On the Computational Complexity of Solving Ordinary Differential Equations

Olivier Bournez
LIX, Ecole Polytechnique, France

RP 2018
Marseille

24 September 2018

Our question

- Computational hardness of solving an **Initial Value Problem** (also called **Cauchy's problem**) of the form:

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned}$$

Our question

- Computational hardness of solving an **Initial Value Problem** (also called **Cauchy's problem**) of the form:

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned}$$

- Well, this require to discuss deeper questions than expected.
  - What means "solving"?

Our question

- Computational hardness of solving an **Initial Value Problem** (also called **Cauchy's problem**) of the form:

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned}$$

- Well, this require to discuss deeper questions than expected.
  - What means "solving"?
    - Answer: solving with a computer

Our question

- Computational hardness of solving an **Initial Value Problem** (also called **Cauchy's problem**) of the form:

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned}$$

- Well, this require to discuss deeper questions than expected.
  - What means "solving"?
    - Answer: solving with a computer
  - but what means computer?

Our question

- Computational hardness of solving an **Initial Value Problem** (also called **Cauchy's problem**) of the form:

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned}$$

- Well, this require to discuss deeper questions than expected.
  - What means "solving"?
    - Answer: solving with a computer
  - but what means computer?
  - how complexity is measured?

Menu

What is a computer ?

Laptop

Supercomputer

Servers

The highest-selling single computer model of all time

source: Guinness World Records

What is a computer ?

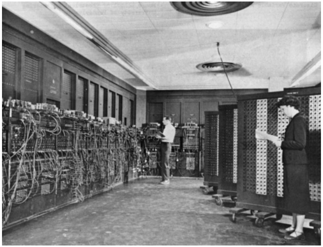
Laptop

Supercomputer

Servers

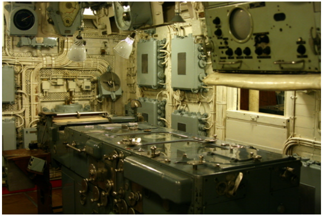
Commodore 64

What is a computer ?



ENIAC

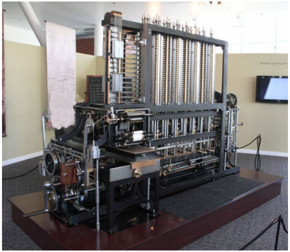
Kelvin's Tide Predicter



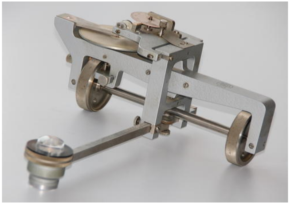
Admiralty Fire Control Table

Differential Analyzer

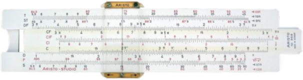
What is a computer ?



