Model Predictive Control and Reinforcement Learning
– Lecture 2: Dynamic Systems and Simulation –

Joschka Boedecker and Moritz Diehl

University Freiburg

July 25, 2021

Overview

Acknowledgement

Slides contain some figures from slides by Rien Quirynen and from the textbook
Model Predictive Control: Theory, Computation, and Design (by Rawlings, Mayne, and Diehl)

Optimal Control based on Dynamic System Models

- optimal control = optimization of dynamic systems
- each optimal control problem (OCP) is characterized by three ingredients:
  - dynamic system model
  - constraints
  - objective function, i.e., cost or reward

Optimal Control based on Dynamic System Models

- optimal control = optimization of dynamic systems
- each optimal control problem (OCP) is characterized by three ingredients:
  - **dynamic system model** (focus of this talk)
  - constraints
  - objective function, i.e., cost or reward

Dynamic System Models

- system model describes evolution of system as function of
  - system **state** s from state space $\mathbb{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)
  - control **action** a from action space $\mathbb{A} \subset \mathbb{R}^{n_a}$ (or $\subset \mathbb{Z}^{n_a}$ for discrete actions)
  - random **disturbance** ϵ from some disturbance space $\mathbb{D}$
- examples:
  - **stochastic discrete time system**, for $k = 0, 1, 2, \ldots$

$$\boxed{s_{k+1} = f(s_k, a_k, \epsilon_k)} \quad \text{with "evolution function" } f : \mathbb{S} \times \mathbb{A} \times \mathbb{D} \to \mathbb{S}$$

  - deterministic continuous time **ordinary differential equation (ODE)**, for $t \in [0, \infty)$

$$\boxed{\frac{\mathrm{d}s}{\mathrm{d}t}(t) = f_\mathrm{c}(s(t), a(t))} \quad \text{with "right hand side function" } f_\mathrm{c} : \mathbb{S} \times \mathbb{A} \to \mathbb{R}^{n_s}$$

(stochastic continuous time systems need intricate notation and are therefore omitted here)

Notation for Ordinary Differential Equation (ODE) Models

- denote $\frac{\mathrm{d}s}{\mathrm{d}t}(t)$ by $\dot{s}(t)$
- drop time argument, abbreviate $\dot{s}(t) = f_\mathrm{c}(s(t), a(t))$ by

$$\dot{s} = f_\mathrm{c}(s, a)$$

- In this course, we use the RL notation: s for state and a for control action
- But in control engineering, one uses: x for state and u for control action, i.e.,

$$\dot{x} = f_\mathrm{c}(x, u)$$

(this notation might accidentally "slip through" on some slides)

ODE Example: Harmonic Oscillator

Mass m with spring constant k and friction coefficient β:

$$\begin{aligned}
\dot{x}_1(t) &= x_2(t) \\
\dot{x}_2(t) &= -\frac{k}{m}(x_2(t) - u(t)) \quad - \frac{\beta}{m} x_1(t)
\end{aligned}$$

- state $x(t) \in \mathbb{R}^2$
- position of mass $x_1(t)$ $\longleftarrow$ measured
- velocity of mass $x_2(t)$
- control action: spring position $u(t) \in \mathbb{R}$ $\longleftarrow$ manipulated

Can summarize as $\dot{x} = f_c(x, u)$ with

$$f_c(x,u) = \begin{bmatrix} x_2 \\ -\frac{k}{m}(x_2 - u) - \frac{\beta}{m} x_1 \end{bmatrix}$$

ODE Example: Harmonic Oscillator

Mass m with spring constant k and friction coefficient β:

$$\begin{aligned} \dot{s}_1(t) &= s_2(t) \\ \dot{s}_2(t) &= -\frac{k}{m}(s_2(t) - a(t)) \quad - \frac{\beta}{m} s_1(t) \end{aligned}$$

- state $s(t) \in \mathbb{R}^2$
- position of mass $s_1(t)$ $\longleftarrow$ measured
- velocity of mass $s_2(t)$
- control action: spring position $a(t) \in \mathbb{R}$ $\longleftarrow$ manipulated

Can summarize as $\dot{s} = f_c(s, a)$ with

$$f_c(s, a) = \begin{bmatrix} s_2 \\ -\frac{k}{m}(s_2 - a) - \frac{\beta}{m} s_1 \end{bmatrix}$$

Some ODE Examples - what are their state vectors?

