Control Systems

State-Space Models

University of BRISTOL

Contents

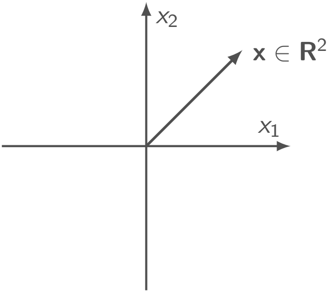
Maths Recap

Euclidean Space

The notation $\mathbf{x} \in \mathbf{R}^n$ means that $\mathbf{x}$ is a **vector** in n-dimensional Euclidean space:

$$\mathbf{x} \in \mathbf{R}^n \quad \Longleftrightarrow \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = (x_1, \ldots, x_n), \quad x_i \in \mathbf{R}$$

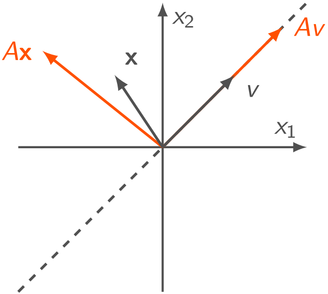
For example, $\mathbf{R}^2$ is a 2-dimensional **plane**:



Eigenvalues

- Recall that an **eigenvector**, v, is defined as a vector (space) in which
$$Av = \lambda v$$
for $\lambda = a + ib$.

- This can be visualised as a vector that points the same way after multiplication by A:



Determining Eigenvalues

- From the definition of an Eigenvalue

$$Av = \lambda v \quad \Longrightarrow \quad (A - \lambda I)v = 0$$

- If $(A - \lambda I)$ is invertible, then the only possible vector is $v = (A - \lambda I)^{-1}0 = 0$
  - This is trivial and not interesting.

- We can only have interesting, nonzero eigenvectors if $(A - \lambda I)$ is **singular**
  - This is again equivalent to the condition $\det(A - \lambda I) = 0$

- This implies that **the eigenvalues of A are the values of λ that satisfy**:

$$\det(A - \lambda I) = 0$$

- This equates to finding the roots of a polynomial in λ.

Eigenvectors

- A matrix A is **diagonalizable** if it has n **unique** eigenvalues
  - This implies that it also has n **independent** eigenvectors.

- This means we can form a matrix, V, out of these eigenvectors, that is **invertible**:

$$V = \begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix} \qquad V^{-1} \text{ exists}$$

Diagonalization

- Let's now look at what happens if we multiply V by A:

$$AV = \begin{bmatrix} | & & | \\ Av_1 & \cdots & Av_n \\ | & & | \end{bmatrix} = \begin{bmatrix} | & & | \\ \lambda_1 v_1 & \cdots & \lambda_n v_n \\ | & & | \end{bmatrix}$$

$$= \begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} = V\Lambda$$

- We know that V is invertible, so

$$A = V\Lambda V^{-1} \quad \Longleftrightarrow \quad V^{-1}AV = \Lambda$$

- A is **similar** to a matrix Λ with the eigenvalues of A on the diagonal.

Matlab

- Using Matlab, the diagonal decomposition can be calculated using `eig`.

- Applied to the matrix

$$A = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}$$

```
A = [0, 1; -1, -1];
[V, Lambda] = eig(A);
disp(V * Lambda * inv(V)) % this will print A
```

- This will calculate the diagonal decomposition

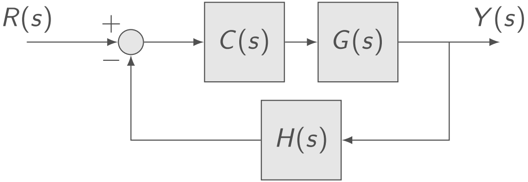
$$V = \begin{bmatrix} 0.7071 & 0.7071 \\ -0.3536 + 0.6124i & -0.3536 - 0.6124i \end{bmatrix}$$

$$\text{and} \quad \Lambda = \begin{bmatrix} -0.5 + 0.866i & 0 \\ 0 & -0.5 - 0.866i \end{bmatrix}$$

