Table of Contents I

notes

- this is the table of contents of this document; each section corresponds to a specific part of the course

state space from ARMA (and viceversa)

notes

Contents map

developed content units	**taxonomy levels**
realization	u1, e1

prerequisite content units	**taxonomy levels**
ARMA model	u1, e1
state space model model	u1, e1
matrix inversion	u1, e1
Zeta transforms	u1, e1

notes

▪

Main ILO of sub-module "state space from ARMA (and viceversa)"

Determine the state space structure of an discrete time LTI system starting from an ARMA RR

notes

- by the end of this module you shall be able to do this

ARMA models

$$y^{[n]} = a_{n-1} y^{[n-1]} + \ldots + a_0 y + b_m u^{[m]} + \ldots + b_0 u$$

notes

- the $a_{n-1} y^{[n-1]} + \ldots + a_0 y$ part is called Auto-Regressive
- the $b_m u^{[m]} + \ldots + b_0 u$ part is called Moving-Average
- these names make very much sense in discrete time systems of the type $y[k+n] = a_{n-1} y[k+n-1] + \ldots + a_0 y[k] + b_m u^[k+m] + \ldots + b_0 u[k]$ and k a discrete time index. Here we see that the a's correspond to an autoregression, and the b's to the coefficients of a moving average. In any case we use ARMA for both continuous and discrete dynamics of these types

State space representations - Notation

$$\begin{aligned} x_1^+ &= f_1 \left(x_1, \ldots, x_n, \; u_1, \ldots, u_m \right) \\ &\vdots \\ x_n^+ &= f_n \left(x_1, \ldots, x_n, \; u_1, \ldots, u_m \right) \\ y_1 &= g_1 \left(x_1, \ldots, x_n, \; u_1, \ldots, u_m \right) \\ &\vdots \\ y_p &= g_p \left(x_1, \ldots, x_n, \; u_1, \ldots, u_m \right) \end{aligned}$$

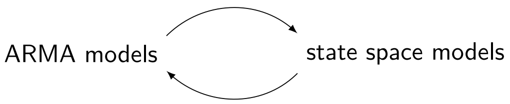
$$\begin{aligned} \boldsymbol{x}^+ &= \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} &= \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) \end{aligned}$$

- $\boldsymbol{f}$ = state transition map
- $\boldsymbol{g}$ = output map

notes

- notation wide, remember that state space means first order RRs
- they will thus look like these ones, in general
- we can also compress the notation in this way
- remember that bold non-capital fonts mean vectors in this course
- and we give to the various things these names

This module:



But why do we study this?

because from physical laws we get ARMA,
but with state space we get more explainable models

notes

- we will learn how to do two simple operations
- we will only scratch the surface though, there is a lot of material to cover here and you will do it much better in other modules / courses
- and often one does the "ARMA to SS" operation

From state space to ARMA

notes

SS to ARMA

Tacit assumption: $\boldsymbol{x}[0] = \boldsymbol{0}$

$$\begin{cases} \boldsymbol{x}^+ & = A\boldsymbol{x} + Bu \\ y & = C\boldsymbol{x} + Du \end{cases} \quad \rightarrow \quad \mathcal{Z}\left(\begin{cases} \boldsymbol{x}^+ & = A\boldsymbol{x} + Bu \\ y & = C\boldsymbol{x} + Du \end{cases}\right)$$

$$\rightarrow \quad \begin{cases} zX & = AX + BU \\ Y & = CX + DU \end{cases}$$

$$\rightarrow \quad \begin{cases} (zI - A)X & = BU \\ Y & = CX + DU \end{cases}$$

$$\rightarrow \quad \begin{cases} X & = (zI - A)^{-1}BU \quad (*) \\ Y & = CX + DU \end{cases}$$

$$\Rightarrow \quad Y = \left(C(zI - A)^{-1}B + D\right)U$$

$$\Rightarrow \quad Y(z) = \frac{\text{polynomial in } z}{\text{polynomial in } z}U(z)$$

notes

- This slide shows the step-by-step derivation of the transfer function from the state-space representation using Zeta transforms.
- the assumption $\boldsymbol{x}[0] = \boldsymbol{0}$ simplifies the Zeta transform of the derivative.
- the key step where $X = (zI - A)^{-1}BU$ is derived is the foundation for the transfer function.
- the final result, $Y(z) = \frac{\text{polynomial in } z}{\text{polynomial in } z}U(z)$, is the ARMA representation of the system.
- For computations, I recommend using tools like `simpy` for symbolic algebra, but you should be able to handle 2x2 systems by hand.