- Pendulum
- Hot plate with pot
- Continuously Stirred Tank Reactors (CSTR)
- Robot arms
- Moving robots
- Race cars
- Airplanes in free flight

Overview

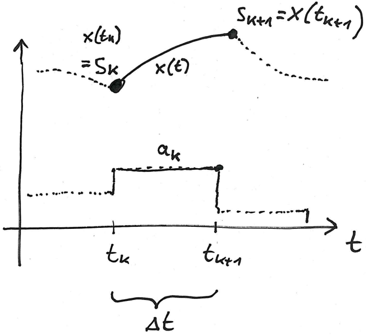
From Continuous to Discrete Time via Direct Multiple Shooting
[Bock & Plitt 1984]

Transform continuous $\boxed{\dot{s}(t) = f_c(s(t), a(t))}$ into discrete time $\boxed{s_{k+1} = f(s_k, a_k)}$ as follows:

1. define $s_k := s(t_k)$ on **equidistant time grid** $t_k = k\,\Delta t$ with sampling time Δt
2. use **zero order hold** control $a(t) = a_k$ on $t \in [t_k, t_{k+1}]$
3. use **numerical simulation** to compute ODE solution $x(t) \equiv x(t; s_k, a_k)$ satisfying

$$\begin{aligned} x(t_k) &= s_k \\ \dot{x}(t) &= f_c(x(t), a_k) \quad \text{for} \quad t \in [t_k, t_{k+1}] \end{aligned}$$

4. define $f(s_k, a_k) := x(t_{k+1}; s_k, a_k)$

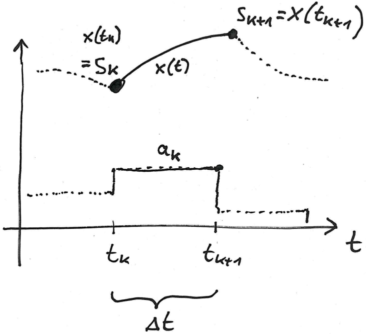


From Continuous to Discrete Time via Direct Multiple Shooting
[Bock & Plitt 1984]

Transform continuous $\boxed{\dot{s}(t) = f_c(s(t), a(t))}$ into discrete time $\boxed{s_{k+1} = f(s_k, a_k)}$ as follows:

Exact ODE solution

$$
\begin{aligned}
x(0) &= s,\\
\dot{x}(t) &= f_c(x(t), a),\\
&\qquad\qquad \text{for} \quad t \in [0, \Delta t]\\
f(s, a) &:= x(\Delta t)
\end{aligned}
$$



How to simulate ODE numerically?

Numerical Simulation/Integration, Three Examples

- simplest (but not recommended) implementation is a single step of an Euler integrator:

$$\boxed{f(s,a) := s + \Delta t \; f_\mathrm{c}(s,a)}$$

- more accurate are N steps of an Euler integrator:

$x_0 := s$
for $i = 0$ to $N-1$ do
$\quad x_{i+1} := x_i + (\Delta t/N) f_\mathrm{c}(x_i, a)$
$f(s,a) := x_N$

- more efficient are higher order **Runge Kutta (RK)** methods, e.g. a single RK4 step:

$$
\begin{aligned}
v_1 &:= f_\mathrm{c}(s,a)\\
v_2 &:= f_\mathrm{c}(s + (\Delta t/2)\, v_1, a)\\
v_3 &:= f_\mathrm{c}(s + (\Delta t/2)\, v_2, a)\\
v_4 &:= f_\mathrm{c}(s + \Delta t\, v_3, a)\\
f(s,a) &:= s + (\Delta t/6)\,(v_1 + 2v_2 + 2v_3 + v_4)
\end{aligned}
$$

Euler vs 4th Order Runge Kutta Method (RK4) for Test Problem

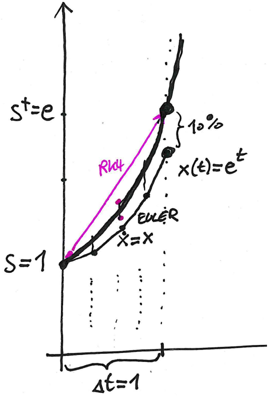
Aim: solve $\dot{s} = s + a$ for $\Delta t = 1, s = 1, a = 0$. Exact solution is $f(s,a) = e = 2.718$.

