Matrix Multiply Code

```
clear all

M = 2;
P = 3;
N = 4;

A = rand(M, P);
B = rand(P, N);
X = rand(M, N);

% loop equivalent to X = A*B;

for j = 1:N
    for i = 1:M
        s = 0;
        for k = 1:P
            s = s + A(i,k)*B(k,j);
        end
        X(i,j) = s;
    end
end
```

# Matrix Display Code

Nested loops to multiply matrices:

```
for i = 1:M
    for j = 1:N
        fprintf(1, '%.2f \t', X(i,j));
    end
    fprintf(1, '\n');
end
```

Equivalent to

```
disp(X)
```

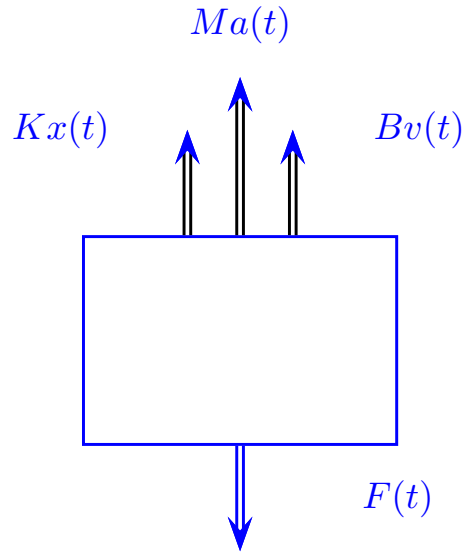
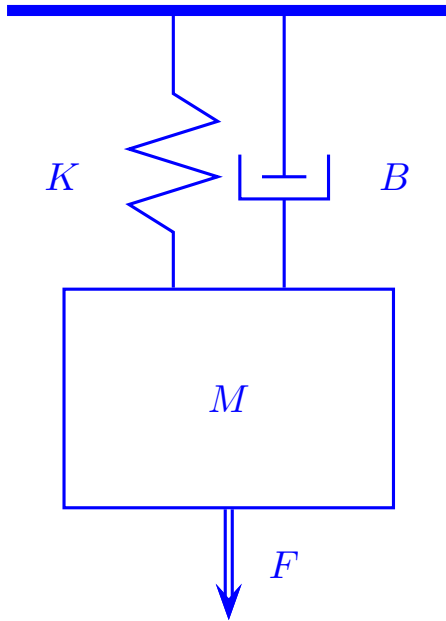
or just

```
X
```

# State Variable Model

- Now let's look at a simple state-variable model.
- The model will be the mass-spring-damper system we considered (and solved) previously.
- *You should review this now...*

# Mass-Spring-Damper



# Mass-Spring-Damper

The equations of motion were...

$$F = M \frac{dv}{dt} + Bv + Kx \quad (4)$$

$$v = \frac{dx}{dt} \quad (5)$$

$\therefore$

$$dv = (F dt - B v dt - Kx dt)/M \quad (6)$$

$$dx = v dt \quad (7)$$

*Recall how we solved this previously...*

# Mass-Spring-Damper

```
clear all
clf

% initial conditions
F = 0;    % no external force
v = 0;    % initial velocity
x = 1;    % initial position

% system parameters - try different values & observe result
K = .1;    % spring constant
M = .1;    % mass
B = .025;  % damping coefficient

dt = 0.01;
MaxTime = 20;

k = 0;
plott = [];
plotx = [];
plotv = [];
```

# Mass-Spring-Damper

And the simulation loop was...

```
for t = 0:dt:MaxTime
```

```
    % instantaneous increments  
    dv = (F/M)*dt - (B/M)*v*dt - (K/M)*x*dt;  
    dx = v*dt;
```

```
    % update instantaneous values  
    v = v + dv;  
    x = x + dx;
```