A note on the last formula

$$Y(z) = \frac{\text{polynomial in } z}{\text{polynomial in } z}U(z) \quad \mapsto \quad \text{ARMA:}$$

$$Y(z) = \frac{z + 3}{2z^3 + 3z}U(z) \quad \mapsto \quad 2y^{+++} + 3y^{+} = u^{+} + 3u$$

notes

- This slide connects the transfer function to the ARMA model in the time domain.
- the numerator and denominator polynomials in z directly translate to differential equations in the time domain.
- the example shows how the transfer function $Y(z) = \frac{z + 3}{2z^3 + 3s}U(z)$ corresponds to the differential equation $2y^{+++} + 3y^{+} = \dot{u} + 3u$.
- this is a key step in understanding the relationship between the Zeta domain and time domain.

A note on the second to last formula

$$Y = \left(C(zI - A)^{-1}B + D\right)U$$

DISCLAIMER: in this course we consider SISO systems, thus C and $B =$ vectors, and $D =$ scalar (if present)

notes

- that this course focuses on Single Input Single Output (SISO) systems, which simplifies the matrices C, B, and D.
- C and B are vectors, and D is a scalar (often zero in many systems).
- this simplification is important for understanding the structure of the transfer function.
- MIMO (Multiple Input Multiple Output) systems in other courses!

Numerical Example: 2×2 State-Space to ARMA

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \end{bmatrix}$$

notes

- this numerical example is used to illustrate the conversion from state-space to ARMA.
- this is a 2x2 system, which is manageable by hand and helps students understand the process.
- the matrices A, B, C, and D are chosen for simplicity, but the method is general.

Numerical Example: 2×2 State-Space to ARMA

Step 1: State-Space Equations

$$\begin{cases} x_1^+ = x_1 + 2x_2 + u \\ x_2^+ = 3x_1 + 4x_2 \\ y = x_1 \end{cases}$$

notes

- the state-space equations explicitly using the given matrices.
- x_1^+ and x_2^+ are linear combinations of the states and the input u.
- the output y is simply the first state variable x_1.

Numerical Example: 2×2 State-Space to ARMA

Step 2: Zeta Transform

$$\begin{cases} zX_1(z) = X_1(z) + 2X_2(z) + U(z) \\ zX_2(z) = 3X_1(z) + 4X_2(z) \\ Y(z) = X_1(z) \end{cases}$$

notes

- Apply the Zeta transform to the state-space equations, assuming zero initial conditions.
- the Zeta transform converts differential equations into algebraic equations in z.
- $Y(z) = X_1(z)$, which connects the output directly to the first state variable.

Numerical Example: 2×2 State-Space to ARMA

Step 3: Rearrange in Matrix Form

$$\begin{cases} (zI - A)X(z) = BU(z) \\ Y(z) = CX(z) + DU(z) \end{cases}$$

implies

$$\begin{cases} \begin{bmatrix} z-1 & -2 \\ -3 & z-4 \end{bmatrix} \begin{bmatrix} X_1(z) \\ X_2(z) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} U(z) \\ Y(z) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} X_1(z) \\ X_2(z) \end{bmatrix} \end{cases}$$

notes

- Rearrange the Zeta-transformed equations into matrix form.
- the output equation $Y(z) = CX(z)$ remains simple due to the choice of C.