Motivation

Motivation

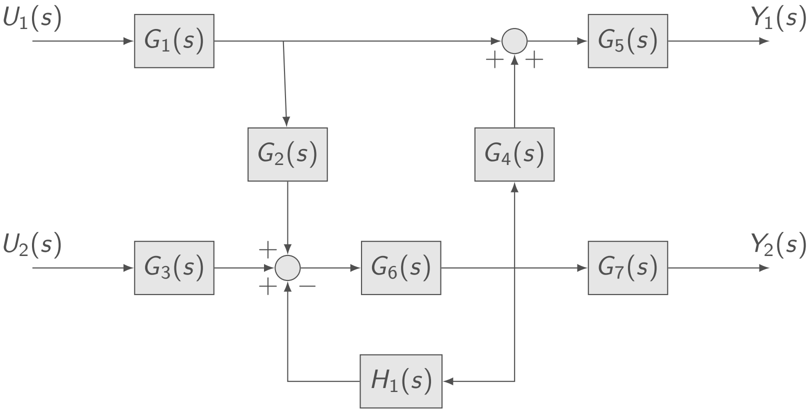
- Simple techniques (e.g. PID) are effective for controlling single output (SISO) transfer function models:



What happens when we need to control multiple inputs and outputs?

Multiple Input and Outputs

- Should be able to condense more complex models down to $N_U \times N_Y$ transfer functions:



Total Solution

- For this form of system we can characterize the total output as

$$Y_1(s) = G_{11}(s)U_1(s) + G_{21}(s)U_2(s)$$
$$Y_2(s) = G_{21}(s)U_1(s) + G_{22}(s)U_2(s)$$

- This has the equivalent representation

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix}$$

  or, more concisely

$$\mathbf{Y}(s) = \mathbf{G}(s)\mathbf{U}(s)$$

Controller Design

- Suppose we want to control this system so that

$$y_1(t) \to r_1(t) \quad \text{and} \quad y_2(t) \to r_2(t)$$

as $t \to \infty$, using negative feedback:

$$U_1(s) = C_{11}(s)[R_1(s) - Y_1(s)] + C_{12}(s)[R_2(s) - Y_2(s)]$$
$$U_2(s) = C_{21}(s)[R_1(s) - Y_1(s)] + C_{22}(s)[R_2(s) - Y_2(s)]$$

- This is, in turn, equivalent to

$$\begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix} = \begin{bmatrix} C_{11}(s) & C_{12}(s) \\ C_{21}(s) & C_{22}(s) \end{bmatrix} \begin{bmatrix} R_1(s) - Y_1(s) \\ R_2(s) - Y_2(s) \end{bmatrix}$$

or, more concisely

$$\mathbf{U}(s) = \underbrace{\mathbf{C}(s)}_{\text{To be designed.}} [\mathbf{R}(s) - \mathbf{Y}(s)]$$

Design Issues

- Why can't we apply our current techniques to design $\mathbf{C}(s)$?
- The dynamics are **coupled**:

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix}$$

$\Longrightarrow$ Have to design all controllers **simultaneously**.

- Even if they weren't, the approach doesn't scale
  - ► Need to design $N_U \times N_Y$ controllers individually.
- Solution: **state-space methods**
  - ► Allow us to leverage the power of **linear algebra** for controller design.
  - ► Foundation of **modern** control theory (since 1950/1960's).

Learning Objectives

- The main focus of this week is on **models** of system behaviour
  1. What is the state-space approach?
  2. How can we model linear time-invariant systems with state-space models?
  3. How can we linearize nonlinear state-space models?
  4. Are state-space representations unique?
  5. ...

- This material will be built on when **analysing** state-space models and **designing** controllers for state-space systems.