Difference Engine



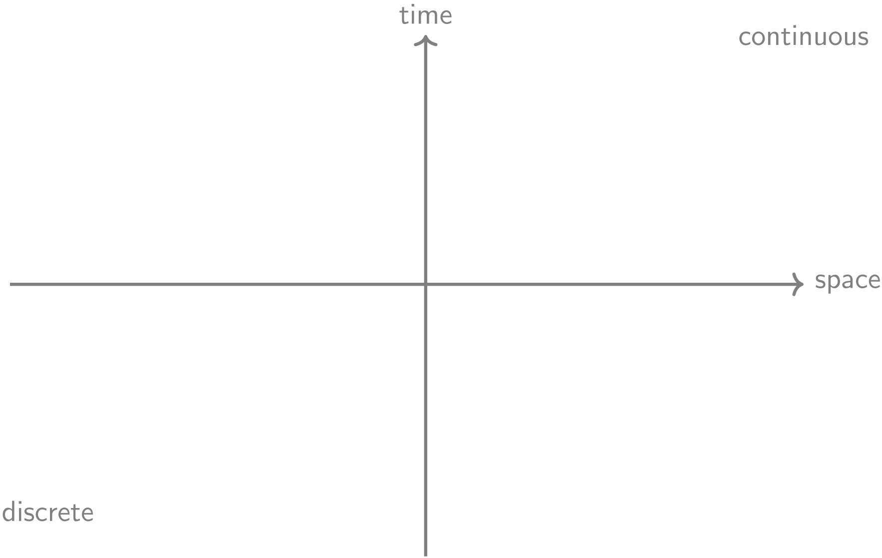
Linear Planimeter



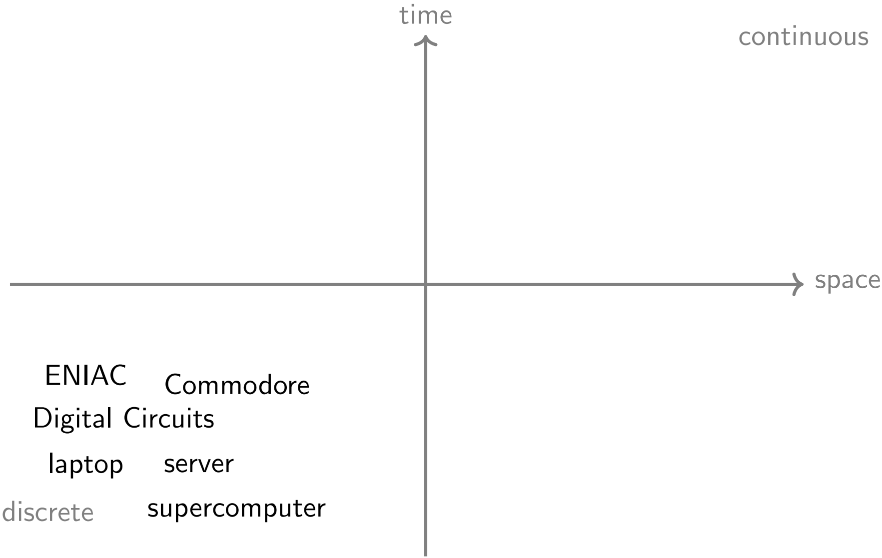
Slide Rule

Antikythera mechanism

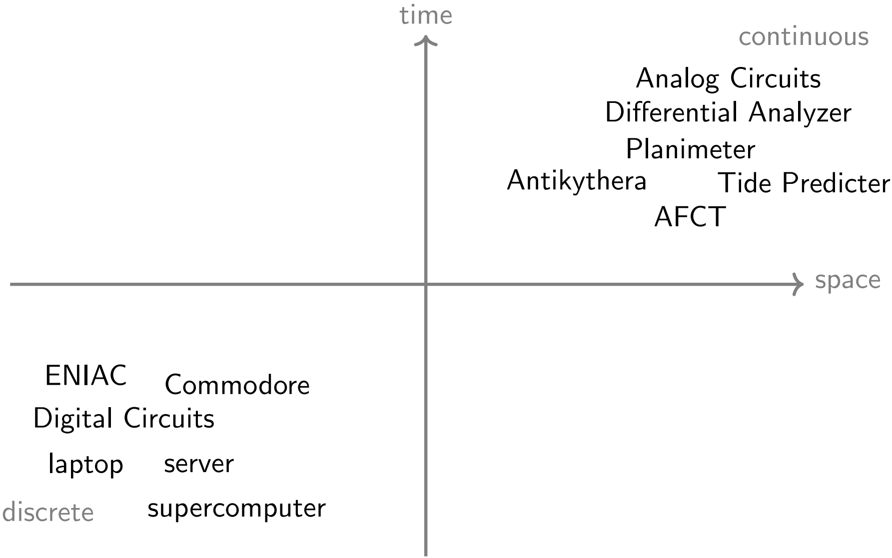
A first classification



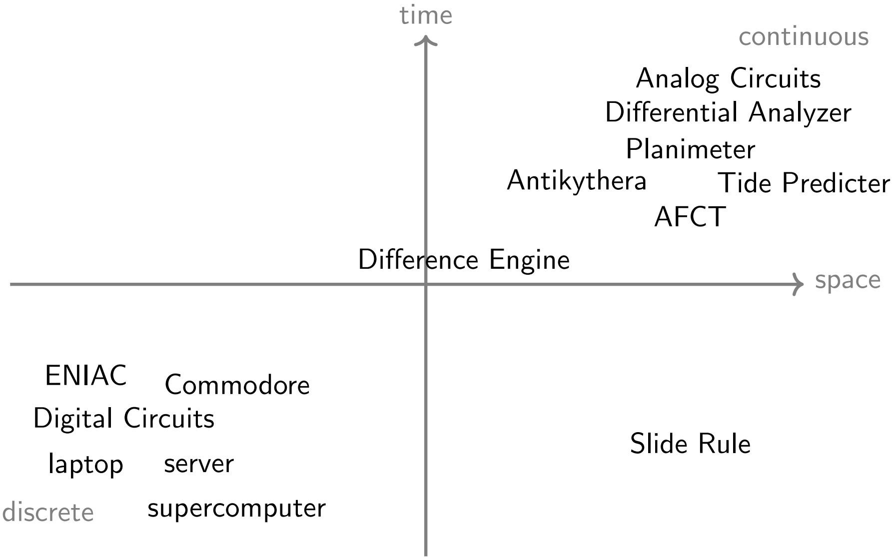
A first classification



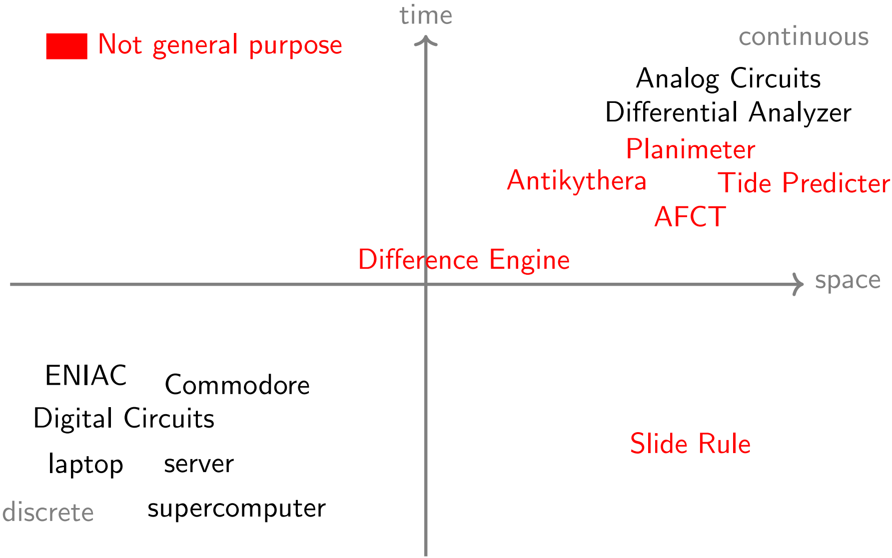
A first classification



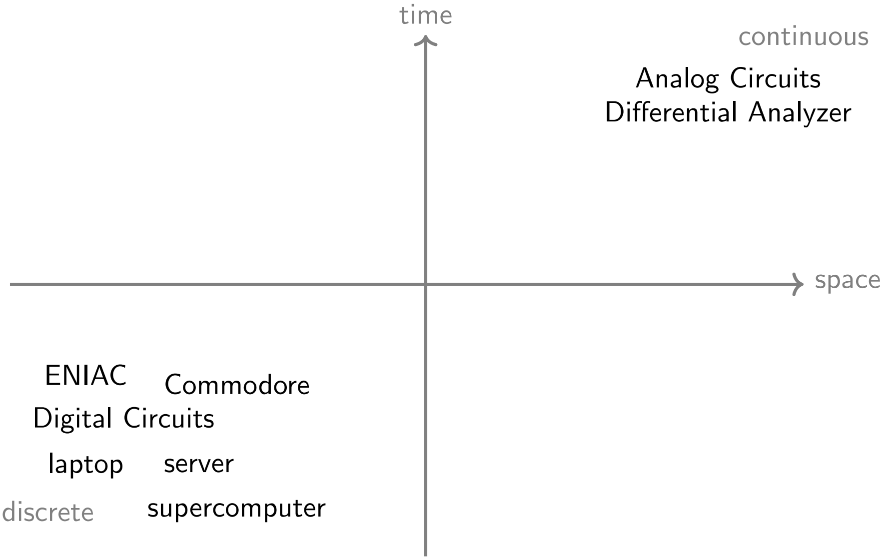
A first classification



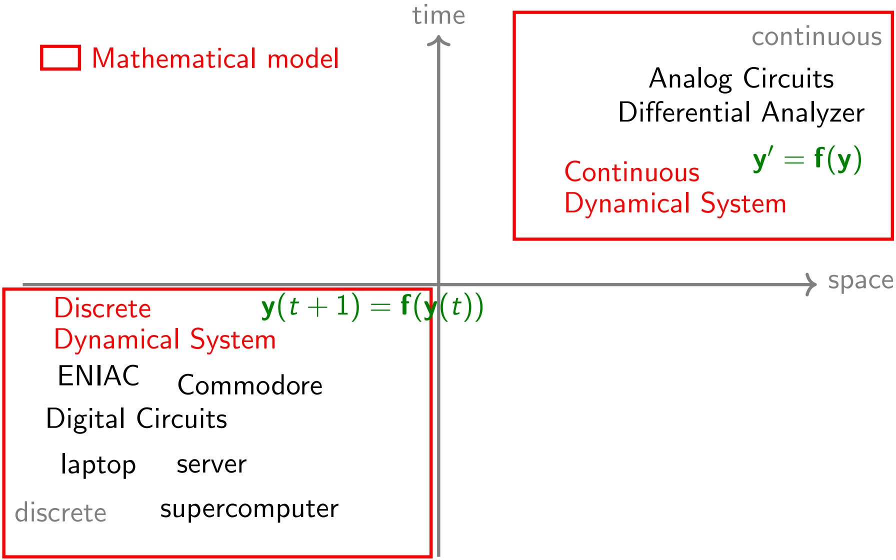
A first classification



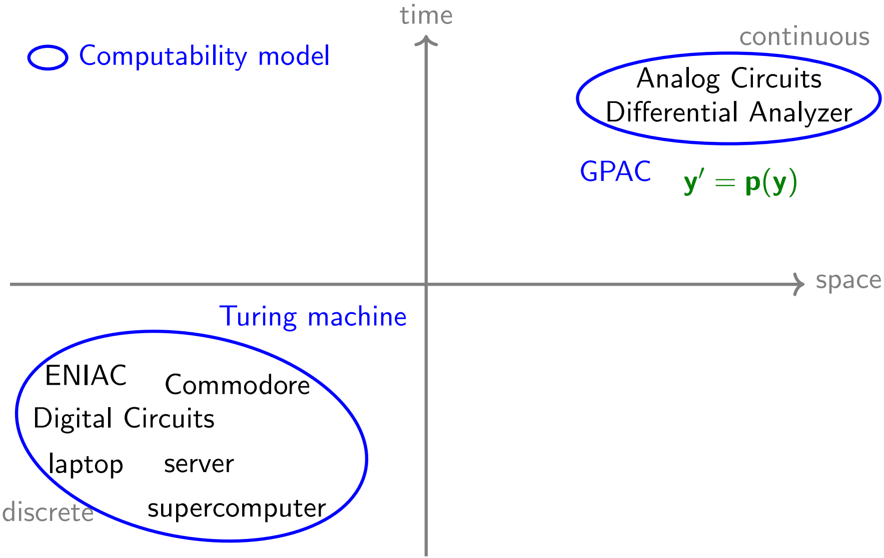
A first classification



A first classification



A first classification



Our actual motivation

- Understand how analog models compare to classical digital models of computation.
  - At **computability** level
  - At **complexity** level.
- Continuous time analog models correspond to various classes of ordinary differential equations.
- Discussing hardness of solving IVP according to various classes of dynamics is basically discussing the **computational power** of various classes of analog models.

Sub-menu

Shannon's General Purpose Analog Computer

- The **GPAC** is a mathematical abstraction from Claude Shannon (1941) of the **Differential Analyzers**.

- [Graça Costa 03]: This corresponds to **polynomial Ordinary Differential Equations** (pODEs), i.e.

$$\begin{aligned} \mathbf{y}' &= \mathbf{p}(t, \mathbf{y}) \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned}$$