Numerical Example: 2×2 State-Space to ARMA

Step 4: Solve for $X(z)$

$$X(z) = (zI - A)^{-1} BU(z)$$

$$(zI - A) = \begin{bmatrix} z-1 & -2 \\ -3 & z-4 \end{bmatrix}$$

$$(zI - A)^{-1} = \frac{1}{(z-1)(z-4) - (-2)(-3)} \begin{bmatrix} z-4 & 2 \\ 3 & z-1 \end{bmatrix}$$

$$\det(zI - A) = (z-1)(z-4) - 6 = z^2 - 5z - 2$$

$$(zI - A)^{-1} = \frac{1}{z^2 - 5z - 2} \begin{bmatrix} z-4 & 2 \\ 3 & z-1 \end{bmatrix}$$

notes

- Solve for $X(z)$ by computing $(zI - A)^{-1}$.
- the determinant $\det(zI - A)$, which appears in the denominator of the transfer function, is key.
- the step-by-step computation of the inverse matrix is assumed as a given skill.

Numerical Example: 2×2 State-Space to ARMA

Step 5: Multiply by B

Now, multiply by B:

$$X(z) = \frac{1}{z^2 - 5z - 2} \begin{bmatrix} z-4 & 2 \\ 3 & z-1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} U(z) = \frac{1}{z^2 - 5z - 2} \begin{bmatrix} z-4 \\ 3 \end{bmatrix} U(z)$$

notes

- Multiply $(zI - A)^{-1}$ by B to obtain $X(z)$.
- this step simplifies the expression for $X(z)$.
- $X_1(z)$ and $X_2(z)$ are now expressed in terms of $U(z)$.

Numerical Example: 2×2 State-Space to ARMA

Step 6: Solve for $Y(z)$

Substitute $X(z)$ into the output equation:

$$Y(z) = CX(z) + DU(z) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} X_1(z) \\ X_2(z) \end{bmatrix} = X_1(z)$$

Thus:

$$Y(z) = \frac{z-4}{z^2 - 5z - 2} U(z)$$

notes

- Substitute $X(z)$ into the output equation to find $Y(z)$.
- $Y(z)$ is directly proportional to $X_1(z)$.

Numerical Example: 2×2 State-Space to ARMA

Step 7: Final Result

Transfer function $H(z)$:

$$H(z) = \frac{Y(z)}{U(z)} = \frac{z-4}{z^2-5z-2}$$

and from this we get the ARMA representation of the system as before

notes

- this is the ARMA representation of the system.

From ARMA to SS

notes

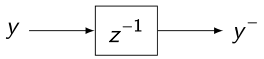
Starting point (blending Zeta notation with time notation)

$$y[k] = \frac{b(z)}{a(z)} u[k] = \frac{b_1 z^{n-1} + \ldots + b_n}{z^n + a_1 z^{n-1} + \ldots + a_n} u[k]$$

notes

- our goal is now that of converting an ARMA model to a state-space representation.
- the starting point is the transfer function in the Zeta domain.
- the numerator and denominator polynomials define the ARMA model.

Building block = the time-delay (block)

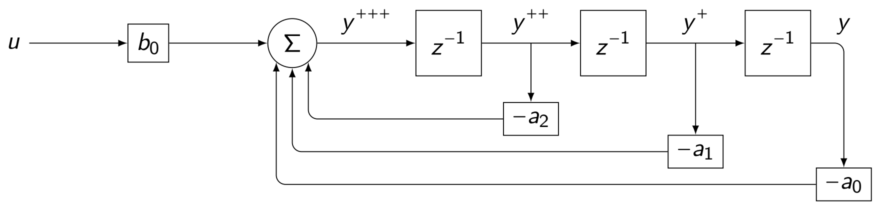


notes

- the time delay block is a fundamental building block for state-space representations.
- this block is key to constructing state variables.

How do we use delays?

$$y^{+++} + a_2 y^{++} + a_1 y^{+} + a_0 y = b_0 u$$
$$\downarrow$$
$$y^{+++} = -a_2 y^{++} - a_1 y^{+} - a_0 y + b_0 u$$



instrumental for later: $x^{[3]} = -a_2 x^{[2]} - a_1 x^{[1]} - a_0 x^{[0]} + b_0 u$

and the state vector is $\left[x^{[2]}, x^{[1]}, x^{[0]}\right]$

notes

- This shows how to rearrange a higher-order differential equation into a form suitable for state-space representation.
- the highest derivative is expressed as a function of lower derivatives and the input.
- this step is crucial for defining the state variables.

