



Numerical methods for dynamical systems

Julien Alexandre dit Sandretto

Department U2IS
ENSTA Paris
INF6561 2022-2023



Context

To cope with the increasing complexity of systems, more and more models are used.

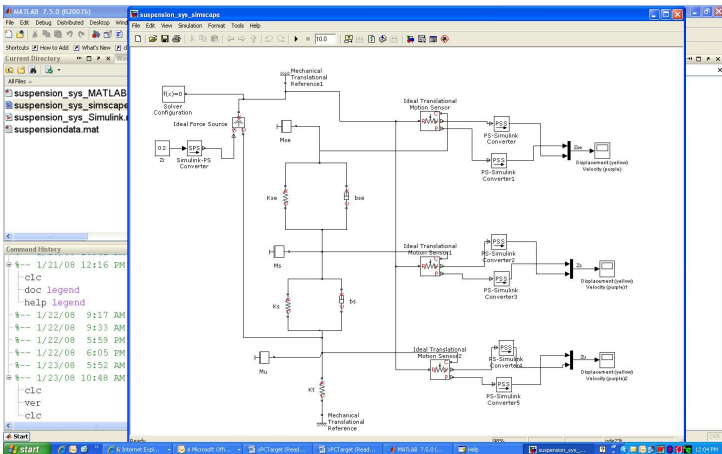
- ▶ Countless number of software used to help designing/verifying systems
Matlab/Simulink, Dymola, etc.
- ▶ These software heavily rely on numerical simulation methods
- ▶ Numerical simulation is a computer-aided method to solve mathematical problems

Questions?

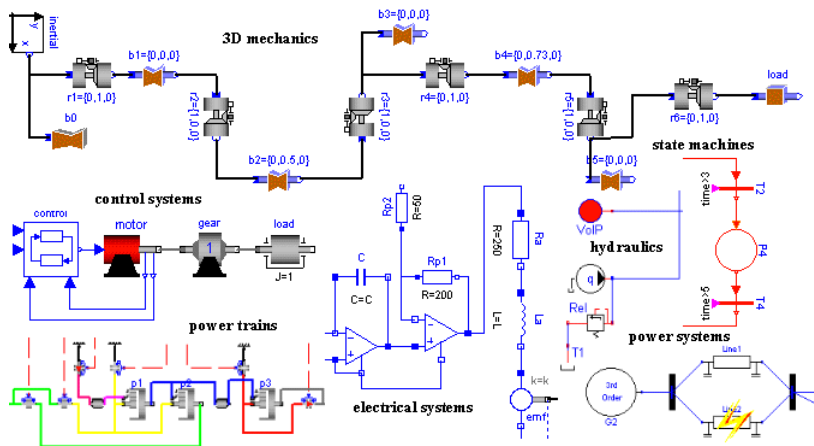
- ▶ What are their limits? (expressiveness, etc.)
- ▶ How do they work? (algorithms, properties, etc.)

Why should we trust these software?

Example Matlab/Simulink



Example Modelica



What do they do?

For each software, some kind of differential equations are solved:

ordinary differential equations (ode)

$$\dot{\mathbf{y}} = f(t, \mathbf{y}(t)) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0$$

differential-algebraic equations (dae)

$$F(t, \mathbf{y}, \dot{\mathbf{y}}) = 0 \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0, \dot{\mathbf{y}}(0) = \mathbf{y}'_0$$

Questions?

- ▶ How these equations are solved?
- ▶ What are the limits of the methods used to solve them?

A few words about the Simulink's solver



All the methodologies are centered around the numerical simulation of the plant and the controller.

Idea

The main principle is to solve the differential equations used to describe mechanical/electrical and controller model.

Remarks on solving mathematical problem on computers

It cannot be done without approximations !

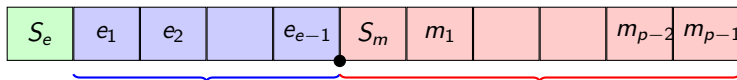
- ▶ Error in the model.
- ▶ Error in the method to solve it (*i.e.*, numerical integration methods)
- ▶ Error in data given by sensors.
- ▶ Error in the computation (*i.e.*, floating-point arithmetic)

The engineer should be aware of such errors.