- Four Euler steps give

$$
\begin{aligned}
x_0 &:= 1 \\
x_1 &:= x_0 + 1/4 x_0 \quad [= (1+1/4)x_0\,] \\
x_2 &:= (1+1/4)x_1 \\
x_3 &:= (1+1/4)x_2 \\
x_4 &:= (1+1/4)x_3 \\
f_{\text{Euler}}(s,a) &:= x_4 \quad [= (1{+}1/4)^4 = 2.441], \text{ error} > 10\%
\end{aligned}
$$

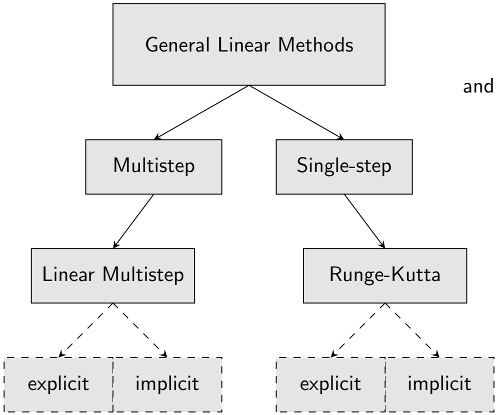
- One RK4 step gives

$$
\begin{aligned}
v_1 &:= 1 \\
v_2 &:= 1 + 1/2 v_1 \quad [= 6/4\,] \\
v_3 &:= 1 + (1/2)v_2 \quad [= 7/4\,] \\
v_4 &:= 1 + v_3 \quad [= 11/4\,] \\
f_{\text{RK4}}(s,a) &:= 1{+}(1/6)\;(v_1{+}2v_2{+}2v_3{+}v_4)\;[= 2.708\,]
\end{aligned}
$$



RK4 is 27x more accurate than Euler for same number $M = 4$ of function evaluations

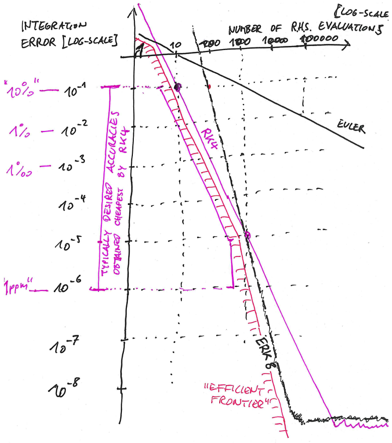
Classes of Numerical Simulation Methods



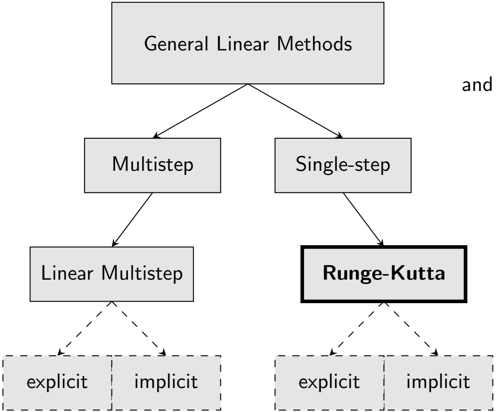
and others ...

Fourth order RK method most efficient for typically desired accuracies

- each integration method is characterized by
  - integration order P and
  - number of internal stages S
- can increase accuracy by more integration steps N
- total number of function evaluations is $M = N \cdot S$
- integration error proportional to M^{-P}
- for small M, low order methods are most accurate, e.g., Euler with $P = 1$
- for large M, high order methods are more accurate
- humans typically want errors smaller than 10%, but rarely smaller than 10^{-6}
- accidentally, this favours the RK4 method ($P = 4$)



Classes of Numerical Simulation Methods



and others ...