where

- $\mathbf{p}$ is a (vector of) polynomials.

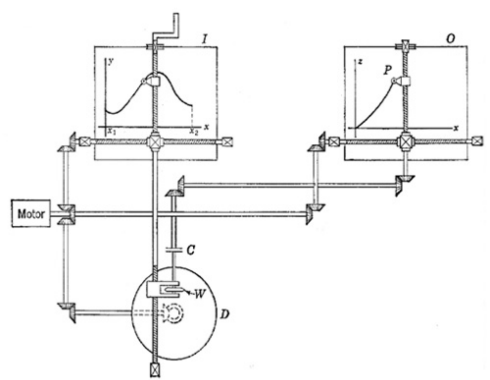
A machine from 20th Century: Differential analyzers

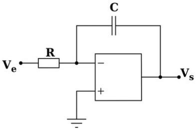
Vannevar Bush's 1938 mechanical Differential Analyser

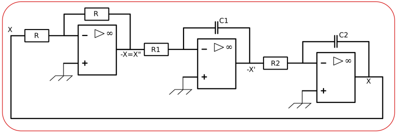
- Underlying principles: Lord Kelvin 1876.
- First ever built: V. Bush 1931 at MIT.
- Applications: from gunfire control up to aircraft design
- Intensively used during U.S. war effort.
- Electronic versions from late 40s, used until 70s

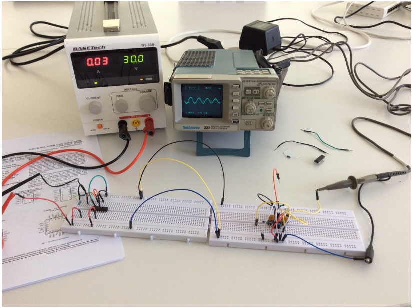
A machine from 21th Century: Analog Paradigm Model-1

- http://analogparadigm.com
- Fully modular
- Basic version.
  - 4 integrators, 8 constants, 8 adders, 8 multipliers.
  - 14 kgs.


I
O
y
x
x_1
x_2
z
P
Motor
C
W
D


C
R
V_e
V_s
−
+

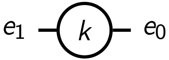

X
R
R
-X=X"
R1
C1
-X'
R2
C2
X


BASETech
0.03
30.0

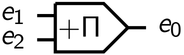
The General Purpose Analog Computer

Shannon's 41 presentation:

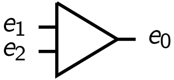
- Basic units:



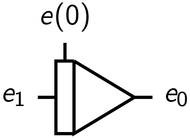
constant: $e_0 = ke_1$



product: $e_0 = e_1 e_2$



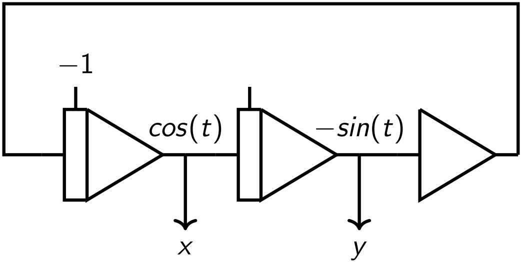
summer: $e_0 = -(e_1 + e_2)$



integrator:
$e_0 = -\int_0^t (e_1(u)du + e(0))$

- (Feedback connections are allowed).
- A function is GPAC-generated if it corresponds to the output of some unit of a GPAC.