Floating-point arithmetic

A real number is represented by a floating-point number f such that:

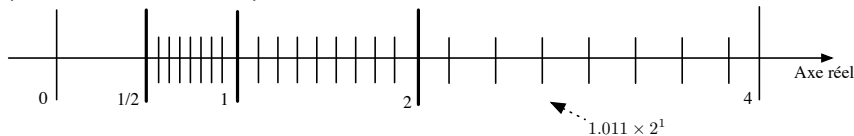
$$r = S \cdot m \cdot 2^e$$



Exponent (e bits)

Mantissa (p bits)

For example, the set of floating-point numbers with a 2 bits exponent (i.e. between -1 and 1) and a 3 bits mantissa is:



Floating-Point Arithmetic

IEEE754-2008 Standard: representation (double precision: 64bits)

This Standard is implemented in almost every computer, it defines:

Encoding: floating-point numbers $f = s.m.2^e$ with

- ▶ s the sign (1 bit)
- ▶ e the exponent (10 bits $\Rightarrow e_{\min} = -1022$ and $e_{\max} = -e_{\min} + 1$)
- ▶ m the significant with p bits and an implicit bit ($p = 53$ bits)
 - if implicit bit is 1 \Rightarrow normal number
 - if implicit bit is 0 \Rightarrow subnormal number

Special values:

- ▶ NaN for undefined results e.g. $\sqrt{-1}$ (absorbing value)
- ▶ $+\infty$ or $-\infty$ for overflows

We also consider three important values which characterize \mathbb{F} :

The relative rounding error unit: $\mu = 2^{-p}$

The smallest positive subnormal floating-point: $\sigma = 2^{e_{\min} - p + 1}$

The biggest positive floating-point: $\Sigma = (2 - 2^{1-p})2^{e_{\max}}$

Floating-Point Arithmetic

IEEE754-2008 Standard: arithmetic



Rounding modes: mainly towards $\pm\infty$, to the nearest (denoted by $\text{fl}()$)
 In particular, generation of overflow and zero are defined by:

$$\forall x \in \mathbb{F}, x > 0, \quad \text{fl}(x) = \begin{cases} +0 & \text{if } 0 < x \leq \sigma/2 \\ +\infty & \text{if } x \geq \Sigma \end{cases}$$

Arithmetic operations: have the correct rounding property.
 More precisely, we have

$$\text{fl}(x \diamond y) = (x \diamond y)(1 + \epsilon_1) + \epsilon_2 \quad \text{with } |\epsilon_1| \leq \mu \text{ and } |\epsilon_2| \leq \frac{1}{2}\sigma$$

with $\diamond \in \{+, -, \times, \div\}$ (still true for $\sqrt{\quad}$)

and

$\epsilon_2 = 0$ if $\diamond \in \{+, -\}$ or $\text{fl}(x \diamond y)$ is in normal range

$\epsilon_1 = 0$ if $\text{fl}(x \diamond y)$ is in subnormal range

Floating-Point Arithmetic

IEEE754-2008 Standard: consequences

Fact

Nevertheless, floating-point numbers only approximate real numbers.

So when using floating-point numbers, we have to deal with:

Poor mathematical properties

- ▶ No associativity (for number of operators > 2)
- ▶ No distributivity
- ▶ No inversion

Numerical instabilities

- ▶ Absorption:
if $|x| \leq \mu|y|$ then $\text{fl}(x + y) = \text{fl}(y)$
- ▶ Catastrophic cancellation

Remark

Usually, results produced by floating-point arithmetic are sufficient for classical algorithms.

Example: absorption

```
#include <stdio.h>

int main ()
{
    float r1 = 10.0;
    float r2 = 0.0000004;
    float r3 = 0.0000003;

    float t1 = r1 + r2 + r3 + r3;
    float t2 = r2 + r3 + r3 + r1;

    printf ("t1 = %f\n t2 = %f\n", t1, t2);

    return 0;
}
```

Results

- ▶ $t1 = 10.0$
- ▶ $t2 = 10.000001$

Example: catastrophic cancellation

```
#include <stdio.h>

int main ()
{
    double eps = 1e-14;
    double x = 10.0 + eps;
    double r = eps / (x - 10.0);

    printf ("r=%f\n", r);

    return 0;
}
```

Results

- ▶ Mathematical result 1.0
- ▶ $r = 0.938250$

Example: inversion

```
#include <stdio.h>

int main ()
{
    double r = sqrt(3.0);
    double m = r * r;
    printf ("m=%f\n", m);

    return 0;
}
```

Results

- ▶ Mathematical result 3.0
- ▶ $m = 2.99999999999999956$

A few words on NaN

Recall that NaN is a special value to represent non valid computation like $\sqrt{-1}$.

