CHEM-E7140
2019-2020

FC

Aalto University

Ordinary differential equations

Process Automation (CHEM-E7140)
2019-2020

Francesco Corona

Chemical Engineering and Materials
School of Chemical Engineering

Eigendecomposition and diagonalisation

Ordinary differential equation

Eigendecompostion and diagonalisation

Eigendecompostion and diagonalisation

Linearisation of nonlinear ODEs

Ordinary differential equation

Linearisation of nonlinear ODEs

ODEs with external forcing

Ordinary differential equation

ODEs with external forcing

$$\dot{x}(t) = Ax(t)$$

External forcing, new system (first step to do control)

$$\dot{x}(t) = Ax(t) + \underbrace{b(t)}_{\text{external forcing}} \tag{1}$$

ODEs with external forcing (cont.)

Higher-order linear ODEs in input output form, first

$$\ddot{x}(t) + 3\dot{x}(t) + 2x = \underbrace{0}_{\text{Add external forcing}} \tag{2}$$

Then, we re-write as a system of first-order linear ODEs

ODEs with external forcing (cont.)

$$
\begin{cases}
\ddot{x}(t) + 3\dot{x}(t) + 2x = 0 & \text{(Homogeneous)} \\
\ddot{x}(t) + 3\dot{x}(t) + 2x = \underbrace{e^{-3t}}_{\text{Forcing term}} & \text{(Inhomogeneous)}
\end{cases} \tag{3}
$$

The forcing term is a function of time

How to solve the equation in this case?

- We can guess a solution?

ODEs with external forcing (cont.)

We first solve the homogeneous equation, to find the homogeneous solution

- Characteristic equation

$$
\begin{aligned}
x_o(t) &= e^{\lambda t} \\
\dot{x}_o(t) &= \lambda e^{\lambda t} \\
\ddot{x}_o(t) &= \lambda^2 e^{\lambda t}
\end{aligned}
$$

$$
\begin{aligned}
\lambda_1 &= -1 \\
\lambda_2 &= -2
\end{aligned}
$$

$$[\lambda^2 + 3\lambda + 2] e^{\lambda t} = 0$$

The homogeneous solution

$$
\begin{aligned}
x_o(t) &= k_1 e^{\lambda_1 t} + k_2 e^{\lambda_2 t} && (4a) \\
&= k_1 e^{-t} + k_2 e^{-2t} && (4b)
\end{aligned}
$$

The system is stable, we need to set the constants by using initial conditions

ODEs with external forcing (cont.)

Now we guess how we can make the homogeneous equation equal the forcing function

- Find the particular solution

We can start by guessing a solution, though this works only for simple forcing functions[1]

$$x_{\mathrm{p}}(t) = ke^{-3t} \tag{5}$$

- We substitute and solve for k

$$\dot{x}_{\mathrm{p}} = -3ke^{-3t}$$
$$\ddot{x}_{\mathrm{p}} = 9ke^{-3t}$$

$$k = 1/2$$

$$[9k - 9k + 2k]e^{-3t} = e^{-3t}$$

The particular solution

$$x_p(t) = ke^{-3t} \tag{6a}$$
$$= 1/2e^{-3t} \tag{6b}$$

[1] those that look like solutions of ODEs themselves.

ODEs with external forcing (cont.)

We get, the full solution $x(t)$ by adding the homogeneous and the particular solution

$$
\begin{aligned}
x(t) &= x_{\mathrm{o}}(t) + x_{\mathrm{p}}(t) && \text{(7a)} \\
&= k_1 e^{-t} + k_2 e^{-2t} + 1/2 e^{-3t} && \text{(7b)}
\end{aligned}
$$

We still need to set constants k_1 and k_2

ODEs with external forcing (cont.)

ODEs with external forcing (cont.)

Numerical simulation/integration

Ordinary differential equation

Numerical simulation/integration

Consider an ODE of the type $\dot{x} = f(x)$, the object $f(x)$ is also known as vector field

- A object that gives us a vector $\dot{x}$ for each point x

We can think of an ordinary differential equation as a trajectory in a vector field

We are interested in obtaining a numerical solution for general nonlinear systems

- We set an initial condition for the system x_0 and approximate its trajectory
- A trajectory is understood a sequence of discrete system positions
- It is constructed from a finite difference approximation of $\dot{x}$
- From an iteration scheme, $\{x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots\}$

Numerical simulation/integration (cont.)