Towards SS with a useful trick

$$y[k] = \frac{b(z)}{a(z)} u[k] = \frac{b_1 z^{n-1} + \ldots + b_n}{z^n + a_1 z^{n-1} + \ldots + a_n} u[k] \rightarrow \begin{cases} x^{[0]} = \frac{1}{a(z)} u \\ y = b(z) x^{[0]} \end{cases}$$

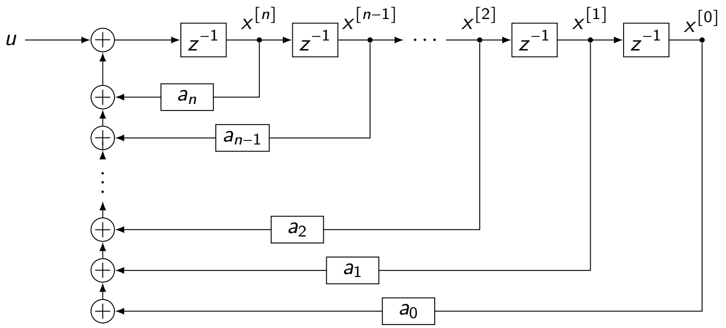
notes

- We then use the trick of defining an intermediate variable $x_n[k]$ to simplify the conversion process.
- $x_n[k]$ is the output of the denominator dynamics driven by the input $u[k]$.
- this trick separates the AR (denominator) and MA (numerator) parts of the system.

This is an AR model on $x^{[0]}$

$$x^{[0]} = \frac{1}{a(z)} u \quad \Longrightarrow \quad a(z) x^{[0]} = u$$

implies

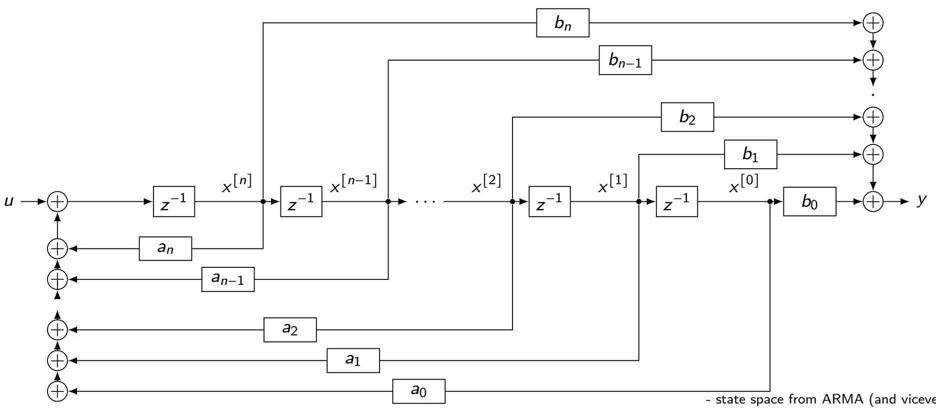


notes

- We now use a block diagram to illustrate the relationship between the state variables.
- the state variables are interconnected through integrators.
- this structure is the foundation of the state-space representation.

Completing the picture (a MA from x_n to y)

$$y[k] = b_n x^{[n]}[k] + \ldots + b_0 x^{[0]}[k]$$



notes

- the output $y[k]$ is constructed as a linear combination of the state variables.
- the coefficients of the linear combination are the numerator coefficients $b_1, b_2, \ldots, b_n$.
- this step completes the state-space representation.