Introduction to State-Space

General State-Space Form

- The most general form of a state-space model is

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)\\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t)\end{aligned}$$

  where
  - $\mathbf{x} \in \mathbf{R}^n$ are the **states**
  - $\mathbf{u} \in \mathbf{R}^m$ are the **inputs**
  - $\mathbf{f}()$ describes the **system dynamics**
  - $\mathbf{g}()$ describes the **sensors**
  - $\mathbf{y} \in \mathbf{R}^o$ are the **measured outputs**

- Objective is to design a feedback controller $\mathbf{u}(t) = K(\mathbf{y}(t))$

- For now, we're going to ignore the sensors and outputs and focus on the **dynamics**

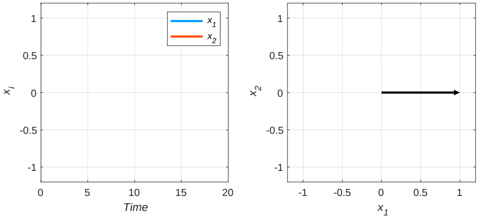
$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)$$

Why is it Called State Space?

- The conceptual idea behind state-space methods is that our system can be represented by a **vector** that moves over time.
- For example, the system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t)) \quad \Longleftrightarrow \quad \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -x_1(t) \end{bmatrix}$$

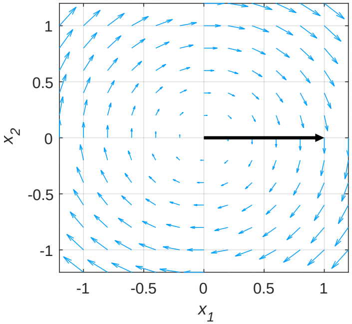
can either be viewed as **two states** that vary with time, or a **single vector** that moves within $\mathbf{R}^2$:



- The vector space that the state moves in **is the state-space**.

The Dynamics

- Similarly, the system's dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t))$ can be viewed as a **vector field** within $\mathbf{R}^n$, where the system's trajectories follow the derivative vectors:



- These **vector-based** ways of thinking about systems will help us to analyze their properties, and **design systems to control them**.

Converting General ODEs to State-Space

- ODEs typically have the form

$$x^{(n)} = f(x^{(n-1)}, \ldots, x(t), \mathbf{u}(t), t)$$

  - (Or they can be rearranged to this form)

- These ODEs can be converted to n **first-order** ODEs
  - This is the standard technique for 'shoehorning' ODEs into `ODE45`...

General Method

1. Create n 'dummy variables' $x_i(t)$ for $i = 1, \ldots, n$, so that $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$
2. Set $\dot{x}_i = x_{i+1}(t)$ for $i = 1, \ldots, n-1$
3. Set $\dot{x}_n = f(x_{n-1}(t), \ldots, x_1(t), \mathbf{u}(t), t)$

Example

- Dynamics of a simplified pendulum are given by

$$\ddot{\theta} = \frac{2g}{l}\sin\theta(t) + \frac{4}{ml^2}\tau(t)$$

- Using the general method:
  1. Create two dummy variables: $\mathbf{x}(t) = (x_1(t), x_2(t))$
  2. Set $\dot{x}_1 = x_2(t)$
  3. Set $\dot{x}_2 = \frac{2g}{l}\sin x_1(t) + \frac{4}{ml^2}\tau(t)$

  we get

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix} x_2(t) \\ \frac{2g}{l}\sin x_1(t) + \frac{4}{ml^2}\tau(t)\end{bmatrix} = \mathbf{f}(\mathbf{x}(t), \tau(t))$$

- Note that $(\theta(t), \dot{\theta}) \iff (x_1(t), x_2(t))$

Coupled Systems

- Often we are interested in systems constituted by multiple smaller, interacting systems, e.g.

$$
\begin{aligned}
x^{(n)} &= f(x^{(n-1)}, \ldots, x(t), y^{(m-1)}, \ldots, y(t)) \\
y^{(m)} &= g(x^{(n-1)}, \ldots, x(t), y^{(m-1)}, \ldots, y(t))
\end{aligned}
$$

- In this case we just repeat the process for the additional derivatives and just stack them together.