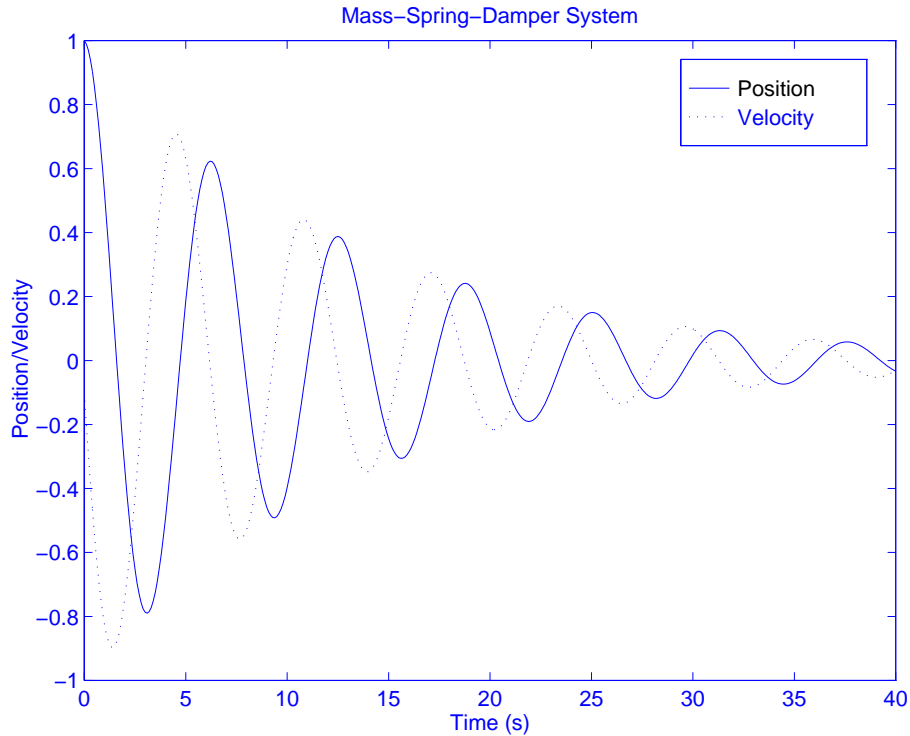
```
    % save instantaneous values for plotting  
    plott = [plott t];  
    plotx = [plotx x];  
    plotv = [plotv v];
```

```
end
```

```
plot(plott, plotx, '-', plott, plotv, ':');  
legend('position', 'velocity');
```