From concepts to formulas

$$\begin{cases} y[k] = b_n x^{[n]}[k] + \ldots + b_0 x^{[0]}[k] \\ x^{[n]+}[k] = -a_n x^{[n]}[k] - \ldots - a_0 x^{[0]}[k] + u[k] \\ x^{[i]+}[k] = x^{[i]}[k] \end{cases} \rightarrow \begin{cases} \boldsymbol{x}^+ & = A\boldsymbol{x} + Bu \\ y & = C\boldsymbol{x} + Du \end{cases}$$

$$\boldsymbol{x}^+ := \begin{bmatrix} x^{[n]+} \\ x^{[n-1]+} \\ x^{[n-2]+} \\ \vdots \\ x^{[0]+} \end{bmatrix} = \begin{bmatrix} -a_n & -a_{n-1} & \ldots & \ldots & -a_0 \\ 1 & 0 & \ldots & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ & \ddots & \ddots & \ddots & \\ & & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x^{[n]} \\ x^{[n-1]} \\ x^{[n-2]} \\ \vdots \\ x^{[0]} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u$$

notes

- This presents the final state-space equations in matrix form.
- the structure of the A matrix is then in control canonical form.
- the B vector has a single non-zero entry, corresponding to the input $u[k]$.

And y?

$$\begin{cases} y[k] = b_n x^{[n]}[k] + \ldots + b_0 x^{[0]}[k] \\ x^{[n]+}[k] = -a_n x^{[n]}[k] - \ldots - a_0 x^{[0]}[k] + u[k] \\ x^{[i]+}[k] = x^{[i]}[k] \end{cases} \rightarrow \begin{cases} \boldsymbol{x}^+ & = A\boldsymbol{x} + Bu \\ y & = C\boldsymbol{x} + Du \end{cases}$$

$$y = \begin{bmatrix} b_n & b_{n-1} & b_{n-2} & \ldots & b_0 \end{bmatrix} \begin{bmatrix} x^{[n]} \\ x^{[n-1]} \\ x^{[n-2]} \\ \vdots \\ x^{[0]} \end{bmatrix}$$

notes

- The output equation is constructed from the state variables.
- the C matrix contains the numerator coefficients $b_0, b_1, \ldots, b_n$.
- this step completes the state-space representation.

From ARMA to state space (in Control Canonical Form)

$$
\begin{cases}
\begin{bmatrix} x^{[n]+} \\ x^{[n-1]+} \\ x^{[n-2]+} \\ \vdots \\ x^{[0]+} \end{bmatrix} = \begin{bmatrix} -a_n & -a_{n-1} & \dots & \dots & -a_0 \\ 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & \\ & & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x^{[n]} \\ x^{[n-1]} \\ x^{[n-2]} \\ \vdots \\ x^{[0]} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u \\
y = \begin{bmatrix} b_n & b_{n-1} & b_{n-2} & \dots & b_0 \end{bmatrix} \begin{bmatrix} x^{[n]} \\ x^{[n-1]} \\ x^{[n-2]} \\ \vdots \\ x^{[0]} \end{bmatrix}
\end{cases}
$$

notes

- The state-space representation in control canonical form.
- the structure of the A matrix becomes upper Hessenberg with a diagonal of ones.
- this form is particularly useful for control design and analysis, you will see it very often.

Matlab / Python implementation

```
[A, B, C, D] = tf2ss([bn .. b0], [1 an .. a0])
```


notes

- the MATLAB/Python function `tf2ss` is used for converting transfer functions to state-space form.
- the input arguments are the numerator and denominator coefficients of the transfer function.
- this function automates the process of deriving the state-space matrices.
- you can use this function to verify your hand calculations only for small examples, at work don't do computations by hand

Summarizing

Determine the state space structure of an discrete time LTI system starting from an ARMA RR

- there are some formulas, that you may simply know by heart, or that you may want to understand
- for understanding there is the need to get how the transformations work, and what is what
- likely the most important point is that to go from ARMA to SS the (likely) most simple strategy is to build the states as a chain of delays, and ladder on top of that

notes

- you should now be able to do this, following the pseudo-algorithm in the itemized list

Most important python code for this sub-module

notes

-

These functions have also their opposite, i.e., tf2ss

- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.ss2tf.html`
- `https://python-control.readthedocs.io/en/latest/generated/control.ss2tf.html`

notes

- in the references you will see much more information than what is given in this module

Self-assessment material

notes

Question 1

Given the discrete-time ARMA model:

$$y^{+++} + a_2 y^{++} + a_1 y^{+} + a_0 y = b_0 u,$$

what is the correct state-space representation in control canonical form?