Cosinus and sinus: $x = \cos(t)$, $y = \sin(t)$



$$\begin{cases} x'(t) = -y(t) \\ y'(t) = x(t) \\ x(0) = 1 \\ y(0) = 0 \end{cases} \Rightarrow \begin{cases} x(t) = \cos(t) \\ y(t) = \sin(t) \end{cases}$$

Sub-menu

What is a computer?

- The GPAC
- Programming with the GPAC

Programming Exercice

How to transform initial-value problem

$$\begin{cases} y_1' & = & \sin^2 y_2 \\ y_2' & = & y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) & = & 0 \\ y_2(0) & = & 0 \end{cases}$$

Programming Exercice

How to transform initial-value problem

$$\begin{cases} y_1' &= \sin^2 y_2 \\ y_2' &= y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) &= 0 \\ y_2(0) &= 0 \end{cases}$$

into a polynomial initial value problem

Programming Exercice

How to transform initial-value problem

$$\begin{cases} y_1' & = & \sin^2 y_2 \\ y_2' & = & y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) & = & 0 \\ y_2(0) & = & 0 \end{cases}$$

into a polynomial initial value problem

$$\begin{cases} y_1' & = & y_3^2 \\ & & \\ & & \\ & & \end{cases} \qquad \begin{cases} y_1(0) & = & 0 \\ & & \\ & & \\ & & \end{cases}$$

considering $y_3 = \sin y_2$

Programming Exercice

How to transform initial-value problem

$$\begin{cases} y_1' &= \sin^2 y_2 \\ y_2' &= y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) &= 0 \\ y_2(0) &= 0 \end{cases}$$

into a polynomial initial value problem

$$\begin{cases} y_1' &= y_3^2 \\ y_2' &= y_1 y_4 - y_5 \\ \\ \\ \\ \end{cases} \qquad \begin{cases} y_1(0) &= 0 \\ y_2(0) &= 0 \\ \\ \\ \\ \end{cases}$$

considering $y_3 = \sin y_2, y_4 = \cos y_2 \ , y_5 = e^{e^{y_1}+t}$

How to transform initial-value problem

$$\begin{cases} y_1' = \sin^2 y_2 \\ y_2' = y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \end{cases}$$

into a polynomial initial value problem

$$\begin{cases} y_1' = y_3^2 \\ y_2' = y_1 y_4 - y_5 \\ y_3' = y_4(y_1 y_4 - y_5) \\ \\ \\ \end{cases} \qquad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \\ y_3(0) = 0 \\ \\ \\ \end{cases}$$

considering $y_3 = \sin y_2, y_4 = \cos y_2\ , y_5 = e^{e^{y_1}+t}$

Programming Exercice

How to transform initial-value problem

$$\begin{cases} y_1' = \sin^2 y_2 \\ y_2' = y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \end{cases}$$

into a polynomial initial value problem

$$\begin{cases} y_1' = y_3^2 \\ y_2' = y_1 y_4 - y_5 \\ y_3' = y_4(y_1 y_4 - y_5) \\ y_4' = -y_3(y_1 y_4 - y_5) \\ \\ \end{cases} \qquad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \\ y_3(0) = 0 \\ y_4(0) = 1 \\ \\ \end{cases}$$

considering $y_3 = \sin y_2, y_4 = \cos y_2 \ , y_5 = e^{e^{y_1}+t}$

Programming Exercice

How to transform initial-value problem

$$\begin{cases} y_1' &= \sin^2 y_2 \\ y_2' &= y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) &= 0 \\ y_2(0) &= 0 \end{cases}$$

into a polynomial initial value problem

$$\begin{cases} y_1' &= y_3^2 \\ y_2' &= y_1 y_4 - y_5 \\ y_3' &= y_4(y_1 y_4 - y_5) \\ y_4' &= -y_3(y_1 y_4 - y_5) \\ y_5' &= y_5(y_6 y_3^2 + 1) \end{cases} \qquad \begin{cases} y_1(0) &= 0 \\ y_2(0) &= 0 \\ y_3(0) &= 0 \\ y_4(0) &= 1 \\ y_5(0) &= e \end{cases}$$

considering $y_3 = \sin y_2, y_4 = \cos y_2\ , y_5 = e^{e^{y_1}+t}, y_6 = e^{y_1}$

Programming Exercice

How to transform initial-value problem

$$\begin{cases} y_1' &= \sin^2 y_2 \\ y_2' &= y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases} \qquad \begin{cases} y_1(0) &= 0 \\ y_2(0) &= 0 \end{cases}$$

into a polynomial initial value problem

$$\begin{cases} y_1' &= y_3^2 \\ y_2' &= y_1 y_4 - y_5 \\ y_3' &= y_4(y_1 y_4 - y_5) \\ y_4' &= -y_3(y_1 y_4 - y_5) \\ y_5' &= y_5(y_6 y_3^2 + 1) \\ y_6' &= y_6 y_3^2 \end{cases} \qquad \begin{cases} y_1(0) &= 0 \\ y_2(0) &= 0 \\ y_3(0) &= 0 \\ y_4(0) &= 1 \\ y_5(0) &= e \\ y_6(0) &= 1 \end{cases}$$

considering $y_3 = \sin y_2$,$y_4 = \cos y_2$,$y_5 = e^{e^{y_1}+t}$,$y_6 = e^{y_1}$

Closure Properties

- The class of generated functions include all (analytic) **common functions**.
- It is stable by many operations:
  - if f and g can be generated, then $f+g$, $f-g$, fg, $\frac{1}{f}$, $f \circ g$ can be generated.
- It is stable by ODE solving:
  - if f can be generated, and y satisfies $y' = f(y)$ then y can be generated.
- A generated function must be analytic.
  - **Famous** analytic **non-generable functions:** [Shannon 41]
    - Euler's Gamma function $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$ [Hölder 1887]
    - Riemann's Zeta function $\zeta(x) = \sum_{k=0}^\infty \frac{1}{k^x}$ [Hilbert].
- The set of pODE computable constants (of the form $f(1)$) is a field.

