Table of Contents I

notes

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

how to get a RR from an ODE

notes

Contents map

developed content units	taxonomy levels
Euler forward discretization	u1, e1
Euler backwards discretization	u1, e1

prerequisite content units	taxonomy levels
ODE	u1, e1
RR	u1, e1

notes

Main ILO of sub-module "how to get a RR from an ODE"

Discretize ODEs and thus find recurrent relations that approximate ODEs by means of Euler schemes

Discuss which factors affect the validity of Euler discretization schemes and **exemplify** which problems may occur in practice

notes

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

Roadmap

- why do we need to numerically simulate?
- Euler methods
- pros and cons
- connections with linearization

notes

- in this section we will discuss a bit more about how to simulate a dynamical model
- we will already mentioned this method before, but now we discuss it in more details, so to fix knowledge
- the good point is that now that we saw linearization, we can say something more about the method
- you can find a good explanation plus also some code in its wiki page, `https://en.wikipedia.org/wiki/Euler_method`
- for a reference on python ODE solvers check `https://pythonnumericalmethods.berkeley.edu/notebooks/chapter22.06-Python-ODE-Solvers.html`, while for Matlab check `https://se.mathworks.com/help/matlab/math/choose-an-ode-solver.html`

Our computers are digital machines, but the ODEs are "analogic" objects

$$\begin{cases} \dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} &= \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) \end{cases}$$

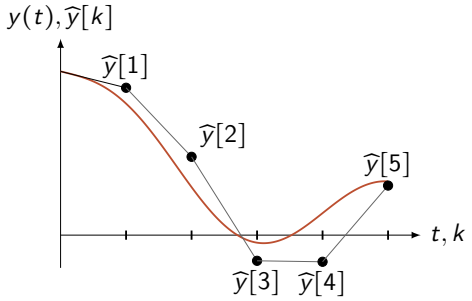
the need is for discretizing these objects, both in time and in space

notes

- we start recalling that our computers do not work with continuous time or continuous numbers
- to simulate such a system we need to introduce some approximation
- no matter how, note that there will always be some approximation. Maybe insignificant, but numerically solving ODEs require discretizing both time and numbers

Simulating nonlinear systems = solving the ODE numerically and in a discrete way

i.e., use the fact that we know that $\dot{y} = f(y, u)$, we know the whole u, and we know the initial condition $y(0)$ to compute a series of points $\widehat{y}[0], \widehat{y}[1], \widehat{y}[2], \ldots, \widehat{y}[H]$, that approximate the whole trajectory $y(0:H)$ with H a user-defined prediction horizon:



notes

- the main intuition is to compute a set of points that correspond to a noisy sampling of the actual trajectory of the system
- to do this one has as available information the fact that f provides information about where the state is going

The simplest numerical solver: Euler's (forward) method

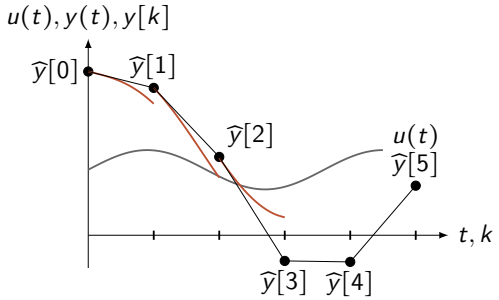
step 0: $\widehat{y}[0] = y(0)$

step 1: $\widehat{y}[1] = \widehat{y}[0] + f\big(\widehat{y}[0], u(0)\big)T$

step 2: $\widehat{y}[2] = \widehat{y}[1] + f\big(\widehat{y}[1], u(T)\big)T$

step 3: $\widehat{y}[3] = \widehat{y}[2] + f\big(\widehat{y}[2], u(2T)\big)T$

$\vdots \quad \vdots \quad \vdots \quad \vdots$

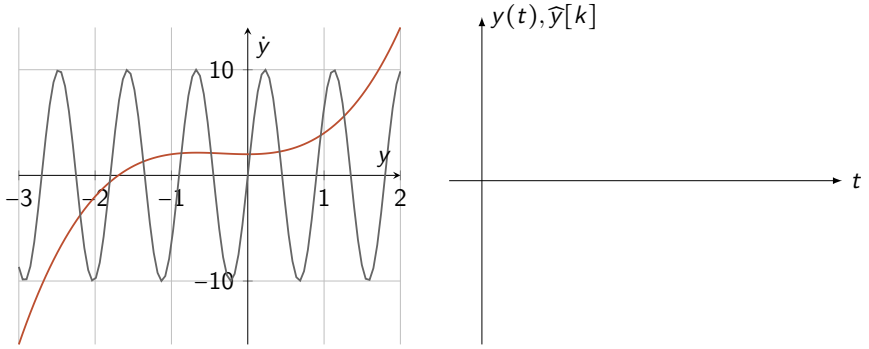


notes

- the easiest way is to proceed by using first order approximations as follows
- this corresponds to follow the direction indicated by f for a determined period

Tradeoffs:

- the more "gentle" $\dot{y} = f(y)$, the more accurate the approximated trajectory $\widehat{y}$
- the smaller T, the more accurate $\widehat{y}$ & the longer the CPU time
- the longer the prediction horizon in $y(0:H)$, the less accurate the final prediction



notes

- there are some known tradeoffs, that somehow are as intuition drives
- for example if f varies a lot between two neighboring points in the phase plane, then a small difference in where the approximation makes the trajectory lands will make f vary a lot, that means increasing even more in where the approximation will make the rest of the trajectory land later on

Known problem

Euler forward may be numerically unstable, especially for "stiff ODEs" (i.e., ODEs for which some terms that can lead to rapid variation in the solution). Will be seen extensively in following courses!

notes

- remember this name: "stiff ODE" is a synonym of f that has a high sensitivity to some of its parameters
- as intuition says, with a stiff ODE we have a situation as before – it is like having a non-gentle state update map
- for this type of ODEs there is the need for choosing some other solver. If you find yourself in this situation check for example `https://se.mathworks.com/help/matlab/math/choose-an-ode-solver.html`

Note: Euler's forward is only one algorithm of many

Euler Forward:

$$\widehat{y}[k+1] = \widehat{y}[k] + f\big(\widehat{y}[k], u(kT)\big)T$$

Euler Backward:

$$\widehat{y}[k+1] = \widehat{y}[k] + f\big(\widehat{y}[k+1], u((k+1)T)\big)T$$

can be generalized to Runge-Kutta methods, with known tradeoffs and robustness properties; they will be studied in following courses

notes

- actually what we have seen before is only one of the potential strategies that one may follow
- there is actually plenty of ODE solvers, it is an active field of research
- different solvers have different properties, and one may make them very tailored for some specific types of ODEs. I.e., make solvers that simulate very well a certain type of systems, but worse some other ones
- for an extensive introduction (but you will see this topic also later on in your study course) check `https://issc.uj.ac.za/appliedmaths/honours/apm0137/2018/Runge%20Kutta%20Notes_new.pdf`

the specific case of ARMA models

notes

From Continuous-Time to Discrete-Time ARMA Models (with backwards)

from

$$\frac{d^n y(t)}{dt^n} + a_{n-1}\frac{d^{n-1} y(t)}{dt^{n-1}} + \cdots + a_0 y(t) = b_m \frac{d^m u(t)}{dt^m} + \cdots + b_0 u(t)$$

substitute

$$\frac{d\star}{dt} \mapsto \frac{1-q^{-1}}{T}$$

$$\frac{d^2\star}{dt^2} \mapsto \left(\frac{1-q^{-1}}{T}\right)^2$$

$$\frac{d^3\star}{dt^3} \mapsto \left(\frac{1-q^{-1}}{T}\right)^3$$

$$\vdots$$

notes

- Continuous-time ARMA models are differential equations driven by white noise.
- The goal is to discretize this model using Euler's backward method.

From Continuous-Time to Discrete-Time ARMA Models (with forward)

from

$$\frac{d^n y(t)}{dt^n} + a_{n-1}\frac{d^{n-1} y(t)}{dt^{n-1}} + \cdots + a_0 y(t) = b_m \frac{d^m u(t)}{dt^m} + \cdots + b_0 u(t)$$

substitute

$$\frac{d\star}{dt} \mapsto \frac{q-1}{T}$$

$$\frac{d^2\star}{dt^2} \mapsto \left(\frac{q-1}{T}\right)^2$$

$$\frac{d^3\star}{dt^3} \mapsto \left(\frac{q-1}{T}\right)^3$$

$$\vdots$$

notes

- Continuous-time ARMA models are differential equations driven by white noise.
- The goal is to discretize this model using Euler's forward method.

Example with backwards

$$\dot{y}(t) + ay(t) = bu(t)$$

becomes

$$\frac{\widehat{y}[k] - \widehat{y}[k-1]}{T} + a\widehat{y}[k] = bu[k]$$

or, rearranging,

$$\left(\frac{1}{T} + a\right)\widehat{y}[k+1] + \left(-\frac{1}{T}\right)\widehat{y}[k] = bu[k]$$

notes

- here we are using Euler's backward method only on the only derivative we see

Example with forward

$$\dot{y}(t) + ay(t) = bu(t)$$

becomes

$$\frac{\widehat{y}[k+1] - \widehat{y}[k]}{T} + a\widehat{y}[k] = bu[k]$$

or, rearranging,

$$\left(\frac{1}{T}\right)\widehat{y}[k+1] + \left(-\frac{1}{T} + a\right)\widehat{y}[k] = bu[k]$$

notes

- here we are using Euler's backward method only on the only derivative we see

Example with backwards

$$\ddot{y}(t) + a_1\dot{y}(t) + a_0 y(t) = b_1\dot{u}(t) + b_0 u(t)$$

becomes

$$\widehat{y}[k]\left(\frac{1-q^{-1}}{T}\right)^2 + a_1\widehat{y}[k]\left(\frac{1-q^{-1}}{T}\right) + a_0\widehat{y}[k] = b_1 u[k]\left(\frac{1-q^{-1}}{T}\right) + b_0 u[k]$$