Discretization equations for general Runge Kutta (RK) methods

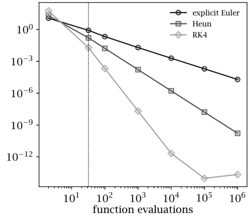
Exact ODE solution

$$
\begin{aligned}
x(0) &= s,\\
\dot{x}(t) &= v(t)\\
v(t) &= f_\mathrm{c}(x(t), a),\\
&\qquad \text{for} \quad t \in [0, \Delta t]\\
f(s, a) &:= x(\Delta t)
\end{aligned}
$$

N steps of general RK method with S stages

$$
\begin{aligned}
x_0 &= s, \quad x_{k+1} = x_k + h \textstyle\sum_{j=1}^{S} b_j v_{k,j}\\
x_{k,i} &= x_k + h \textstyle\sum_{j=1}^{S} a_{ij} v_{k,j}\\
v_{k,i} &= f_\mathrm{c}(x_{k,i}, a),\\
&\qquad \text{for} \quad i = 1, \ldots, S, \quad k = 0, \ldots, N-1\\
f(s, a) &:= x_N
\end{aligned}
$$

- a_{ij} and b_j are **Butcher tableau entries** of (potentially implicit) Runge Kutta method
- step length $h := \Delta t / N$; intermediate states $x_k, x_{k,i}, v_{k,i} \in \mathbb{R}^{n_s}$ with integration step index $k \in \{0, 1, \ldots, N\}$ and RK stage index $i, j \in \{1, \ldots, S\}$
- N nonlinear equation systems with each $2Sn_s$ equations in $2Sn_s$ unknowns $(x_{k,i}, v_{k,i})$
- solved by Newton's method (or imposed as equality constraints in optimization)



Euler

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Heun

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$

RK4

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

Implicit Euler

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Midpoint rule (GL2)

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$$

Gauss-Legendre of order 4 (GL4)

$$\begin{array}{c|cc} 1/2-\sqrt{3}/6 & 1/4 & 1/4-\sqrt{3}/6 \\ 1/2+\sqrt{3}/6 & 1/4+\sqrt{3}/6 & 1/4 \\ \hline & 1/2 & 1/2 \end{array}$$

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ c_2 & a_{21} & \cdots & a_{2s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

Intermediate Milestone: Deterministic State Space Models

From now on, throughout the course, we exclusively focus on discrete time models

$$s_{k+1} = f(s_k, a_k)$$

with integer time index $k = 0, 1, 2, \ldots$. We often simplify notation to

$$\boxed{s^+ = f(s, a)}$$

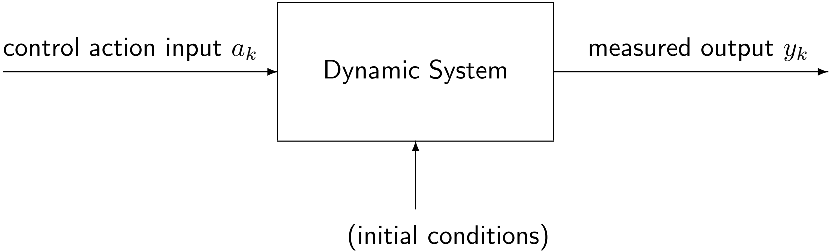
Aim of optimal feedback control (including both MPC and RL) is to design a map, or **policy**, $\pi : \quad \mathbb{S} \to \mathbb{A}, \quad s \mapsto a := \pi(s)$ such that **closed-loop system** $\boxed{s^+ = f(s, \pi(s))}$ has desirable properties, such as respecting constraints and minimizing a cost.

In practice, however, we might not be able to directly measure the state s ...

Overview

The (realistic) Input Output Perspective

- In practice, we cannot measure the state. And the state representation is not even unique.
- A system model should allow us to predict, for any horizon length N and sequence of control actions $(a_1, \ldots, a_N)$, the sequence of measured **outputs** $(y_0, \ldots, y_N)$.
- Typically, we need to also specify some **initial conditions** (e.g. the initial state s_0)



Two Ways to Represent Deterministic Systems with Outputs

- State Space Models with outputs:

$$\begin{aligned} s_{k+1} &= f(s_k, a_k) \\ y_k &= g(s_k, a_k) \qquad \text{for} \quad k = 0, 1, 2, \dots \end{aligned}$$

  Initial conditions = initial state s_0.