Menu

Our question

- Computational hardness of solving an **Initial Value Problem** (also called **Cauchy's problem**) of the form:

$$\begin{array}{rcl} \mathbf{y}' & = & \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t_0) & = & \mathbf{y}_0 \end{array}$$

- Various classes of functions f:

$$\begin{array}{ll} & \mathbf{f} \text{ is continuous} \\ \Leftarrow & \mathbf{f} \text{ is (locally) Lipchitz}^1 \\ \Leftarrow & \mathbf{f} \text{ is } \mathcal{C}^k,\ k \geq 1. \\ \Leftarrow & \mathbf{f} \text{ is analytic}^2 \\ \Leftarrow & \mathbf{f} \text{ is polynomial} \end{array}$$

[1] $||\mathbf{f}(\mathbf{y}) - \mathbf{f}(\mathbf{y}')|| \leq L||\mathbf{y} - \mathbf{y}'||$ for some L.
[2] $\mathbf{f}$ is equal to its Taylor's expansion in every point.

Maths: Back to school

$$\begin{cases} \mathbf{y}' &= \mathbf{f}(\mathbf{y}(t)) \\ \mathbf{y}(0) &= \mathbf{x} \end{cases} \qquad (1)$$

- Famous theorems:

Peano-Ascoli	if	$\mathbf{f}$ is continuous	then	existence of solutions
Cauchy - Lipschitz Picard - Lindelöf	if	$\mathbf{f}$ is Lipchitz[3]		+ unicity of solutions
		$\mathbf{f}$ is $\mathcal{C}^k$, $k \geq 1$.		
Cauchy-Kowalevski	if	$\mathbf{f}$ is analytic[4]		+ solutions are analytic
		$\mathbf{f}$ is polynomial		

- Other facts:
  - No restriction in considering ODEs in this form $\mathbf{y}' = \mathbf{f}(\mathbf{y}(t))$.

[3] $||\mathbf{f}(\mathbf{y}) - \mathbf{f}(\mathbf{y}')|| \leq L||\mathbf{y} - \mathbf{y}'||$ for some L.
[4] $\mathbf{f}$ is equal to its Taylor's expansion in every point.

Important preliminary

- Discussing the hardness of the problem
  - over $t \in [0, 1]$ (or any compact domain)

    is really different from
  - over $t \in \mathbb{R}$.

Classical numerical methods

- Euler's method:

$$
\begin{aligned}
y_{i+1} &= y_i + f(t_i, y_i)h \qquad (2)\\
t_{i+1} &= t_i + h \qquad (3)
\end{aligned}
$$

- Runge Kutta's 4th oder method:

$$
\begin{aligned}
y_{i+1} &= y_i + 1/6(k_1 + 2k_2 + 2k_3 + k_4) \qquad (4)\\
k_1 &= hf(t_i, y_i) \qquad (5)\\
k_2 &= hf(t_i + h/2, y_i + k_1/2) \qquad (6)\\
k_3 &= hf(t_i + h/2, y_i + k_2/2) \qquad (7)\\
k_4 &= hf(t_i + h, y_i + k_3) \qquad (8)
\end{aligned}
$$

- <your prefered method >
- . . .

Work well and are efficient **over a compact domain** $t \in [0, 1]$, assuming f **Lipschitz**

- But are **NOT polynomial** in t when $t \in \mathbb{R}$.

Menu

Sub-menu

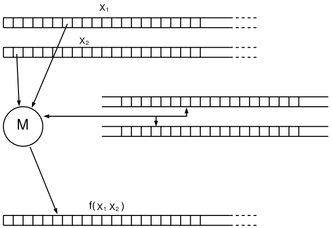
The Starting Point of Recursive Analysis[5]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by a finite means.

Cited from A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem". Proc. London Math. Soc. 42 (1936) 230-265.

[5] This part is mostly borrowed from Vasko Brattka's Tutorial, CIE 2005

Computable Functions: informal presentation for $f : \mathbb{R}^2 \to \mathbb{R}$



A tape represents a real number

Each real number x is represented via an infinite sequence $(x_n)_n \in \mathbb{Q}$,

$$||x_n - x|| \leq 2^{-n}.$$

M behaves like a Turing Machine

Read-only one-way input tapes
Write-only one-way output tape.
M outputs a representation of $f(x_1, x_2)$ from representations of x_1, x_2.

Sub-menu

Computable analysis point of view: impossible in the general case.

Theorem (Pour-El Richards 79)

There exists some **computable** $f : [0,1] \times [-1,1] \to \mathbb{R}$ *such that ordinary differential equation* $y' = f(t,y)$, *has no computable solution over any closed domain.*

Computable analysis point of view: impossible in the general case.

Theorem (Pour-El Richards 79)

There exists some **computable** $f : [0,1] \times [-1,1] \to \mathbb{R}$ *such that ordinary differential equation* $y' = f(t, y)$, *has no computable solution over any closed domain.*

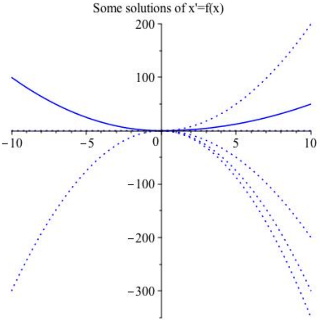
- However:
  - Imposing unicity and existence of solutions leads to computability **[Ruohonen 96]**.
  - If f is continuous and y is the unique solution of $x' = f(t,x)$, $x(t_0) = x_0$, then the operator which maps (f, t_0, x_0) to y is computable **[Graça Collins 09]**

Sub-menu

Non unicity: a classical counter-example

Solutions over $\mathbb{R}$ of f **continuous** such that:

$$\begin{cases} y' = f(t,y) \\ y(0) = 0 \end{cases} \quad \text{with} \quad \begin{cases} f(t,y) = \frac{2y}{t} \text{ for } t \neq 0 \\ f(0,y) = 0 \end{cases} \quad :$$

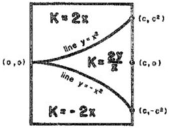


all functions y_{C_1,C_2} with $C_1, C_2 \in \mathbb{R}$, where

$$y_{C_1,C_2}(t) = \begin{cases} C_1 t^2 & \text{if } t < 0 \\ C_2 t^2 & \text{if } t \geq 0 \end{cases}$$