Handle such values could be problematic:

- ▶ NaN is a absorbing element for arithmetic operations.

Comparison may be tricky

- ▶ $x \neq \text{NaN}$ is true
- ▶ $\text{NaN} \neq \text{NaN}$ is false

Comparison trap: equality

```

#include <stdio.h>

int main () {
  /* Exemple de F. de Dinechin (modifie pour IA64) */
  int i;
  double ref , index ;

  ref = 169.0f / 170.0f;

  for (i=0; i<250; i++){
    index = i;
    if (ref == (index / (index + 1))) { break; }
  }
  printf ("i=%d\n", i);

  return 0;
}

```

Solutions

$x \geq y \ \&\& \ x \leq y$ or better $\text{fabs}(x-y) \geq \text{eps}$

Conclusion: floating-point arithmetic

Standard of representation of numbers in computer but still we may have the apparent excess of precision is not exempt of numerical problems. In particular, it requires some knowledge of the hardware, for example x86 desktop CPU have 80bits long registers !!
The use of transcendental functions (e.g. \sin, \dots) does not have the correct rounding property!

A few words about the Simulink's solver

The Mathworks methodology is centered around the numerical simulation of the plant and the controller.

Idea

The main principle is to solve the differential equations used to describe mechanical/electrical and controller model.

Remarks on solving mathematical problem on computers

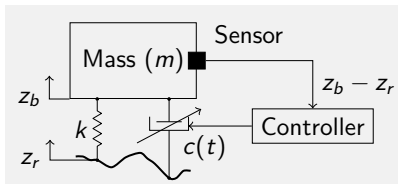
It cannot be done without approximations !

- ▶ Error in the model.
- ▶ Error in the method to solve it.
- ▶ Error in data given by sensors.
- ▶ Error in the computation.

The engineer should be aware of such errors.

A simple example: mathematical models

A semi-active suspension of a quarter-car model



- ▶ $m = 250 \text{ kg}$
- ▶ $k = 20000 \text{ N/m}$
- ▶ $c_{\max} = 16000 \text{ N/m/s}$
- ▶ $c_{\min} = 0$

Mathematical model of the mechanical system

$$\ddot{z}_b = -\frac{1}{m} \left(k(z_b - z_r) + c(t) \dot{z}_b - \dot{z}_r \right) .$$

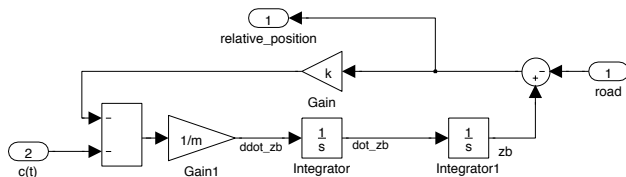
Mathematical model of the controller

$$c(t) = \begin{cases} -c_{\max}(z_b - z_r) & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) < 0 \\ c_{\min} & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) \geq 0 \end{cases} .$$

A simple example: Simulink implementation – 1

Mathematical model of the mechanical system

$$\ddot{z}_b = -\frac{1}{m} \left(k(z_b - z_r) + c(t) \right) .$$



Integrator block

Associated to a first order dynamic system:

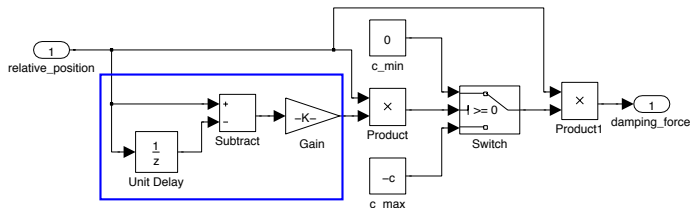
$$\begin{cases} \dot{\mathbf{y}}(t) = \text{input}(t) \\ \text{output}(t) = \mathbf{y}(t) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0 \end{cases}$$

A simple example: Simulink implementation – 2

Mathematical model of the controller

$$c(t) = \begin{cases} -c_{\max}(z_b - z_r) & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) < 0 \\ c_{\min} & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) \geq 0 \end{cases}$$

Implementation: $(\dot{z}_b - \dot{z}_r)$ is given by differentiating the sensor output.



Discrete differentiation at rate 1/40 sec.