Example

- Consider the ODEs

$$\ddot{x} = \sin(y(t) - x(t))$$
$$\ddot{y} = -y(t)^3 - \dot{y}\dot{x}$$

- We now create **four** dummy variables:

$$\mathbf{x}(t) = (x_1(t), x_2(t), x_3(t), x_4(t)) = (x(t), \dot{x}, y(t), \dot{y})$$

  then follow the same process for each of the ODEs

  1. Set $\dot{x}_1 = x_2(t)$
  2. Set $\dot{x}_2 = \sin(x_3(t) - x_1(t))$
  3. Set $\dot{x}_3 = x_4(t)$
  4. Set $\dot{x}_4 = -x_3(t)^3 - x_4(t)x_2(t)$

- Therefore

$$\dot{\mathbf{x}} = \begin{bmatrix} x_2(t) \\ \sin(x_3(t) - x_1(t)) \\ x_4(t) \\ -x_3(t)^3 - x_4(t)x_2(t) \end{bmatrix}$$

Linear Time-Invariant State-Space Models

Linear Time Invariant State-space Models

- Recall that state-space models of system dynamics have the general form
$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)$$
- **Linear** state-space models have the form:
$$\dot{\mathbf{x}} = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t),$$
where A and B are **matrices**[1].
- Of particular interest are linear models that are also **time-invariant**:
$$\dot{\mathbf{x}} = A\mathbf{x}(t) + B\mathbf{u}(t)$$
- This form of model allows us to leverage powerful techniques from linear algebra.

[1] Of conformal size (i.e. $A(t) \in \mathbf{R}^{n \times n}$ and $B(t) \in \mathbf{R}^{n \times m}$)

Linear ODEs with Constant Coefficients

- This entire course has been concerned with systems described by **linear ODEs with constant coefficients**:

$$a_n x^{(n)} + a_{n-1} x^{(n-1)} + \cdots + a_1 x' + a_0 x(t) = b_m u^{(m)} + b_{m-1} u^{(m-1)} + \cdots + b_1 u' + b_0 u(t)$$

- Recall that this is equivalent to the transfer function

$$\frac{X(s)}{U(s)} = \frac{b_m s^m + \cdots + b_0}{a_n s^n + \cdots + a_0}$$

- We will now look at how we can represent these in state space.

Linear State-space Form

- For now, assume that we have an ODE of the form

$$a_n x^{(n)} + a_{n-1} x^{(n-1)} + \cdots + a_1 x' + a_0 x(t) = u(t)$$

- If we follow the standard steps for converting ODEs to state-space we get the equivalent representation

$$\begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & & \\ -\frac{a_0}{a_n} & \cdots & & -\frac{a_{n-2}}{a_n} & -\frac{a_{n-1}}{a_n} \end{bmatrix} \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1/a_n \end{bmatrix} u(t) \tag{1}$$

- This is **the controllable canonical form**.
  - State-space representations of systems are not unique.
  - We will later see that they are all *equivalent*.

Input Derivatives

- Typically we are also interested in ODEs that contain derivatives of the input function:

$$a_n x^{(n)} + a_{n-1}x^{(n-1)} + \cdots + a_1 x' + a_0 x(t) = \\ b_m u^{(m)} + b_{m-1}u^{(m-1)} + \cdots + b_1 u' + b_0 u(t)$$

- We approach this with dummy variable $z(t)$ with Laplace transform $Z(s)$

- Multiply top and bottom of out transfer function by $Z(s)/Z(s)$

$$G(s) = \frac{X(s)}{U(s)} = \frac{(b_m s^m + \cdots + b_0)Z(s)}{(a_n s^n + \cdots + a_0)Z(s)}$$

- Therefore

$$x(t) = b_m z^{(m)} + \cdots + b_0 z(t)$$
$$u(t) = a_n z^{(n)} + \cdots + a_0 z(t)$$

Input Derivatives

- Under the assumption that $n \geq m$, we can therefore model a stste-space system in terms of $z(t)$ using just the ODE:

$$u(t) = a_n z^{(n)} + \cdots + a_0 z(t)$$

- Therefore, using

$$\begin{aligned} x_1(t) &= z(t) \\ x_2(t) &= \dot{z} = \dot{x}_1 \\ &\vdots \\ x_n(t) &= z^{(n)} = \frac{1}{a_n} u(t) - \frac{a_{n-1}}{a_n} x_{n-1}(t) + \ldots \end{aligned}$$

we get the same state-space representation as (1).