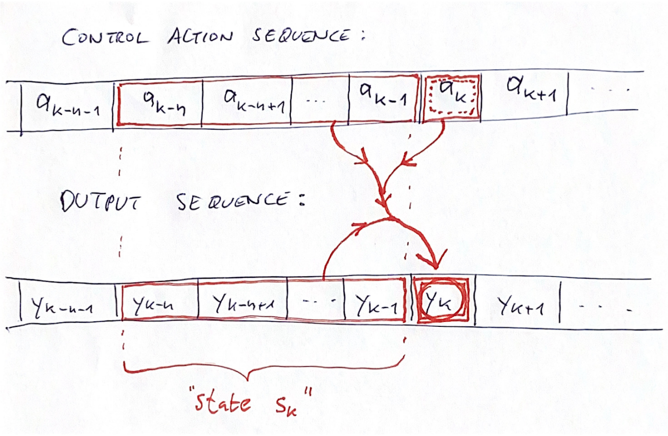
- Input Output Models (of order n):

$$y_k = h(y_{k-1}, \dots, y_{k-n}, a_k, \dots, a_{k-n}) \qquad \text{for} \quad k = n, n+1, n+2, \dots$$

  Initial conditions: $y_0, \dots, y_{n-1}$ and $a_0, \dots, a_{n-1}$.

Recurrence Equation in Input Output Models of order n

Visualization of recurrence $y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, a_{k-1}, \ldots, a_{k-n})$:



State Space Form of Input Output Models

- can **always transform input-output to state-space models:**
- state: $s_k = (y_{k-1}, a_{k-1}, \ldots, y_{k-n}, a_{k-n})$ (defined for $k \geq n$)
- state transition $s \mapsto s^+ = f(s, a)$ described by

$$s_k = \begin{bmatrix} y_{k-1} \\ a_{k-1} \\ \vdots \\ y_{k-n+1} \\ a_{k-n+1} \\ y_{k-n} \\ a_{k-n} \end{bmatrix} \mapsto s_{k+1} = \begin{bmatrix} y_k \\ a_k \\ y_{k-1} \\ a_{k-1} \\ \vdots \\ y_{k-n+1} \\ a_{k-n+1} \end{bmatrix} = f(s_k, a_k) := \begin{bmatrix} h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}) \\ a_k \\ y_{k-1} \\ a_{k-1} \\ \vdots \\ y_{k-n+1} \\ a_{k-n+1} \end{bmatrix}$$

- output equation: $y_k = g(s_k, a_k) := h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n})$.
- conversely, we can **sometimes transform state-space to input-output models**, e.g. in case of observable and controllable linear time invariant (LTI) models

Linear Time Invariant (LTI) Input Output Models

- Difference equation for **Auto Regressive models with eXogenous inputs (ARX):**

$$\boxed{y_k = c_1 y_{k-1} + \ldots + c_n y_{k-n} \quad + \quad b_0 a_k + \ldots + b_n a_{k-n}}$$

  for $k = n, n+1, \ldots$, with initial conditions: $y_0, \ldots, y_{n-1}$ and $a_0, \ldots, a_{n-1}$.
- also called **Infinite Impulse Response (IIR)** model (if some c_i coefficients are nonzero)
- If all $c_i = 0$ we speak of **Finite Impulse Response (FIR)** models:

$$y_k = b_0 a_k + \ldots + b_n a_{k-n}$$

- There exist also auto regressive (AR) models without inputs:

$$y_k = c_1 y_{k-1} + \ldots + c_n y_{k-n}$$

  Example: Fibonacci numbers 1,1,2,3,5,8,13,21, ... (with $c_1 = c_2 = 1$ and $y_0 = y_1 = 1$)

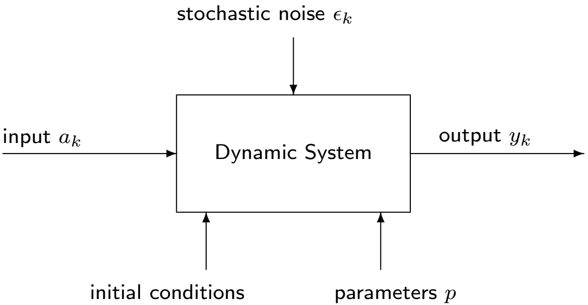
Some ODE Examples - what can be measured ?