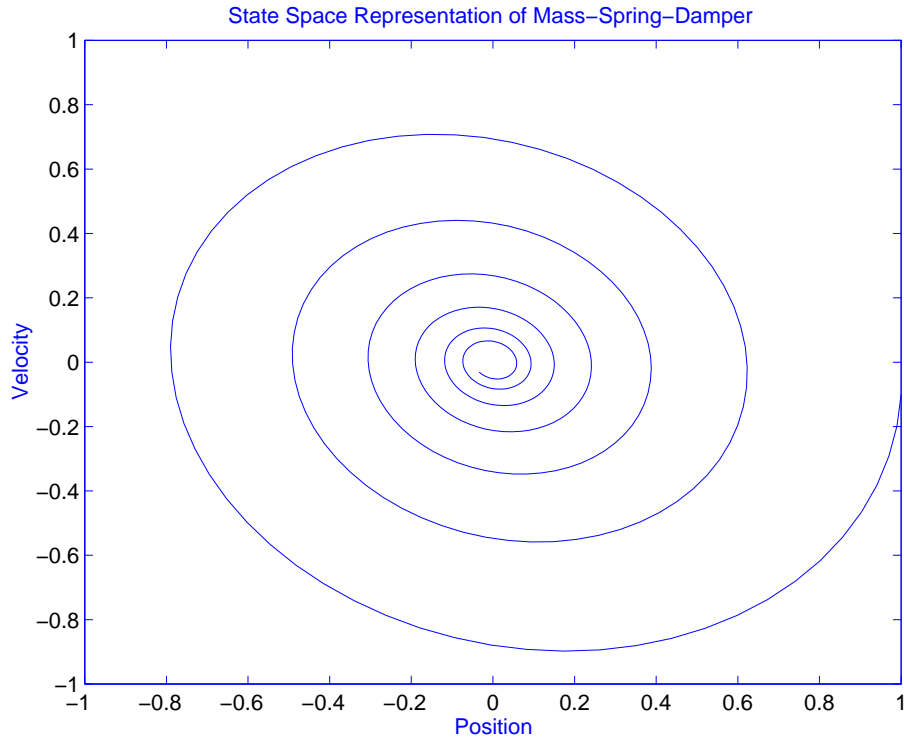
# Mass-Spring-Damper

We could represent this as a function of time:



# Mass-Spring-Damper

Or as state variables:



# State Space

- Time is represented as position along the curve.
- The system is progressing to an equilibrium point on the state plane.
- In this case, the equilibrium operating point is in fact the origin (because there was no external input or driving function).

# State Space Representation

To get in matrix form, get the derivatives on the left-hand side:

$$M\dot{v} = -Bv - Kx + f \quad (8)$$

$$\dot{x} = v \quad (9)$$

Thus

$$\dot{v} = -\frac{B}{M}v - \frac{K}{M}x + \frac{1}{M}f \quad (10)$$

$$\dot{x} = 1v + 0x + 0f \quad (11)$$

# State Space Representation

We can now write in matrix form as:

$$\begin{pmatrix} \dot{v} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} -\frac{B}{M} & -\frac{K}{M} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} v \\ x \end{pmatrix} + \begin{pmatrix} \frac{1}{M} \\ 0 \end{pmatrix} f \quad (12)$$

Let the state variables be  $x_1 = v$  and  $x_2 = x$ , and write in Laplace form:

$$s \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} -\frac{B}{M} & -\frac{K}{M} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} + \begin{pmatrix} \frac{1}{M} \\ 0 \end{pmatrix} F \quad (13)$$

# State Space Representation

The general form is:

$$s\mathbf{X} = \mathbf{A}\mathbf{X} + \mathbf{B}u \quad (14)$$

where

$s$  = Laplace operator

$\mathbf{X}(t)$  = State vector (defines the present state)

$\mathbf{A}$  = “plant matrix” (constant)

$\mathbf{B}$  = “driving matrix” (constant)

$u$  = input variable

# State Space Representation

If we were interested in the velocity  $v$  as the output, then:

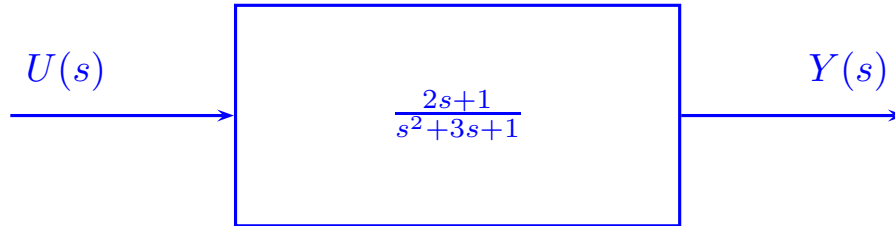
$$y = (1 \ 0) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (15)$$

Or in general:

$$y = \mathbf{C}\mathbf{X} \quad (16)$$

Where  $y$  = output variable  
 $\mathbf{C}$  = “connection matrix” (constant)

# Transfer Function to State Variables



Given a transfer function, how do we make a state variable representation?