- We can then obtain the original output from

$$x(t) = [b_0, \ldots, b_m] \begin{bmatrix} x_1(t) \\ \vdots \\ x_m(t) \end{bmatrix}$$

Example

- We want to generate a state-space representation of the ODE

$$\ddot{x} + 0.2\dot{x} + x(t) = \dot{u} + u(t)$$

- This has the transfer function representation

$$G(s) = \frac{s+1}{s^2 + 0.2s + 1}$$

- We therefore use the 'dummy' ODE

$$u(t) = \ddot{z} + 0.2\dot{z} + z(t)$$

which has the state-space representation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -0.2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

with $\mathbf{x} = (x_1(t), x_2(t)) = (z(t), \dot{z})$

Output

- The second ‘dummy’ ODE is

$$x(t) = \dot{z} + z(t),$$

so we can then obtain the original output with

$$x(t) = \begin{bmatrix} 1 & 1 \end{bmatrix} \mathbf{x}(t)$$

- This representation can then be validated in Matlab with

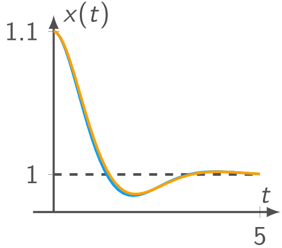
```
sys_tf = tf([1, 1], [1, 0.2, 1]); % Transfer function model
A = [0, 1; -1, -0.2]; B = [0; 1]; C = [1, 1]; D = [];
sys_ss = ss(A, B, C, D); % State space model

figure(1)
step(sys_tf, 'b')
hold on
step(sys_ss, 'r--')
hold off
```

Linearization

Linearization: Recap

- Recall that linearizing a nonlinear model gives us a linear model of its behaviour:



- This approach is useful when we want to control the behaviour of a system **close to a given set-point**.

Linearization: Recap

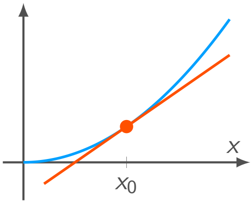
- We previously looked at methods for linearizing models of ODEs of the form

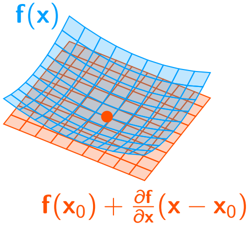
$$x^{(n)} = f(x^{(n-1)}, \ldots, x(t), u^{(m)}, \ldots, u(t))$$

- Sometimes it is easier to develop models of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$$

- The same process can be applied to models of this form using **vector calculus**.





Taylor's Series

- We can make a first order approximation of a **differentiable** function $\mathbf{f}$ at a point $(\mathbf{x}_0, \mathbf{u}_0)$ as

$$\dot{\mathbf{x}} \approx \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) + \frac{\partial \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{x}}(\mathbf{x}(t) - \mathbf{x}_0) + \frac{\partial \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{u}}(\mathbf{u}(t) - \mathbf{u}_0)$$

where $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ are the **Jacobian matrices** of $\mathbf{f}$ w.r.t. $\mathbf{x}$ and $\mathbf{u}$:

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}, \quad \frac{\partial \mathbf{f}}{\partial \mathbf{u}} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_m} \end{bmatrix}.$$

Linearized Model

- If we then
  - Choose $\mathbf{u}_0$ such that $\mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) = 0$
  - Change variables to $\Delta\mathbf{x}(t) = \mathbf{x}(t) - \mathbf{x}_0$ and $\Delta\mathbf{u}(t) = \mathbf{u}(t) - \mathbf{u}_0$

  we get

$$\dot{\Delta\mathbf{x}} \approx \frac{\partial \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{x}} \Delta\mathbf{x}(t) + \frac{\partial \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{u}} \Delta\mathbf{u}(t).$$