Closed-loop system

- ▶ Connect the output of the plant to the input of the controller.
- ▶ Connect the output of the controller to the input of the plant.

Simulink as an equation-based programming language

Input/Output relation

Each block defines the time invariant relation between its input and its output.

Library	Blocks	Representation	Equations
Sources	Input		$l_1 = \text{in1}$
	Constant		$l_1 = c$
Arithmetic	Add		$l_3 = l_1 + l_2$
Signal routing	Switch		$l_4 = \text{if}(p_r(l_2), l_1, l_3)$
Continuous-time	Integrator		$\{l_2 = x; \dot{x} = l_1; x(0) = \text{init}\}$
Discrete-time	Unit Delay		$\{l_2 = d; \bar{d} =_S l_1; d(0) = \text{init}\}$

$\bar{d} =_S l$ stands for at each $t = kS$, $d = l$ else it keeps its previous value.

Simulink as an equation-based programming language



Input/Output relation

Each block defines the time invariant relation between its input and its output.

A core language of equations

$$e ::= r \mid \ell \mid x \mid d \mid e_1 \diamond e_2 \mid e_1 \bowtie e_2 \mid \text{if}(e_1, e_2, e_3) \quad (1)$$

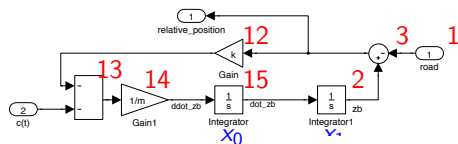
$$eq ::= \ell :=_S e \mid \dot{\ell} := e \mid \bar{d} :=_S e \quad (2)$$

$$p ::= eq \mid eq; p \quad (3)$$

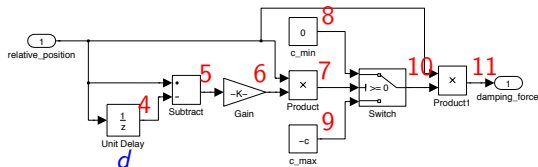
with

- ▶ $r \in \mathbb{R}$, constant values;
- ▶ $\ell \in \mathcal{V}$, variables associated to a **block output**;
- ▶ $x \in \mathcal{V}$, variables associated to **continuous-time states**;
- ▶ $d \in \mathcal{V}$, variables associated to **discrete-time states**;
- ▶ $\diamond \in \{+, -, \times, \div\}$, arithmetic operations;
- ▶ $\bowtie \in \{<, \leq, >, \geq, =, <>\}$, relational operations;
- ▶ S is the set of all the sampling times.

Simulink as an equation-based programming language



In red: evaluation order of blocks



For each block in the evaluation order, a simple translation gives:

$$l_1 = \text{input}$$

$$\dot{x}_1 = l_{15}$$

$$l_2 = x_1$$

$$l_3 = l_2 - l_1$$

$$\bar{d} = l_3$$

$$l_4 = d$$

$$l_5 = l_3 - l_4$$

$$l_6 = 1/40 \times l_5$$

$$l_7 = l_3 \times l_6$$

$$l_8 = 0$$

$$l_9 = -16000$$

$$l_{10} = \text{if } l_7 \geq 0 \text{ then } l_8 \text{ else } l_9$$

$$l_{11} = l_3 \times l_{10}$$

$$l_{12} = 20000 \times l_3$$

$$l_{13} = -l_{12} - l_{11}$$

$$l_{14} = 1/250 \times l_{13}$$

$$\dot{x}_0 = l_{14}$$

$$l_{15} = x_0$$

Simulink as an equation-based programming language



Kinds of equations

A Simulink model is made of four kinds of functions:

- ▶ the output function;
- ▶ the update function of discrete-time states;
- ▶ the update function of continuous-time states;

$$l_1 = \text{input}$$

$$\dot{x}_1 = l_{15}$$

$$l_2 = x_1$$

$$l_3 = l_2 - l_1$$

$$\bar{d} =_S l_3$$

$$l_4 = d$$

$$l_5 = l_3 - l_4$$

$$l_6 = 1/40 \times l_5$$

$$l_7 = l_3 \times l_6$$

$$l_8 = 0$$

$$l_9 = -16000$$

$$l_{10} = \text{if } l_7 \geq 0 \text{ then } l_8 \text{ else } l_9$$

$$l_{11} = l_3 \times l_{10}$$

$$l_{12} = 20000 \times l_3$$

$$l_{13} = -l_{12} - l_{11}$$

$$l_{14} = \frac{1}{250} \times l_{13}$$

$$\dot{x}_0 = l_{14}$$

$$l_{15} = x_0$$

Remark

These functions allow a **state-space representation** of a model.