- Pendulum
- Hot plate with pot
- Continuously Stirred Tank Reactors (CSTR)
- Robot arms
- Moving robots
- Race cars
- Airplanes in free flight

Overview

General Stochastic Models

- in reality, we always have some random noise ϵ_k, e.g., disturbances or measurement errors
- also, we usually have unknown, but constant system parameters p



(parameters can be seen as states that obey the dynamics $p_{k+1} = p_k$ and will often be omitted)

Stochastic Systems in State Space and Input Output Form

General Form (with random ϵ_k):

Stochastic State Space Model

$$s_{k+1} = f(s_k, a_k, \epsilon_k)$$
$$y_k = g(s_k, a_k, \epsilon_k)$$

Stochastic Input Output Model

$$y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}, \epsilon_k, \ldots, \epsilon_{k-n})$$

Special Cases:

- State Noise and Output Errors:

$$s_{k+1} = f(s_k, a_k) + \epsilon_k^{\mathrm{SN}}$$
$$y_k = g(s_k, a_k) + \epsilon_k^{\mathrm{OE}}$$

- Equation Errors:

$$y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}) + \epsilon_k^{\mathrm{EE}}$$

(note: different than output error)

MPC needs System Identification and State Estimation

Prior to implementing an MPC controller, one needs to address two tasks:

- **System Identification (offline):**
  use a long sequence of recorded input and output data, $(a_0, \ldots, a_N)$ and $(y_0, \ldots, y_N)$, to identify parameters p using e.g. least squares optimization or subspace identification

- **State Estimation (online):**
  estimate the state s_k by using the previous control actions $(..., a_{k-2}, a_{k-1})$ and the past measurements $(..., y_{k-2}, y_{k-1})$ using e.g. Extended Kalman Filter (EKF) or moving horizon estimation (MHE) (MHE uses a fixed window of past data for fitting)

Learning-based MPC typically refers to an online model adaptation, i.e., to estimating parameters online (for which MHE is particularly suitable) ("learning a model" $=$ "system identification")

Note: need state estimation only for *partially observable markov decision processes (POMDP)*

Fully and Partially Observable Markov Decision Processes (MDP)

State Space View:

Partially Observable MDP

$$
\begin{aligned}
s_{k+1} &= f(s_k, a_k, \epsilon_k) \\
y_k &= g(s_k, a_k, \epsilon_k)
\end{aligned}
$$

with independent identically distributed ϵ_k

Fully Observable MDP

$$
\begin{aligned}
s_{k+1} &= f(s_k, a_k, \epsilon_k) \\
y_k &= s_k
\end{aligned}
$$

with $y_k \in \mathbb{R}^{n_s}$

Probabilistic View:

Partially Observable MDP

$$
\begin{aligned}
&P_{\text{state}}(s_{k+1} | s_k, a_k) \\
&P_{\text{meas}}(\; y_k \; | s_k, a_k)
\end{aligned}
$$

with probability density functions $P.(\cdot)$

Fully Observable MDP

$$
\begin{aligned}
&P_{\text{state}}(s_{k+1} | s_k, a_k) \\
&P_{\text{meas}}(\; y_k \; | s_k, a_k) \quad = \quad \delta(y_k - s_k)
\end{aligned}
$$

with Dirac's Delta function $\delta(\cdot)$ in $\mathbb{R}^{n_s}$

Input output (I/O) models avoid need for state estimation

- We can avoid estimation task by assuming input-output (I/O) models of fixed order n
- This assumption leads to a **fully observable** markov decision process (MDP)
- State s_k at time k is then given by $s_k = (y_{k-1}, a_{k-1}, \ldots, y_{k-n}, a_{k-n})$
- Reinforcement Learning (RL) algorithms often use I/O-models ("end-to-end learning")
- I/O-models also used in some *linear MPC* implementations based on LTI models, e.g.

$$y_k \quad = \quad \sum_{i=0}^{n} b_i \, a_{k-i} \; + \; (y_{k-1} - \sum_{i=0}^{n} b_i \, a_{k-i-1}) \; + \; \epsilon_k$$

- I/O-models also used for nonlinear black-box MPC or model-based RL which use neural networks for the mapping $y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n})$

Summary

- We distinguish different model types
  - continuous vs discrete state and control
  - continuous vs discrete time
  - linear vs nonlinear
  - state space vs input output
  - deterministic vs stochastic
  - fully or partially observable
    (not to be confused with "observability" in systems theory)
- We transform differential equations to discrete time via *numerical simulation*
- We denote *deterministic discrete time models* and *Markov Decision Processes (MDP)* by

$$\boxed{s^+ = f(s, a)} \quad \text{and} \quad \boxed{P(s^+|s, a)}$$

with state $s \in \mathbb{R}^{n_s}$ and control action $a \in \mathbb{R}^{n_a}$