- Remember that if we then design a controller $\Delta\mathbf{u}(t) = K(\Delta\mathbf{x}(t))$ for this linearized system, the controller for the **original system** is

$$\mathbf{u}(t) = \mathbf{u}_0 + K(\mathbf{x}(t) - \mathbf{x}_0).$$

Example

- Consider the state-space model

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -\sin(x_1(t) - u(t)) - x_2(t) + x_2(t)^2 \end{bmatrix}$$

  which we would like to control close to the point $\mathbf{x}_0 = (0.5, 0)$.
- Choose $\mathbf{u}_0 = [u_0] = 0.5 \implies \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) = 0$
- The Jacobian matrices are given by

$$\frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -\cos(x_1(t) - u(t)) & -1 + 2x_2(t) \end{bmatrix}$$

$$\frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}} = \begin{bmatrix} 0 \\ \cos(x_1(t) - u(t)) \end{bmatrix}$$

Solution

- Evaluating the Jacobian matrices at $\mathbf{x}_0$ and $\mathbf{u}_0$ gives

$$\frac{\partial \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} \quad \frac{\partial \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{u}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Therefore, we can approximate the nonlinear state-space model with

$$\dot{\Delta \mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} \Delta \mathbf{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Delta \mathbf{u}.$$

Coordinate Transformations

Coordinate Transformations

- We have introduced the canonical form of LTI state-space model

$$\dot{\mathbf{x}} = A\mathbf{x}(t) + B\mathbf{u}(t)$$

- Sometimes it is helpful to consider this system in different coordinate system

$$\mathbf{x}(t) = P\mathbf{z}(t)$$

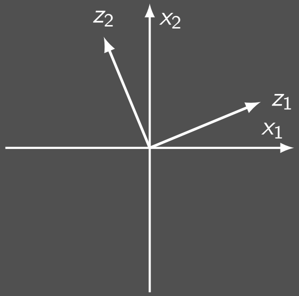
- Always consider square matrix P with full-rank (often, though not always, orthogonal vectors)

$$\implies \mathbf{z}(t) = P^{-1}\mathbf{x}(t)$$

Example: Rotation Matrix

- Given a system with a state $\mathbf{x}(t) \in \mathbf{R}^2$, we can consider this in a coordinate frame rotated anticlockwise by θ about the origin with

$$P = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \implies P\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}\cos\theta\\ \sin\theta\end{bmatrix}, \quad P\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}-\sin\theta\\ \cos\theta\end{bmatrix}$$



Similarity Transformation

- We can differentiate both sides of $\mathbf{x}(t) = P\mathbf{z}(t)$ to get

$$\dot{\mathbf{x}} = P\dot{\mathbf{z}}(t) \quad \Longrightarrow \quad \dot{\mathbf{z}} = P^{-1}\dot{\mathbf{x}}$$

- Therefore,

$$\dot{\mathbf{x}} = A\mathbf{x}(t) + B\mathbf{u}(t) \quad \Longleftrightarrow \quad P\dot{\mathbf{z}} = AP\mathbf{z}(t) + B\mathbf{u}(t)$$

- We can then left-multiply by P^{-1} to get new state-space system:

$$\dot{\mathbf{z}} = P^{-1}AP\mathbf{z}(t) + P^{-1}B\mathbf{u}(t) = \hat{A}\mathbf{z}(t) + \hat{B}\mathbf{u}(t)$$

- The matrices $\hat{A}$ that can be obtained from

$$\hat{A} = P^{-1}AP \quad \Longleftrightarrow \quad A = P\hat{A}P^{-1}$$

  are said to be **similar** to A, and P is therefore commonly known as a **similarity transformation**.

Preserved Properties

- Several important properties are preserved under similarity transformation:
  - Eigenvalues ($\Longrightarrow$ stability)
  - Controllability
  - Observability
  - ...
- These properties can be significantly easier to demonstrate in new coordinate frames
- Therefore similarity transformations are very common in analysis of state-space systems

Eigenvalue Decomposition

- An important transformation can be made when A has n unique eigenvalues

$\implies$ A has n **independent** eigenvectors.
$\implies$ We can form an **invertible** matrix V from the eigenvalues of A:

$$V = \begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix} \qquad V^{-1} \text{ exists}$$

- Under the transformation $\mathbf{x}(t) = V\mathbf{z}(t)$ we get

$$\dot{\mathbf{z}} = V^{-1}AV\mathbf{z}(t) + V^{-1}B\mathbf{u}(t) = \Lambda\mathbf{z}(t) + V^{-1}B\mathbf{u}(t)$$

where

$$\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

- This is the **diagonalized form** of our state-space system.