Simulink as an equation-based programming language



Kinds of equations

A Simulink model is made of four kinds of functions:

- ▶ the output function;
- ▶ the update function of discrete-time states;
- ▶ the update function of continuous-time states;

$$l_1 = \text{input}$$

$$\dot{x}_1 = l_{15}$$

$$l_2 = x_1$$

$$l_3 = l_2 - l_1$$

$$\bar{d} = s l_3$$

$$l_4 = d$$

$$l_5 = l_3 - l_4$$

$$l_6 = 1/40 \times l_5$$

$$l_7 = l_3 \times l_6$$

$$l_8 = 0$$

$$l_9 = -16000$$

$$l_{10} = \text{if } l_7 \geq 0 \text{ then } l_8 \text{ else } l_9$$

$$l_{11} = l_3 \times l_{10}$$

$$l_{12} = 20000 \times l_3$$

$$l_{13} = -l_{12} - l_{11}$$

$$l_{14} = \frac{1}{250} \times l_{13}$$

$$\dot{x}_0 = l_{14}$$

$$l_{15} = x_0$$

Notation

We denote this function by $\mathbf{g}(\mathbf{t}, \mathbf{x}, \mathbf{d})$.

Simulink as an equation-based programming language



Kinds of equations

A Simulink model is made of four kinds of functions:

- ▶ the output function;
- ▶ the update function of discrete-time states;
- ▶ the update function of continuous-time states;

$$l_1 = \text{input}$$

$$\dot{x}_1 = l_{15}$$

$$l_2 = x_1$$

$$l_3 = l_2 - l_1$$

$$\bar{d} =_S l_3$$

$$l_4 = d$$

$$l_5 = l_3 - l_4$$

$$l_6 = 1/40 \times l_5$$

$$l_7 = l_3 \times l_6$$

$$l_8 = 0$$

$$l_9 = -16000$$

$$l_{10} = \text{if } l_7 \geq 0 \text{ then } l_8 \text{ else } l_9$$

$$l_{11} = l_3 \times l_{10}$$

$$l_{12} = 20000 \times l_3$$

$$l_{13} = -l_{12} - l_{11}$$

$$l_{14} = \frac{1}{250} \times l_{13}$$

$$\dot{x}_0 = l_{14}$$

$$l_{15} = x_0$$

Notation

We denote this function by $\mathbf{f}_d(\mathbf{t}, \mathbf{x}, \mathbf{d}) = g(t, x, d) |_{\bar{d}=_S \ell}$.

Simulink as an equation-based programming language



Kinds of equations

A Simulink model is made of four kinds of functions:

- ▶ the output function; (2 versions: major g and **minor** \tilde{g})
- ▶ the update function of discrete-time states;
- ▶ the update function of continuous-time states;

$$l_1 = \text{input}$$

$$\dot{x}_1 = l_{15}$$

$$l_2 = x_1$$

$$l_3 = l_2 - l_1$$

$$\bar{d} =_S l_3$$

$$l_4 = d$$

$$l_5 = l_3 - l_4$$

$$l_6 = 1/40 \times l_5$$

$$l_7 = l_3 \times l_6$$

$$l_8 = 0$$

$$l_9 = -16000$$

$$l_{10} = \text{if } l_7 \geq 0 \text{ then } l_8 \text{ else } l_9$$

$$l_{11} = l_3 \times l_{10}$$

$$l_{12} = 20000 \times l_3$$

$$l_{13} = -l_{12} - l_{11}$$

$$l_{14} = \frac{1}{250} \times l_{13}$$

$$\dot{x}_0 = l_{14}$$

$$l_{15} = x_0$$

Notation

We denote this function by $\mathbf{f}_x(\mathbf{t}, \mathbf{x}, \mathbf{d}) = \tilde{g}(t, x, d) \big|_{\bar{d}=_S \ell}$.

Simulink as an equation-based programming language



Kinds of equations

A Simulink model is made of four kinds of functions:

- ▶ the output function;
- ▶ the update function of discrete-time states;
- ▶ the update function of continuous-time states;

Hidden equations: the 5th kind

Some blocks are associated to equations to detect **zero-crossing events** (only with variable step solvers).

Notation

We denote a zero-crossing equation $f_z(t, x, d)$.