$$\frac{Y(s)}{U(s)} = \frac{2s + 1}{s^2 + 3s + 1} \quad (17)$$

# Transfer Function to State Variables

$$Y(s) = \frac{2s + 1}{s^2 + 3s + 1} \cdot U(s) \quad (18)$$

$$= \frac{2s + 1}{s^2 + 3s + 1} \cdot \frac{s^{-2}}{s^{-2}} \cdot U(s) \quad (19)$$

$$= \frac{2s^{-1} + 1s^{-2}}{1 + 3s^{-1} + s^{-2}} \cdot U(s) \quad (20)$$

# Transfer Function to State Variables

hence

$$Y(s) = (2s^{-1} + 1s^{-2}) \cdot E(s) \quad (21)$$

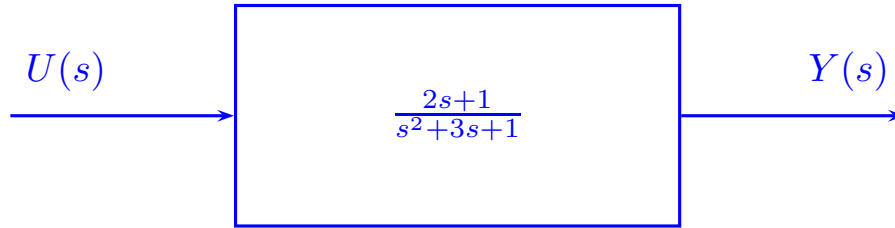
where

$$E(s) = \frac{1}{1 + 3s^{-1} + s^{-2}} \cdot U(s) \quad (22)$$

therefore

$$E(s) = U(s) - 3s^{-1}E(s) - s^{-2}E(s) \quad (23)$$

# Transfer Function to State Variables



# Transfer Function to State Variables

- Begin with  $E(s)$ , and add the two integrators
- Form the *feedforward* path from the integrator outputs to obtain  $Y(s)$
- Form the *feedback* path from the integrator outputs to obtain  $E(s)$
- Call the integrator outputs the *state variables*

# Transfer Function to State Variables

Let  $X_1 = \frac{1}{s}E$  and  $X_2 = \frac{1}{s^2}E$   
(ie state variables from left to right)

$$\frac{X_1}{s} = X_2 \quad (24)$$

$$\therefore X_1 = sX_2 \quad (25)$$

$$\frac{E}{s} = X_1 \quad (26)$$

$$\therefore E = sX_1 \quad (27)$$

# Transfer Function to State Variables

Substitute for  $E(s)$ ,

$$sX_1 = U - 3X_1 - 1X_2 \quad (28)$$

and the output variable  $y$  is

$$Y = 2X_1 + 1X_2 \quad (29)$$

# Transfer Function to State Variables

The system of equations is then

$$sX_1 = -3X_1 - 1X_2 + 1U \quad (30)$$

$$sX_2 = 1X_1 + 0X_2 + 0U \quad (31)$$

hence the matrix is

$$s \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} -3 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} U \quad (32)$$

which is of the form

$$s\mathbf{X} = \mathbf{A}\mathbf{X} + \mathbf{B}u \quad (33)$$

# Transfer Function to State Variables

For the output

$$Y = 2X_1 + 1X_2 \quad (34)$$

hence the matrix is

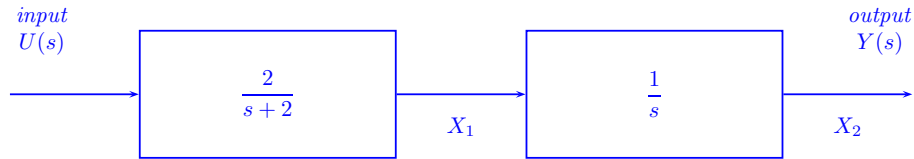
$$Y = (2 \ 1) \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \quad (35)$$

which is of the form

$$y = \mathbf{CX} \quad (36)$$

*Note that the system of equations is not unique — we could also use a partial-fraction method (see Study Guide)*