Importance of Diagonal Form

- To illustrate the importance of the diagonal form, for a moment forget about the input:

$$\dot{\mathbf{x}} = A\mathbf{x}(t) \quad \Longrightarrow \quad \dot{\mathbf{z}} = \Lambda\mathbf{z}(t), \quad \mathbf{x}(t) = V\mathbf{z}(t)$$

- As Λ is diagonal, we have **decoupled** the n ODEs in $\dot{x} = A\mathbf{x}(t)$:

$$\begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix} \dot{z}_1 \\ \vdots \\ \dot{z}_n \end{bmatrix} = \begin{bmatrix} \lambda_1 z_1 \\ \vdots \\ \lambda_n z_n \end{bmatrix}$$

- **Much** easier to deal with than general matrix A.

- Very useful in determining stability of original system.

Discrete Time Models

Motivation

- So far, we have generally considered **continuous time** system dynamics, which provides us:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \quad \rightarrow \quad \mathbf{x}(t) \quad t \in [0, T]$$

- We are often interested in **discrete time** models, where we are only interested in the state variable at particular points in time:

$$\mathbf{x}_{t+\delta_t} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, t) \quad \rightarrow \quad \mathbf{x}_t \quad t \in \{0, \delta_t, 2\delta_2, \ldots, T\}$$

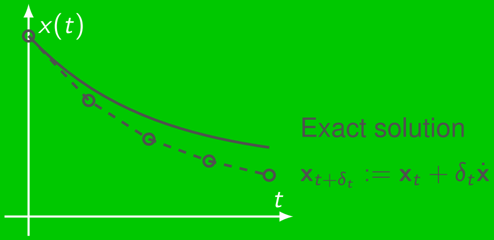
  - Embedded systems generally update state estimates and control after finite intervals
  - Some methods for control synthesis are better suited to discrete time formulations (e.g. MPC).

- We will now explore how to convert our continuous time models to discrete time.

Numerical Integration

- We have previously looked at **numerical methods** for approximating the solution of ODEs.

Euler's method



- Numerical methods can also be used to convert our continuous-time models to discrete time
  - Will generally introduce an **approximation error**.

Euler's Method

- Euler's method provides a simple approximation of the discrete time dynamics

- Using Eulers method:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \quad \rightarrow \quad \mathbf{x}_{t+\delta_t} = \mathbf{x}_t + \delta_t \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, t)$$

- Approximation error is generally large, even for small δ_t.
  - Implicit assumption that $\mathbf{f}()$, $\mathbf{x}(t)$, and $\mathbf{u}(t)$ are constant between t and $t + \delta_t$
  - For $\mathbf{u}(t)$ this is often true in practice (Zero-order hold).
  - For $\mathbf{f}()$ and $\mathbf{x}(t)$ it is **not**

- Higher-order methods can be used (e.g. Runge-Kutta), but rarely result in a simple, closed form solution.