Overview of numerical simulation

Goal: computing the **temporal evolution** of the system.
The steps of the Simulink's simulation engine

Input: $\mathbf{x}_0, \mathbf{d}_0, t_0, h_0$;

$n = 0$;

loop until $t_n \geq t_{\text{end}}$

evaluate $g(t_n, \mathbf{x}_n, \mathbf{d}_n)$;

update $\mathbf{d}' = f_d(t_n, \mathbf{x}_n, \mathbf{d}_n)$;

solve $\dot{\mathbf{x}}(t) = f_x(t, \mathbf{x}(t), \mathbf{d}_n)$ over interval $[t_n, t_n + h_n]$ to get $\mathbf{x}(t_n + h_n)$;

find_zero_crossing;

compute h_{n+1} ;

compute t_{n+1} ;

$\mathbf{d}_{n+1} = \mathbf{d}'$; $\mathbf{x}_{n+1} = \mathbf{x}(t_n + h_n)$; $n = n + 1$;

end loop

Remarks

- ▶ **Major steps:** evaluation of g produces the simulation results at t_n .
- ▶ **Minor steps:** evaluations of \tilde{g} are used as intermediate computations between t_n and $t_n + h_n$.

Overview of numerical simulation

The **temporal evolution** of the system depends on a set of parameters

Parameters (a sample)

A Simulink model is described by a set of parameters:

- ▶ $t_0 = 0$ **start time** of the simulation;
- ▶ $t_{\text{end}} = 10$ **stop time** of the simulation;
- ▶ **numerical integration methods** with an **absolute tolerance** ($\text{atol} = 10^{-6}$), a **relative tolerance** ($\text{rtol} = 10^{-3}$);
- ▶ **minimal** ($\text{hmin} = 0.2$) and **maximal** ($\text{hmax} = 5$) integration step-size;
- ▶ **zero-crossing method**: adaptive or non adaptive;
- ▶ zero-crossing tolerance ($\text{zctol} = 10 \times 128 \times \text{eps}$).

Consequence

There are several semantics of Simulink, *i.e.*, several possible output for one model.

Differential equations

Many classes

- ▶ Ordinary Differential Equations (ODE)

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

- ▶ Differential-Algebraic equations (DAE)

$$\begin{aligned}\dot{\mathbf{y}}(t) &= \mathbf{f}(t, \mathbf{y}(t), \mathbf{x}(t)) \\ 0 &= \mathbf{g}(t, \mathbf{y}(t), \mathbf{x}(t))\end{aligned}$$

- ▶ Delay Differential Equations (DDE)

$$\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t), \mathbf{y}(t - \tau))$$

- ▶ and others: partial differential equations, etc.

Solving ordinary differential equations

We consider the problem:

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

with

- ▶ n is the dimension of the problem
- ▶ $\dot{\mathbf{y}} = \frac{d\mathbf{y}}{dt}$ is the derivative of \mathbf{y} w.r.t. time t .
- ▶ $f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$

Example

$$\begin{cases} \dot{y}_1 = -y_2 \\ \dot{y}_2 = y_1 \end{cases}$$

Remark: it may be have an infinite number of solutions.

Initial value problem (IVP)

Definition of IVP

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0$$

The IVP is **autonomous** if f does not explicitly depend on t : $\dot{\mathbf{y}} = f(\mathbf{y})$

- ▶ High order vs first order

$$\ddot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \dot{\mathbf{y}}) \Leftrightarrow \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ \mathbf{f}(y_1, y_2) \end{pmatrix} \quad \text{with} \quad y_1 = y \quad \text{and} \quad y_2 = \dot{y} .$$

- ▶ Non-autonomous vs autonomous

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \Leftrightarrow \dot{\mathbf{z}} = \begin{pmatrix} \dot{t} \\ \dot{\mathbf{y}} \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{f}(t, \mathbf{y}) \end{pmatrix} = \mathbf{g}(\mathbf{z}) .$$

Initial value problem (IVP)

Definition of IVP

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0$$

IVP has a unique solution if:

- ▶ f is continuous with respect to time t
- ▶ f is Lipschitz with respect to \mathbf{y} that is:

$$\forall t, \forall \mathbf{y}_1, \mathbf{y}_2 \in \mathbb{R}^n, \exists L > 0, \quad \| f(t, \mathbf{y}_1) - f(t, \mathbf{y}_2) \| \leq L \| \mathbf{y}_1 - \mathbf{y}_2 \|$$

Remark it is still true for piece-wise Lipschitz functions.