$\Longrightarrow$ rearranging $=$ "algebra"

notes

- here we are using Euler's backward method only on the only derivative we see
- arriving at the final numbers is actually just a matter of computations

Discussion

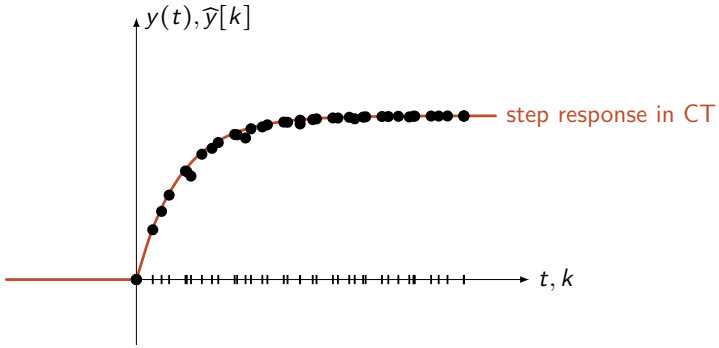
At work, if you have to discretize an ODE of order bigger than one, will you do by hand, or will you use `https://python-control.readthedocs.io/en/latest/generated/control.matlab.c2d.html`?

notes

- don't do things by hand if they are not trivial

What happens as T decreases?

Example



notes

- As T decreases, the discrete-time solution (dashed lines) converges to the continuous-time solution (solid blue line).
- This demonstrates the accuracy of Euler's backward method for smaller time steps.

Summarizing

Discretize ODEs and thus find recurrent relations
that approximate ODEs by means of Euler schemes

Discuss which factors affect the validity of Euler discretization
schemes and **exemplify** which problems may occur in practice

- backwards or forward Euler = good initial choice
- stiffness of the starting ODE is an issue
- the longer the time horizon to predict, the more problems you will have

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

C2D

- `https://python-control.readthedocs.io/en/latest/generated/control.matlab.c2d.html`
- `https://www.mathworks.com/help/control/ref/dynamicsystem.c2d.html`

notes

- there is a lot under the hood!

Self-assessment material

notes

Question 1

Which of the following is the most important general key factor affecting the accuracy of Euler's forward method when discretizing ODEs?

Potential answers:

I: **(correct)** The step size T
II: **(wrong)** The initial condition $y(0)$
III: **(wrong)** The type of input signal $u(t)$
IV: **(wrong)** The order of the ODE
V: **(wrong)** I do not know

Solution 1:

The step size T is in general the most important key factor affecting the accuracy of Euler's forward method. A smaller T generally leads to a more accurate approximation of the ODE solution, but it also increases computational cost.

notes

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

Question 2

What is a common issue when using Euler's forward method to solve stiff ODEs?

Potential answers:

I: **(correct)** Numerical instability
II: **(wrong)** Increased computational efficiency
III: **(wrong)** Exact solution with no error
IV: **(wrong)** Reduced sensitivity to initial conditions
V: **(wrong)** I do not know

Solution 1:

Stiff ODEs often lead to numerical instability when using Euler's forward method due to the high sensitivity of the solution to small changes in the state.

notes

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

Question 3

Which of the following is a tradeoff when using Euler's forward method for ODE discretization?

Potential answers:

I: **(wrong)** Smaller step size T reduces accuracy and increases CPU time
II: **(wrong)** Larger step size T reduces accuracy and increases CPU time
III: **(correct)** Smaller step size T increases accuracy but also increases CPU time
IV: **(wrong)** Larger step size T increases accuracy and reduces CPU time
V: **(wrong)** I do not know

Solution 1:

A smaller step size T improves the accuracy of the Euler forward method but increases the computational cost (CPU time) due to the need for more iterations.

notes

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

Question 4

What is the main difference between Euler's forward and backward methods?

Potential answers:

I: **(wrong)** Euler's backward method is always more accurate than Euler's forward method

II: **(wrong)** Euler's backward method does not require an initial condition

III: **(correct)** Euler's backward method uses the derivative at the next time step

IV: **(wrong)** Euler's backward method cannot be used for stiff ODEs

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

Solution 1:

Euler's backward method uses the derivative at the next time step, making it an implicit method. This often provides better stability for stiff ODEs compared to Euler's forward method.

notes

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

Question 5

When discretizing a continuous-time ARMA model using Euler's backward method, what substitution is made for the first derivative $\frac{dy(t)}{dt}$?

Potential answers:

I: **(correct)** $\frac{1-q^{-1}}{T}$

II: **(wrong)** $\frac{q-1}{T}$

III: **(wrong)** $\frac{q+1}{T}$

IV: **(wrong)** $\frac{1+q^{-1}}{T}$

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

Solution 1:

When using Euler's backward method, the first derivative $\frac{dy(t)}{dt}$ is substituted

notes

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

Recap of sub-module "how to get a RR from an ODE"

- there are several ways of solving ODEs in a computer
- Euler is the most simple one, but it does not work well with stiff ODEs
- more advanced schemes have better numerical properties

notes

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