Example

- Consider the state-space model

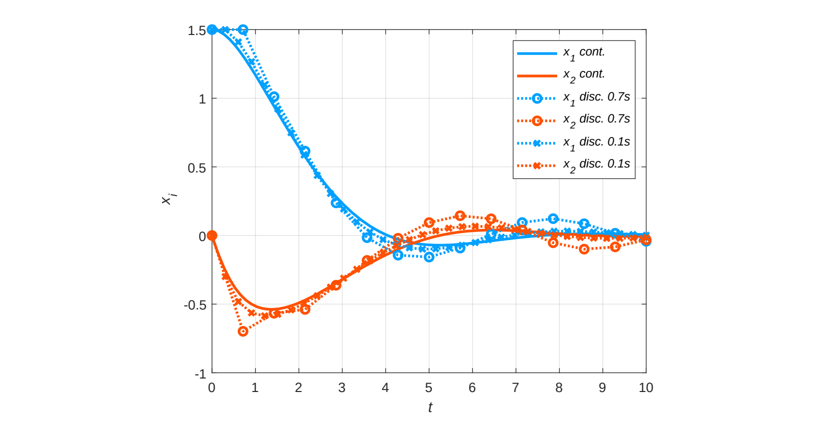
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -\sin(x_1(t) - u(t)) - x_2(t) + x_2(t)^2 \end{bmatrix}$$

  This can be approximated in discrete time by

$$\begin{bmatrix} x_1(t+\delta_t) \\ x_2(t+\delta_t) \end{bmatrix} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \delta_t \begin{bmatrix} x_2(t) \\ -\sin(x_1(t) - u(t)) - x_2(t) + x_2(t)^2 \end{bmatrix}$$

Simulations

- Solution in continuous time from $\mathbf{x}(0) = (1.5, 0)$ (with $u(t) = 0$), and using Euler approximation with $\delta_t = 0.7$ and $\delta_t = 0.1$:



Exact Method

- There are some cases where the discrete-time model can be obtained **exactly**.

- A particular case of this is where the system is **linear and time invariant**
$$\dot{\mathbf{x}} = A\mathbf{x}(t) + B\mathbf{u}(t),$$
A is **invertible**, and the control input is **held constant** between sampling intervals.

- In this case
$$\mathbf{x}_{t+\delta_t} = A_d\mathbf{x}_t + B_d\mathbf{u}_t,$$
where
$$A_d := e^{A\delta t}, \quad B_d := [A_d - I]A^{-1}B$$

Matrix Exponential

- We have now introduced the **matrix exponential**: $e^{\mathbf{X}}$
- It may not be immediately obvious how this relates to the exponential function: e^x
- Recall that the formal definition of the exponential function is

$$e^x := \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots$$

- The matrix exponential is similarly defined as

$$e^{\mathbf{X}} := \sum_{k=0}^{\infty} \frac{\mathbf{X}^k}{k!} = I + \mathbf{X} + \frac{\mathbf{X}^2}{2} + \frac{\mathbf{X}^3}{6} + \ldots$$

Properties of the Matrix Exponential

The matrix exponential has very similar properties to the standard exponential function:

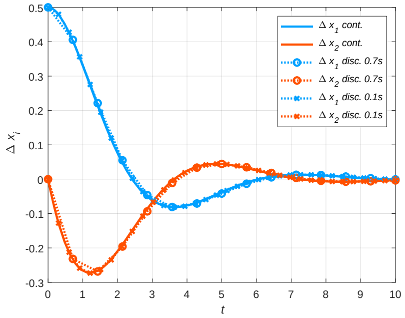
1. $e^{\mathbf{0}} = I$
2. $\frac{\mathrm{d}}{\mathrm{d}t} e^{At} = Ae^{At} = e^{At}A$
3. $\int e^{At} = A^{-1}e^{At} + c = e^{At}A^{-1} + c$ (Assuming A nonsingular)
4. $e^{\mathbf{A}}e^{\mathbf{B}} = e^{\mathbf{A}+\mathbf{B}}$
5. ...

Example

- Simulations for the **linearized** model we obtained previously,

$$\dot{\Delta \mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} \Delta \mathbf{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Delta \mathbf{u},$$

again discretised with $\delta_t = 0.7$ and $\delta_t = 0.1$:



Conclusion

Conclusion

- We have now introduced **state-space** methods for modelling dynamical systems.
  - Starting to think of states as a **vector** that **moves over time**.
  - Investigated methods of generating state-space models.
- We can build on these ideas for controller design
  - Can be applied to **multi input, multi output** models
  - Allow us to leverage techniques from **linear algebra**: a more algorithmic approach.
- To do so we will need to revisit the concept of **stability**, and introduce two new concepts:
  - Controllability
  - Observability