Potential answers:

I: **(wrong)**

$$\begin{cases} x_1^+ = -a_2 x_1 - a_1 x_2 - a_0 x_3 + b_0 u \\ x_2^+ = x_1 \\ x_3^+ = x_2 \\ y = x_3 \end{cases}$$

II: **(correct)**

$$\begin{cases} x_1^+ = -a_2 x_1 - a_1 x_2 - a_0 x_3 + u \\ x_2^+ = x_1 \\ x_3^+ = x_2 \\ y = b_0 x_3 \end{cases}$$

III: **(wrong)**

notes

- see the associated solution(s), if compiled with that ones :)

Question 2

For the state-space system:

$$\begin{cases} x_1^+ = -3x_1 + 2x_2 + u \\ x_2^+ = x_1 \\ y = 4x_1 + x_2 \end{cases},$$

what is the equivalent ARMA model?

Potential answers:

I: **(wrong)** $y^{++} + 3y^{+} - 2y = 4u^{+} + u$

II: **(correct)** $y^{++} + 3y^{+} - 2y = u^{+} + 4u$

III: **(wrong)** $y^{++} - 3y^{+} + 2y = u^{+} + 4u$

IV: **(wrong)** $y^{++} + 3y^{+} + 2y = 4u^{+} + u$

V: **(wrong)** I do not know

Solution 1:

The ARMA model is derived from $(z^2+3z-2)Y(z) = (z+4)U(z)$, corresponding

notes

- see the associated solution(s), if compiled with that ones :)

Question 3

In discrete-time state-space representations, the delay operator z^{-1} primarily:

Potential answers:

- I: **(wrong)** Approximates continuous-time integration
- II: **(correct)** Implements the time-shift operation $x[k] \rightarrow x[k-1]$
- III: **(wrong)** Adds stochastic noise to the system
- IV: **(wrong)** Reduces computational complexity
- V: **(wrong)** I do not know

Solution 1:

The z^{-1} operator represents a unit delay in discrete-time systems, equivalent to the time-shift operation. This is fundamental for implementing state updates in difference equations.

notes

- see the associated solution(s), if compiled with that ones :)

Question 4

The control canonical form's state matrix A always:

Potential answers:

- I: **(wrong)** Is diagonal with poles on the diagonal
- II: **(correct)** Has AR coefficients in its first row and shifted identity below
- III: **(wrong)** Makes the B matrix identical to $C^\top$
- IV: **(wrong)** Minimizes the number of nonzero elements
- V: **(wrong)** I do not know

Solution 1:

Control canonical form structures A with $-a_n$ to $-a_0$ in the first row and shifted identity submatrix, ensuring direct mapping from ARMA coefficients. This form guarantees controllability.

notes

- see the associated solution(s), if compiled with that ones :)

Question 5

When converting state-space to ARMA via Z-transform, the operator $(zI - A)^{-1}$:

Potential answers:

I:	**(wrong)**	Directly gives the system's impulse response
II:	**(correct)**	Is the resolvent matrix needed to solve for $X(z)$
III:	**(wrong)**	Always results in a diagonalizable matrix
IV:	**(wrong)**	Can be omitted if $D \neq 0$
V:	**(wrong)**	I do not know

Solution 1:

The resolvent matrix $(zI - A)^{-1}$ is essential for solving $X(z) = (zI - A)^{-1} BU(z)$, which is then used to derive the transfer function $H(z) = C(zI - A)^{-1} B + D$.

notes

- see the associated solution(s), if compiled with that ones :)

Recap of sub-module "state space from ARMA (and viceversa)"

- one can go from ARMA to state space and viceversa
- we did not see this, but watch out that the two representations are not equivalent: there are systems that one can represent with state space and not with ARMA, and viceversa
- typically state space is more interpretable, and tends to be the structure used when doing model predictive control

notes

- the most important remarks from this sub-module are these ones