Proof IDEA of uncomputability result: Dispersers/Collectors

- **A Disperser**: consider $K(x, y)$ given by



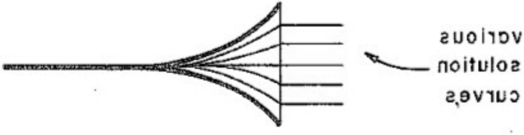
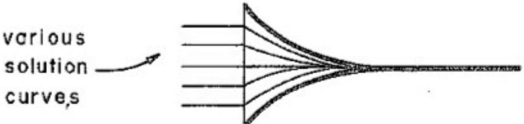
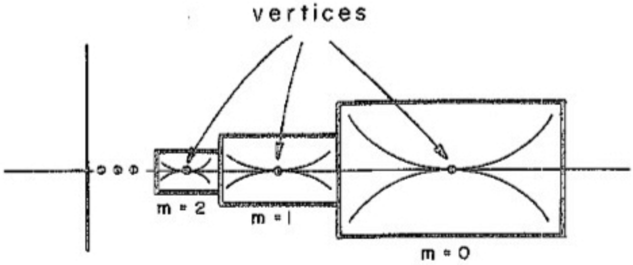


Fig. 1b. A typical collector.

- **A Collector**:



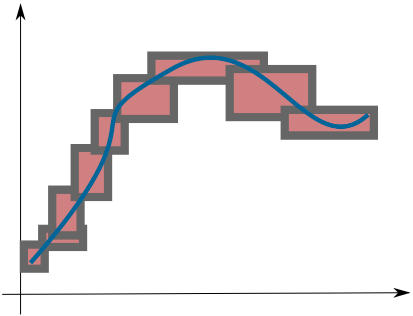
Proof idea of uncomputability result: How the considered function looks like?



- A sequence of "boxes" that become progressively smaller as m increases, and the vertexes converge to the origin.
  - Each box is made of a "collector" and of a "disperser".
  - This provides computability of the function.
- A small "pulse" is placed at the vertex of some of the box:
  - For the mth box, this pulse is positive, negative, or zero, depending on whether $m \in A$, $m \in B$, or $m \notin A \cup B$
  - where (A, B) is a fixed recursively inseparable pair of sets.
- By reading $x = x_m$ at the aperture of disperser m within an error less than half the size of the aperture, one knows whether $x \in A$ or $x \in B$.

Proof idea of computability: Why unicity suffices to get computability

- Exhaustive algorithm:
  - Generate all possible (partial) coverings of the state space
  - Coverings of arbitrary small diameters exist.
  - By a reasoning (similar to Peano-Ascoli's theorem), they must contain at least a solution.
  - So just keep testing coverings until you find an appropriate one



- Named the **"ten thousand monkeys approach"** by its authors ...

Sub-menu

Computational complexity over a compact domain

- Summary:

Assumptions on f	Upper bound	Lower bound
ODE with unique solution	Computable	Arbitrary high complexi
Lipschitz ODE f	PSPACE	PSPACE
f is of class $\mathcal{C}^1$	PSPACE	PSPACE
f is of class $\mathcal{C}^k$, $k > 1$	PSPACE	CH
f is analytic	P	P

- Results due to **[Miller 1970]**, **[Ko 1983]**, **[Müller, 1987]** **[Kawamura, 2010]** and **[Kawamura et al., 2014]**

Over non-compact domains?

- The previous results are only valid in compact sets.
- Actually, this is **provably impossible** to do it in polynomial time over non-compact sets: consider

$$\left\{\begin{array}{lcl} y_1' & = & y_1 \\ y_2' & = & y_1 y_2 \\ y_3' & = & y_2 y_3 \\ \vdots & \vdots & \vdots \\ y_n' & = & y_{n-1} y_n \end{array}\right.$$

Over non-compact domains?

- The previous results are only valid in compact sets.
- Actually, this is **provably impossible** to do it in polynomial time over non-compact sets: consider
  - $e^{e^{e^{\cdots^{e^t}}}}$ is solution of $\left\{\begin{array}{lcl} y_1' & = & y_1 \\ y_2' & = & y_1 y_2 \\ y_3' & = & y_2 y_3 \\ \vdots & \vdots & \vdots \\ y_n' & = & y_{n-1} y_n \end{array}\right.$

Over non-compact domains?

- The previous results are only valid in compact sets.
- Actually, this is **provably impossible** to do it in polynomial time over non-compact sets: consider
  - $e^{e^{e^{\cdots^{e^t}}}}$ is solution of $\begin{cases} y_1' &= y_1 \\ y_2' &= y_1 y_2 \\ y_3' &= y_2 y_3 \\ \vdots & \vdots \quad \vdots \\ y_n' &= y_{n-1} y_n \end{cases}$
  - This **cannot** be computed in a time polynomial over $\mathbb{R}$,
    - since just writing this value in binary cannot be done in polynomial time.

Menu

Parameterized complexity

- This is then natural to use **parameterized complexity**
  - i.e. complexity is measured against one or more extra parameters.

Example:

Proposition

If (α, β) is the maximal interval of existence of the solution y of an ODE $y' = f(t, y)$ and $\beta < +\infty$ then $y(t)$ is unbounded as $t \to \beta$

Some natural examples:

- **Growth of functions** or of their derivatives.
- **Length of the solution curve** $\mathbf{y}$ between $\mathbf{y}(0) := \mathbf{y}(0; 0, y(0))$ and $\mathbf{y}(t) := \mathbf{y}(t; 0, y(0))$.

Parameterized complexity for analytic functions

- **Fact:** Assume $\mathbf{f}$ is is analytic on some compact $K \subset dom\ f$. Then
$$|D^{\beta}\mathbf{f}(x)| \leq C_K^{|\beta|+1}\beta! \tag{9}$$
for some C_K for all $\beta \in \mathbb{N}^d$ and $x \in K$.

Theorem (Kawamura Thies Ziegler 2018)

Assume:

1. *The right-hand side* $\mathbf{f} : D \subset \mathbb{R}^d \to \mathbb{R}^d$ *is analytic and computable*
2. *The solution* $\mathbf{y}(t) := \mathbf{y}(t; 0, y_0)$ *exists for all* $t \in [0, 1]$
3. *Restricted to* $K := \mathbf{y}([0, 1]; 0, y_0)$ *the integer* C *is a derivative bound for* $\mathbf{f}$ *(i.e. (9) holds).*
4. *The algorithm computing* $\mathbf{f}$ *gives a* 2^{-n} *approximation of* $\mathbf{f}(x)$ *in time poly*$(n + C)$ *on any* $\mathbf{x} \in K$*.*

Then $\mathbf{y}(t)$ *can be computed* **in polynomial time** *from* $\mathbf{y}_0$*,* $t \in [0, 1]$ *and* $C(\mathbf{y}_0)$*.*

Idea of the proof

1. Idea 1: **Computing a local solution**: A simple truncation of power series based approach suffices to compute a local solution on some small time interval $[t_0, t_0 + \delta]$ in time polynomial in $n + C(y_0)$.