Remark the uniqueness is lost if continuity is only considered.

Ordinary differential equations

We focus on the continuous-time evolution function:

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0 . \quad (1)$$

Usually, solution $\mathbf{y}(t; \mathbf{y}_0)$ of Equation (1) is studied using numerical integration scheme inducing a recurrence of the form:

$$\mathbf{y}_{k+1} = \mathcal{F}(t_k, h_k, \mathbf{y}_{k+1}, \mathbf{y}_k, \mathbf{y}_{k-1}, \dots, \mathbf{y}_{k-p})$$

such that $\mathbf{y}_{k-i} \approx \mathbf{y}(t_{k-i}; \mathbf{y}_0) \quad i = 0, \dots, p . \quad (2)$

It exists a huge amount of methods whose main features are:

- ▶ explicit (\mathbf{y}_{k+1} is not considered) or implicit method;
- ▶ single-step ($p = 0$) or multi-step method.
- ▶ with fixed (h does not depend on k) or variable step-size;

Goal of numerical integration

Recall, we consider IVP (initial value problem):

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0 . \quad (3)$$

This problem admits a unique solution $\mathbf{y}(t; \mathbf{y}_0)$ on \mathbb{R} , if $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n$ is continuous in t and Lipschitz in \mathbf{y} that is:

$$\forall t, \forall \mathbf{y}_1, \mathbf{y}_2 \in \mathbb{R}^n, \exists L > 0, \quad \| f(t, \mathbf{y}_1) - f(t, \mathbf{y}_2) \| \leq L \| \mathbf{y}_1 - \mathbf{y}_2 \| .$$

Goal

- ▶ Compute a sequence of time instants $t_0 \leq t_1 \leq \dots \leq t_n$
- ▶ Compute a sequence of values $\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n$ such that

$$\forall i \in [1, n], \quad \mathbf{y}_i \approx \mathbf{y}(t_i; \mathbf{y}_{i-1}) .$$

Example: Euler's method

Consider a simple IVP:

$$\dot{x} = -\frac{x^3}{2} \quad \text{with} \quad x(0) = 1 .$$

The exact solution is $x(t) = \frac{1}{\sqrt{1+t}}$.

We consider two fixed-step methods (Euler and Heun) that is they compute the sequence of time instants such that $t_{i+1} = t_i + h$.

- ▶ The Euler's method computes the sequence of values

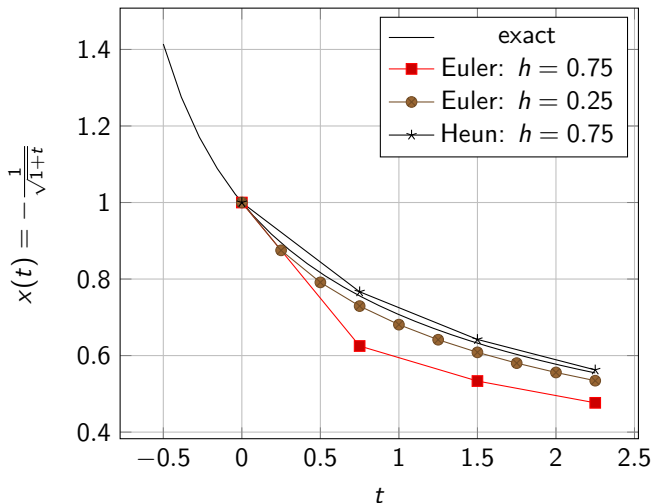
$$x_{i+1} = x_i + h \times -\frac{x_i^3}{2}$$

- ▶ The Heun's method computes the sequence of values:

$$k_1 = x_i + h \times -\frac{x_i^3}{2}$$

$$x_{i+1} = x_i + \frac{h}{2} \times \left(\left(-\frac{x_i^3}{2} \right) + \left(-\frac{k_1^3}{2} \right) \right)$$

Example: Euler's method



Remark: precision vs performance in the application of Euler's method.

Summary on ODEs



There is a huge amount of work on the study of IVP for ODE
A lot of numerical methods exists, *e.g.*, Runge-Kutta, Adams-Bashworth,
...

In the next lectures, we will review

- ▶ the main numerical algorithms to solve IVP for ODE
- ▶ the main features of such methods, *e.g.*, stability, convergence, etc.