## Example 2



$$\frac{X_1}{U} = \frac{2}{s+2} \quad (37)$$

$$\frac{X_2}{X_1} = \frac{1}{s} \quad (38)$$

$$Y = 0X_1 + 1X_2 \quad (39)$$

## Example 2

$$sX_1 = 2U - 2X_1 \quad (40)$$

$$sX_2 = X_1 \quad (41)$$

$$s \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} -2 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \end{pmatrix} U \quad (42)$$

$$s\mathbf{X} = \mathbf{A}\mathbf{X} + \mathbf{B}u \quad (43)$$

## Example 2

$$Y = (0 \ 1) \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \quad (44)$$

$$Y = \mathbf{C}\mathbf{X} \quad (45)$$

## Example 3

Suppose we are given a cascade system, comprised of two first-order lags. The first has a gain 1, with a time constant of 2. The second has a gain of 0.2 and a time constant of 5 seconds. Simulate the continuous-time system. First, derive the matrix equations to get **A**, **B** and **C**.

# Example 3

Initialize:

```
% svexamp.m
% two lags: gain 1, time const 2, gain 0.2, time const 5

clear all

A = [-0.5 0 ; 0.04 -0.2]
B = [0.5 ; 0]
C = [0 1]
TMax = 10;

x = [0 ; 0];
u = 1;
dt = .01;
n = 0;
```

## Example 3

The simulation loop is:

```
for t = 0:dt:TMax
    n = n+1;

    dx = (A*x + B*u)*dt;
    x = x + dx;

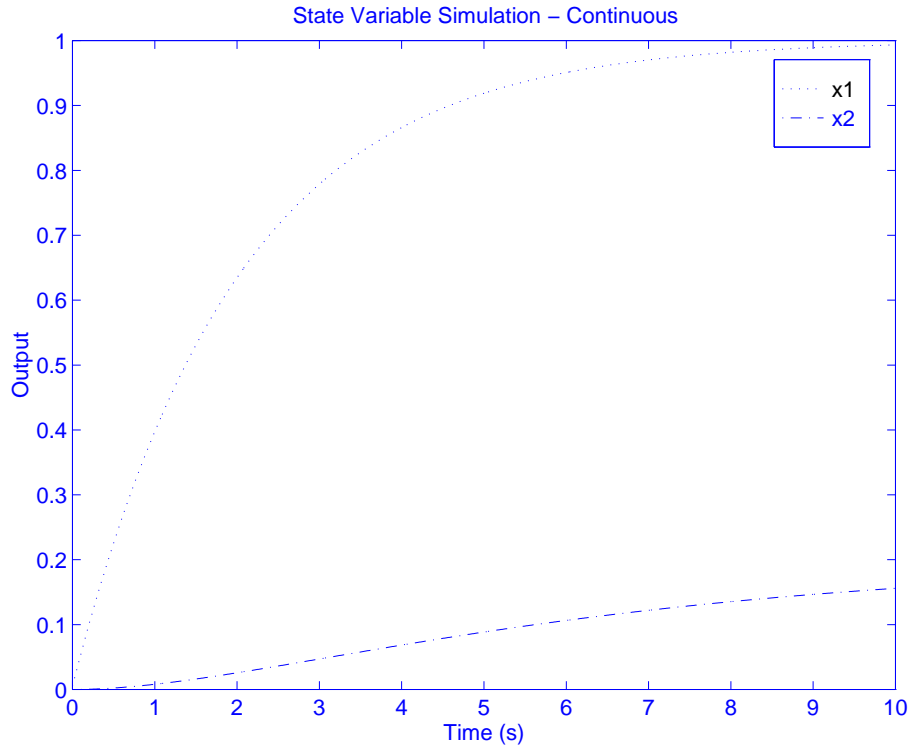
    SaveX(:, n) = x;
    SaveT(n) = t;
end
```