    - See complexity analysis from **[Moiske Müller 93]** of computing an analytic function from the coefficients of its power series.
    - Key remark for the slide to come: $\delta = \frac{1}{2(d+1)C^2}$ is ok.

2. Idea 2: **Extending to a global solution**: To get a solution on a bigger interval this algorithm is iterated several times.

    - it suffices to show that polynomially in $C(y_0)$ many iterations suffice and that it suffices to compute the intermediate values in each iteration with polynomial precision.

Idea 2: Extending to a global solution

Algorithm

$SOLVE - IVP(\mathbf{f}, y_0, t, n, C)$

- $t_{curr} \leftarrow 0$
- $\mathbf{y} \leftarrow APPROX(\mathbf{y}_0, m)$
- $h \leftarrow \frac{1}{2(d+1)C^2}$
- while $t_{curr} + h \leq t$ do
  - $\mathbf{y} \leftarrow LOCALSOLUTION(\mathbf{y}, h, m, C)$
  - $t_{curr} \leftarrow t_{curr} + h$
- return $LOCALSOLUTION(\mathbf{f}, \mathbf{y}, t - t_{curr}, m, C)$

Idea 2: Extending to a global solution

Algorithm

$SOLVE - IVP(\mathbf{f}, y_0, t, n, C)$

- $t_{curr} \leftarrow 0$
- $\mathbf{y} \leftarrow APPROX(\mathbf{y}_0, m)$
- $h \leftarrow \frac{1}{2(d+1)C^2}$
- while $t_{curr} + h \leq t$ do
  - $\mathbf{y} \leftarrow LOCALSOLUTION(\mathbf{y}, h, m, C)$
  - $t_{curr} \leftarrow t_{curr} + h$
- return $LOCALSOLUTION(\mathbf{f}, \mathbf{y}, t - t_{curr}, m, C)$

Analysis:

- $N = 2(d+1)C^2$ steps.
- Error of type 1: precision 2^{-m}
- Error of type 2: instead of solving with initial value y_i we start from an approximation z_i:
  - From Gronwall's Lemma, if $\|\mathbf{y}_0 - \mathbf{z}_0\| \leq \epsilon$ then $\|\mathbf{y}(t; y_0) - \mathbf{y}(t, z_0)\| \leq 2\epsilon$ for $t < \frac{1}{2(d+1)C^2}$.
- The total error E satisfies $E \leq 2^{N+1-m}$.
  - by induction $E_N \leq 2^{N+1-m} - 2^{-m}$ since $E_0 \leq 2^{-m}$, and $E_{k+1} \leq 2E_k + 2^{-m}$
- It suffices to choose $m \geq n + 2(d+1)C^2 + 1$ to prove the theorem.

Parameterized complexity for $\mathbf{f}$ polynomial

Consider

$$\begin{array}{rcl} \mathbf{y}' & = & \mathbf{p}(t, \mathbf{y}) \\ \mathbf{y}(0) & = & \mathbf{y}_0 \end{array}$$

where $\mathbf{p}$ is (some vector of) polynomials.

Theorem (Bournez Graça Pouly 2012)

Then $\mathbf{y}(t)$ *can be computed* **in** *poly*$(t + \log \|y_0\| + \ell)$ **time** *from* $t \in \mathbb{R}$, $\mathbf{y}_0$, $\mathbf{p}$, $\|y_0\|$ *and* ℓ.
where

$$\ell = \int_{t_0}^{t} \max(1, \|y(u)\|_\infty)^{\deg(p)} du \approx \text{length of } y \text{ over } [t_0, t]$$

.

Menu

And conversely ...

- **Turing machines** can be simulated by Ordinary Differential Equations of type:

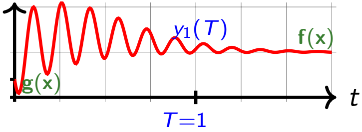
$$\begin{aligned} \mathbf{y}' &= \mathbf{p}(t, \mathbf{y}) \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned}$$

- What about **complexity**?

Time complexity for continuous systems

- Variable t is rather arbitrary.

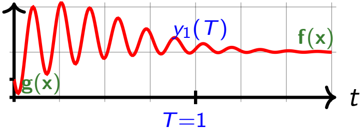
$$y(0) = g(x) \qquad y' = h(y)$$



Time complexity for continuous systems

- Variable t is rather arbitrary.

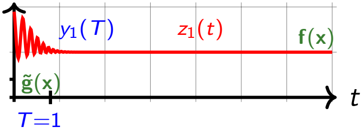
$y(0) = g(x) \qquad y' = h(y)$



$z(t){=}y(e^t)$

$\rightsquigarrow$

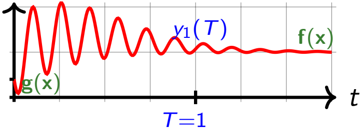
$z(0) = \tilde{g}(x) \qquad z' = \tilde{h}(z)$



Time complexity for continuous systems

- Variable t is rather arbitrary.

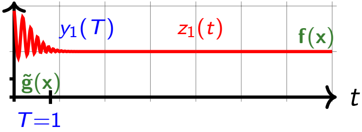
$y(0) = g(x) \qquad y' = h(y)$



$z(t)=y(e^t)$

$\rightsquigarrow$

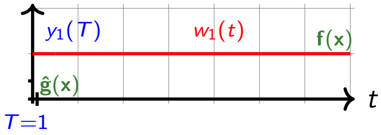
$z(0) = \tilde{g}(x) \qquad z' = \tilde{h}(z)$



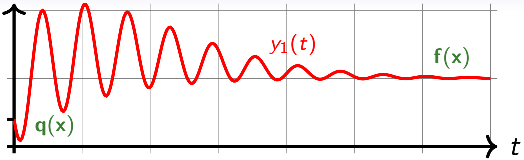
$w(t)=y\left(e^{e^t}\right)$

$\rightsquigarrow$

$w(0) = \hat{g}(x) \qquad w' = \hat{h}(w)$



A Simple & Key Idea: curvilinear abscissa



$$\begin{cases} y(0) = q(x) \\ y'(t) = p(y(t)) \end{cases}$$

Length based: T

$$\ell(t) = \text{length of } y \text{ over } [0, t]$$
$$= \int_0^t \|p(y(u))\|_\infty \, du$$

Consider parameterization

$$t = \text{length of } y \text{ over } [0, t]$$

I.e.:

Follow curve **at constant speed.**

Main Statement: Complexity

- **Theorem**[6] Any polynomial time computable function can be computed in polynomial length, and conversely.