Let $\dot{x}$ be approximated using a forward finite differences

$$\dot{x}(t) = \frac{\mathrm{d}x}{\mathrm{d}t} \approx \frac{x(t + \Delta t - x(t))}{\Delta t} = f(x(t))$$

Equivalently, at some discrete point k in time

$$\dot{x}_k \approx \frac{x_{k+1} - x_k}{\Delta t} = f(x_k) \qquad (\forall k)$$

By multiplying both sides by Δt, we get

$$x_{k+1} = x_k + f(x_k)\Delta t$$

These iterative scheme is known as **forward Euler method**

- We are given the position of the system at time k, x_k
- We can evaluate its position at time $k+1$, x_{k+1}
- (The solution is explicit, given x_k)

Numerical simulation/integration (cont.)

Example

Forward Euler for linear systems

Consider $\dot{x} = Ax$

We get,

$$\begin{aligned} x_{k+1} &= x_k + Ax_k\Delta t \\ &= (I - A\Delta t)x_k \end{aligned}$$

■

Numerical simulation/integration (cont.)

Let $\dot{x}$ be approximated using a backward finite differences

$$\dot{x}(t+\Delta t) = \frac{\mathrm{d}x}{\mathrm{d}t} \approx \frac{x(t+\Delta t - x(t))}{\Delta t} = f(x(t+\Delta))$$

Equivalently, at some discrete point k in time

$$\dot{x}_{k+1} \approx \frac{x_{k+1} - x_k}{\Delta t} = f(x_{k+1}) \qquad (\forall k)$$

By multiplying both sides by Δt, we get

$$x_{k+1} = x_k + f(x_{k+1})\Delta t$$

These iterative scheme is known as **backward Euler method**

- We are given the position of the system at time k, x_k
- We cannot evaluate its position at time $k+1$, x_{k+1}
- As we cannot compute $f(x_{k+1})$, x_{k+1} is unknown
- (The solution is implicit)

Numerical simulation/integration (cont.)

Example

Backward Euler for linear systems

Consider $\dot{x} = Ax$

We get,

$$\begin{aligned} x_{k+1} &= x_k + Ax_k\Delta t \\ &= (I - A\Delta t)x_{k+1} \end{aligned}$$

By algebraic manipulations, we obtain

$$x_{k+1} = (I - A\Delta t)^{-1}x_k$$

Note that matrix $(I - A\Delta t)$ must be invertible

Numerical simulation/integration (cont.)

$$x_{k+1} = Mx_k$$

$$\underbrace{x_0}_{x_0} \rightarrow \underbrace{x_1}_{Mx_0} \rightarrow \underbrace{x_2}_{M^2x_0} \rightarrow \cdots \rightarrow \underbrace{x_n}_{M^nx_0} \rightarrow$$

Numerical simulation/integration (cont.)

Example

Numerical simulation/integration (cont.)

Forward Euler

```
w = 2*pi;
d = 1.75;

A = [0 1; -w^2 -2*d*w];
ns = size(A,1)

T = 10;
dt = 0.1;

t0 = 0;          t(1) = t_0
x0 = [2; 0];     x(:,1) = x_0

for k = 1:round(T/dt)
    t(k+1) = k*dt;
    x(:,k+1) = (eye(ns) + A*dt)*x(:,k)
end

subplot(2,1,1); plot(t,x(1,:))
subplot(2,1,1); plot(t,x(2,:))
```

Numerical simulation/integration (cont.)

Backward Euler

```
w = 2*pi;
d = 1.75;

A = [0 1; -w^2 -2*d*w];
ns = size(A,1)

T = 10;
dt = 0.1;

t0 = 0;          t(1) = t_0
x0 = [2; 0];     x(:,1) = x_0

for k = 1:round(T/dt)
    t(k+1) = k*dt;
    x(:,k+1) = inv(eye(ns) + A*dt)*x(:,k)
end

subplot(2,1,1); plot(t,x(1,:))
subplot(2,1,1); plot(t,x(2,:))
```

Numerical simulation/integration (cont.)

ODE45

```
w = 2*pi;
d = 1.75;

A = [0 1; -w^2 -2*d*w];
ns = size(A,1)

T = 10;
dt = 0.1;

t0 = 0;          t(1) = t_0
x0 = [2; 0];     x(:,1) = x_0

[t,x] = ode45(@(t,x), A*x, 0:dt:T, x0)

subplot(2,1,1); plot(t,x(1,:))
subplot(2,1,1); plot(t,x(2,:))
```

Numerical simulation/integration (cont.)

Stability of $\dot{x} = Ax$

$$\mathrm{Re}(\lambda_A) < 0, \text{ for all } \lambda_A$$

Numerical simulation/integration (cont.)

Stability of $x_{k+1} = Mx_k$

$$M = TDT^{-1}$$

$$|\lambda_M| < 1, \text{ for all } \lambda_M$$

Numerical simulation/integration (cont.)

Numerical simulation/integration (cont.)

Numerical simulation/integration (cont.)

Numerical simulation/integration (cont.)

Numerical simulation/integration (cont.)

Numerical simulation/integration (cont.)

Numerical simulation/integration (cont.)