## Example 3

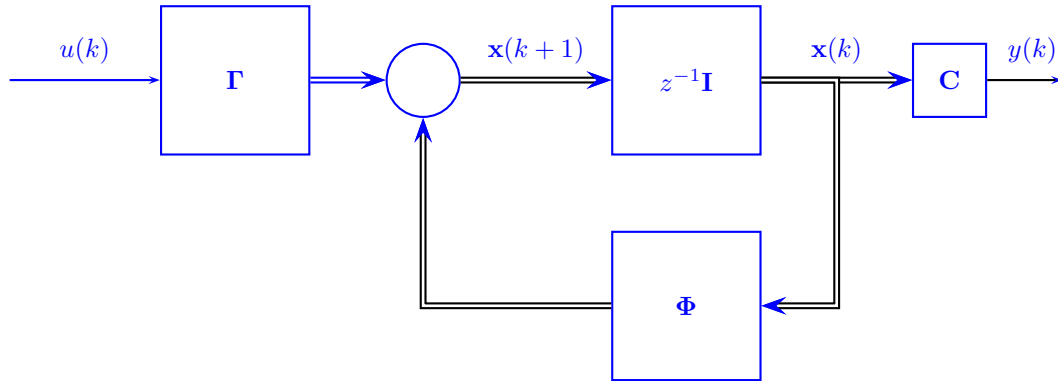
Finally, plot the result:

```
figure(1); clf
plot(SaveT, SaveX(1,:), ':', SaveT, SaveX(2,:), '-.');
legend('x1', 'x2');
```

# Example 3 - Continuous Simulation



# Continuous to Discrete



# State Variable Discrete Model

$$\mathbf{x}(k+1) = \mathbf{\Phi} \mathbf{x}(k) + \mathbf{\Gamma} u(k) \quad (46)$$

$$\mathbf{y}(k) = \mathbf{C} \mathbf{x}(k) \quad (47)$$

With

$$\mathbf{\Phi} = e^{\mathbf{A}T} \quad (48)$$

$$\mathbf{\Gamma} = \int_0^T e^{\mathbf{A}q} \mathbf{B} dq \quad (49)$$

# State Variable Discrete Model

$\Phi$  = discrete plant matrix

$\Gamma$  = discrete driving matrix

The model shows the mapping from the state vector  $\mathbf{x}(k)$  at sample  $k$  to the next state vector  $\mathbf{x}(k + 1)$  at time instant  $k + 1$ .

# State Variable Discrete Model

How to calculate  $\Gamma$  and  $\Phi$  ?

We need an interpretation for  $e^M$ , or the exponential of a *matrix*. For a scalar,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \quad (50)$$

$$= \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad (51)$$

# State Variable Discrete Model

So, we use the same definition for a matrix  $\mathbf{X}$

$$e^{\mathbf{M}} = 1 + \mathbf{M} + \frac{\mathbf{M}^2}{2!} + \frac{\mathbf{M}^3}{3!} + \dots \quad (52)$$

$$= \sum_{n=0}^{\infty} \frac{\mathbf{M}^n}{n!} \quad (53)$$

*Note well: this is different to the matrix exponential defined in MATLAB. The matrix exponential in MATLAB is the exponent of each individual component of the matrix, which is different to that above.*

So, we could write a MATLAB m-file (function) as follows...

# Discrete Model

```
% mexp.m - matrix exponential
function [res] = mexp(x, NumTerms)

if( nargin == 1 )
    NumTerms = 100;
end

N = size(x,1);
if( N ~= size(x,2) )
    error('mexp: matrix must be square');
end

res = eye(N);
num = eye(N);
den = 1;
for n = 1 : NumTerms
    num = num * x;
    den = den * n ;
    res = res + num / den;
end
```

# Discrete Model

- Note that MATLAB performs the matrix multiply at each stage
- In C, we would have to replace each of the matrix multiplies

`num = num * x`  
with a call to a routine

`MatMul(Result, MatA, MatB)`

# Discrete Model

So calculating  $\Phi$  could be done in a MATLAB script as

```
% calphi - calculate matrix Phi  
% function [ResMat] = calphi(A, T)
```

```
function [ResMat] = calphi(A, T)
```

```
[ResMat] = mexp(A*T);
```