[6] OB, D. Graça, A. Pouly ICALP Track B Best Paper Award [**?**], Journal of the ACM [**?**]

Main Statement: Complexity

- **Theorem**[6] Any polynomial time computable function can be computed in polynomial length, and conversely.

- **The notion of polynomial time computable function can be defined using pODE only !!**

---

[6] OB, D. Graça, A. Pouly ICALP Track B Best Paper Award [**?**], Journal of the ACM [**?**]

Main Statement: Complexity

- **Theorem**[6] Any polynomial time computable function can be computed in polynomial length, and conversely.

- **The notion of polynomial time computable function can be defined using pODE only !!**

  - No need to talk of Turing machines, or equivalent concept to define polynomial time computable functions.

[6] OB, D. Graça, A. Pouly ICALP Track B Best Paper Award [**?**], Journal of the ACM [**?**]

Formal Theorem

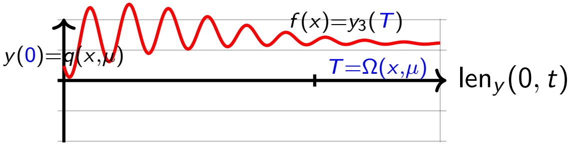
Let $a, b \in \mathbb{Q}$.

- $f \in C^0([a,b], \mathbb{R})$ is polynomial-time computable

iff

- ▶ y satisfies a pODE
- ▶ $y_{1..m}$ is $e^{-\mu}$-close to $f(x)$ after a polynomial length

- Picture:



Formal Theorem

Let $a, b \in \mathbb{Q}$.

- $f \in C^0([a,b],\mathbb{R})$ is polynomial-time computable

iff

$\exists$ polynomials p, q, Ω s.t. $\forall x \in \operatorname{dom} f$,
there exists a (unique) y satisfying for all $t \in \mathbb{R}_+$:

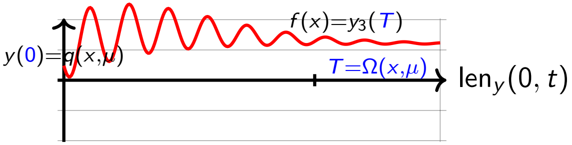
- $y(0) = q(x,\mu)$ and $y'(t) = p(y(t))$ with $\|y'(t)\|_\infty \geqslant 1$

  ▶ y satisfies a pODE

- if $\operatorname{len}_y(0,t) \geqslant \Omega(\|x\|_\infty, \mu)$ then $\|y_{1..m}(t) - f(x)\|_\infty \leqslant e^{-\mu}$

  ▶ $y_{1..m}$ is $e^{-\mu}$-close to $f(x)$ after a polynomial length

- Picture:



For Discrete People

Fix a "reasonable" way to encode words w, length of input, and decision:

- For example $\psi(w) = \left(\sum_{i=1}^{|w|} w_i k^{-i}, |w|\right)$, and $\geqslant 1$, $\leqslant -1$.

Then:

- $\mathcal{L} \subseteq \{0,1\}^*$ is polynomial-time computable

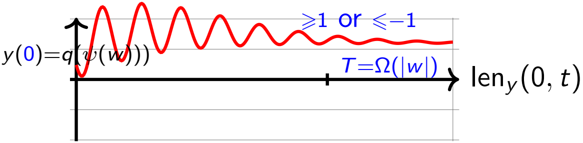
iff

▸ y satisfies a pODE

▸ decision is made after a polynomial length

▸ and corresponds to $\mathcal{L}$

- Picture:



For Discrete People

Fix a "reasonable" way to encode words w, length of input, and decision:

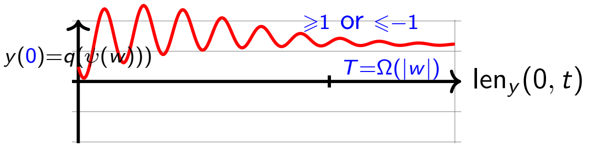
- For example $\psi(w) = \left(\sum_{i=1}^{|w|} w_i k^{-i}, |w|\right)$, and $\geqslant 1$, $\leqslant -1$.

Then:

- $\mathcal{L} \subseteq \{0,1\}^*$ is polynomial-time computable

iff

- $\exists$ polynomials p, q, Ω s.t. $\forall w$,
  there exists a (unique) y satisfying for all $t \in \mathbb{R}_+$:
  - $y(0) = q(\psi(w))$ and $y'(t) = p(y(t))$ with $\|y'(t)\|_\infty \geqslant 1$
    - ▶ y satisfies a pODE
  - if $\text{len}_y(0,t) \geqslant \Omega(|w|)$ then $|y_1(t)| \geqslant 1$
    - ▶ decision is made after a polynomial length
  - $w \in \mathcal{L}$ iff $y_1(t) \geqslant 1$
    - ▶ and corresponds to $\mathcal{L}$
- Picture:



Menu

Conclusion/Take Home Message

- Programming with/Solving ODEs is **simple** and **fun**.
- Solving ODEs over $t \in \mathbb{R}$ is not polynomial.
- Needs for some parameterized algorithms.
  - polynomial in $C(y)$
  - polynomial in length

Conclusion/Take Home Message

- Programming with/Solving ODEs is **simple** and **fun**.

- Solving ODEs over $t \in \mathbb{R}$ is not polynomial.

- Needs for some parameterized algorithms.

  - polynomial in $C(y)$
  - polynomial in length

- Analog's world: Many concepts from **computer science** can be defined using polynomial ODEs

  - Computable functions.

  - Polynomial Time Computable Functions

  - *NP*, *PSPACE*, ...?

Conclusion/Take Home Message

- Programming with/Solving ODEs is **simple** and **fun**.

- Solving ODEs over $t \in \mathbb{R}$ is not polynomial.

- Needs for some parameterized algorithms.

  - polynomial in $C(y)$
  - polynomial in length

- Analog's world: Many concepts from **computer science** can be defined using polynomial ODEs

  - Computable functions.

  - Polynomial Time Computable Functions

  - *NP*, *PSPACE*, ...?

  - Revisiting computation theory with pODEs ...