# Discrete Model

$\Gamma$  could be done using rectangular integration:

```
% calgamma - calculate matrix Gamma
% function [ResMat] = calgamma(A, B, T)

function [ResMat] = calgamma(A, B, T)

% step for integration
NSteps = 100;

% find  $\int_0^T e^{Aq} dq$  using NSteps
ResMat = zeros(size(A));
for q = 0:T/NSteps:T
    % rectangular integration
    ResMat = ResMat + mexp( A*q ) * (T/NSteps) ;
end

ResMat = ResMat * B;
```

# Discrete Model

So to simulate a discrete state-variable system:

```
% discrete
T = 0.1
Phi = calphi(A, T)
Gamma = calgamma(A, B, T)

NStep = floor(TMax/T);

x = [0 ; 0];
u = 1;
for n = 1:NStep
    x = Phi*x + Gamma*u;
    y = C*x;

    DSaveX(:,n) = x;
    DSaveY(n) = y;
end
```

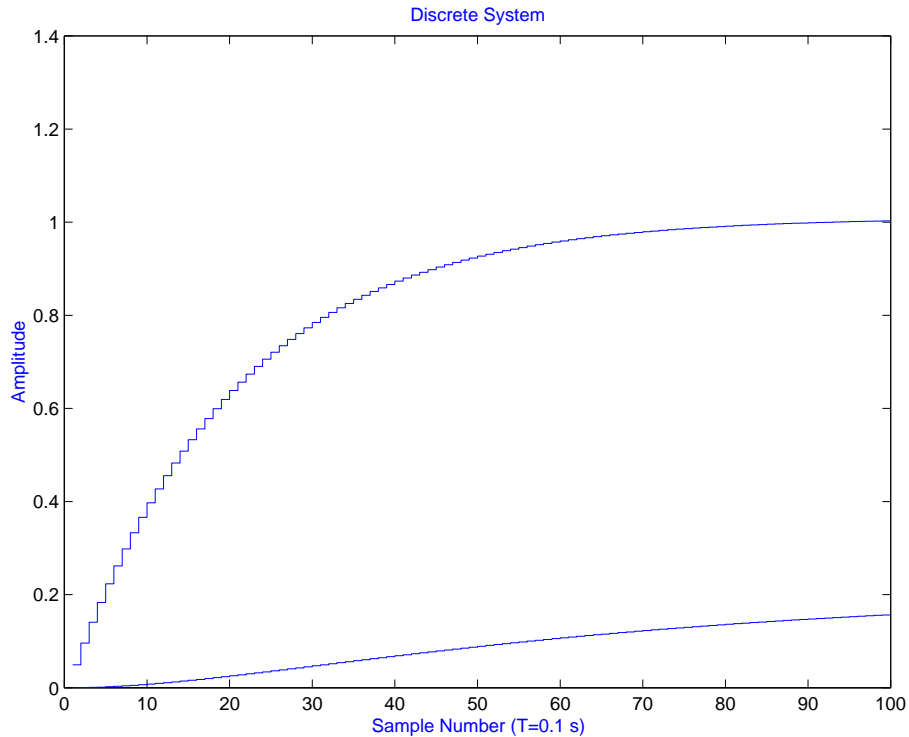
# Discrete Model

And to plot:

```
figure(2); clf
stairs(DSaveX(1,:));
hold on
stairs(DSaveX(2,:));
```

*Note the “stairs” command to show the zero-order hold effect.*

# Example 3 - Discrete Simulation



# Discrete Model with Feedback

Add feedback coefficient vector  $\mathbf{L}$ :

```
% discrete closed-loop
x = [0 ; 0];
r = 1;
L = [1 1];

for n = 1:NStep
    e = r - L*x;
    x = Phi*x + Gamma*e;
    y = C*x;

    DCSaveX(:,n) = x;
    DCSaveY(n) = y;
end
```

# State Feedback

Next question: how to calculate  $\mathbf{L}$  for

- The fastest response.
- Zero steady-state error.
- Plus any other criteria.