Computational complexity of solving polynomial differential equations over unbounded domains

Amaury Pouly[⋆,†]
Daniel Graça[†,‡]

[⋆] Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France
[†] CEDMES/FCT, Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal
[‡] SQIG /Instituto de Telecomunicações, Lisbon, Portugal

July 8, 2013

Outline

Problem statement

We want to solve:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases}$$

where

$y\colon I \subseteq \mathbb{R} \to \mathbb{R}^n$
p: vector of polynomials

Solve ?

Problem statement

We want to solve:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases}$$

where

$y\colon I \subseteq \mathbb{R} \to \mathbb{R}^n$
p: vector of polynomials

Solve ? ▷Compute $y_i(t)$ with arbitrary precision for any $t \in I$

Problem statement

We want to solve:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases}$$

where

y: $I \subseteq \mathbb{R} \to \mathbb{R}^n$
p: vector of polynomials

Solve ? ▷Compute $y_i(t)$ with arbitrary precision for any $t \in I$

Example

$$\begin{cases} c'(t) = -s(t) \\ s'(t) = c(t) \\ x'(t) = 2c(t)s(t)x(t)^2 \end{cases} \begin{cases} c(0) = 1 \\ s(0) = 0 \\ x(t) = \frac{1}{2} \end{cases}$$

Problem statement

We want to solve:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases}$$

where

$$y \colon I \subseteq \mathbb{R} \to \mathbb{R}^n$$
p: vector of polynomials

Solve ? ▷Compute $y_i(t)$ with arbitrary precision for any $t \in I$

Example

$$\begin{cases} c'(t) = -s(t) \\ s'(t) = c(t) \\ x'(t) = 2c(t)s(t)x(t)^2 \end{cases} \quad \begin{cases} c(0) = 1 \\ s(0) = 0 \\ x(t) = \frac{1}{2} \end{cases} \quad \rightsquigarrow \begin{cases} c(t) = \cos(t) \\ s(t) = \sin(t) \\ x(t) = \frac{1}{1+\cos(t)^2} \end{cases}$$

Motivation

- Theoretical complexity of solving differential equations
- Functions generated by the General Purpose Analog Computer (GPAC)
- Solve $y' = f(y)$ where f is elementary (composition of polynomials, exponential,logarithms, (inverse) trigonometric functions, ...)

Motivation

- Theoretical complexity of solving differential equations
- Functions generated by the General Purpose Analog Computer (GPAC)
- Solve $y' = f(y)$ where f is elementary (composition of polynomials, exponential,logarithms, (inverse) trigonometric functions, ...)

Motivation

- Theoretical complexity of solving differential equations
- Functions generated by the General Purpose Analog Computer (GPAC)
- Solve $y' = f(y)$ where f is elementary (composition of polynomials, exponential,logarithms, (inverse) trigonometric functions, ...)

Motivation

- Theoretical complexity of solving differential equations
- Functions generated by the General Purpose Analog Computer (GPAC)
- Solve $y' = f(y)$ where f is elementary (composition of polynomials, exponential,logarithms, (inverse) trigonometric functions, ...)

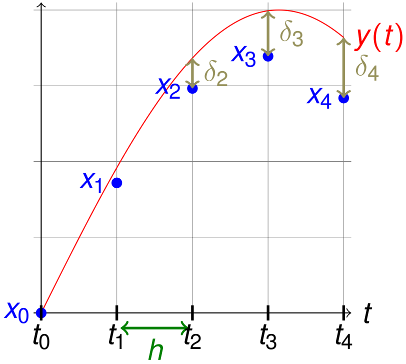
Example

$$\begin{cases} y' = \sin(y) \\ y(0) = 1 \end{cases} \xrightarrow[u=\cos(y)]{z=\sin(y)} \begin{cases} y' = z \\ z' = u \\ u' = -z \end{cases} \begin{cases} y(0) = 1 \\ z(0) = \sin(1) \\ u(0) = \cos(1) \end{cases}$$

Practical

Definition (Folklore)

- Numerical method: $t_{i+1} = t_i + h$ and $x_{i+1} = f(x_0, \dots, x_i; h)$
- Local error: ${\delta_i}^h = \|y(t_i) - x_i\|_\infty$
- Order: maximum ω such that $\delta_n^h = \mathcal{O}\left(h^{\omega+1}\right)$ as $h \to 0$



Practical

Definition (Folklore)

- Numerical method: $t_{i+1} = t_i + h$ and $x_{i+1} = f(x_0, \ldots, x_i; h)$
- Local error: ${\delta_i}^h = \|y(t_i) - x_i\|_\infty$
- Order: maximum ω such that $\delta_n^h = \mathcal{O}\left(h^{\omega+1}\right)$ as $h \to 0$

Theorem (Folklore)

- Euler method has order 1
- Runge-Kutta 4 (RK4) has order 4
- $\forall\omega$, there exist methods of order ω (RKω, Taylor)

Practical

Definition (Folklore)

- Numerical method: $t_{i+1} = t_i + h$ and $x_{i+1} = f(x_0, \ldots, x_i; h)$
- Local error: ${\delta_i}^h = \|y(t_i) - x_i\|_\infty$
- Order: maximum ω such that $\delta_n^h = \mathcal{O}\left(h^{\omega+1}\right)$ as $h \to 0$

Theorem (Folklore)

- Euler method has order 1
- Runge-Kutta 4 (RK4) has order 4
- $\forall\omega$, there exist methods of order ω (RKω, Taylor)

Practical

Definition (Folklore)

- Numerical method: $t_{i+1} = t_i + h$ and $x_{i+1} = f(x_0, \dots, x_i; h)$
- Local error: $\delta_i{}^h = \|y(t_i) - x_i\|_\infty$
- Order: maximum ω such that $\delta_n^h = \mathcal{O}\left(h^{\omega+1}\right)$ as $h \to 0$

Theorem (Folklore)

- Euler method has order 1
- Runge-Kutta 4 (RK4) has order 4
- $\forall\omega$, there exist methods of order ω (RKω, Taylor)

Practical

Definition (Folklore)

- Numerical method: $t_{i+1} = t_i + h$ and $x_{i+1} = f(x_0, \ldots, x_i; h)$
- Local error: ${\delta_i}^h = \|y(t_i) - x_i\|_\infty$
- Order: maximum ω such that $\delta_n^h = \mathcal{O}\left(h^{\omega+1}\right)$ as $h \to 0$

Theorem (Folklore)

- Euler method has order 1
- Runge-Kutta 4 (RK4) has order 4
- $\forall\omega$, there exist methods of order ω (RKω, Taylor)

Remark

- Difficult choice of h
- Quite efficient in practice

Practical (Handwaving)

Definition (Folklore)

- Adaptive method: $t_{i+1} = t_i + h_i$ and $x_{i+1} = f(x_0, \ldots, x_i; h)$
- Local error: $\delta_i = \|y(t_i) - x_i\|_\infty$
- Error estimate: $e_i \geqslant \delta_i, \rightarrow h_i = g(e_i, x, t)$

Idea

- Big steps when smooth and small error estimate
- Small steps when stiff and big error estimate

Remark

- Unknown complexity
- Very efficient in practice

And so ?

Don't we know everything ?

And so ?

Don't we know everything ? Not quite!

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\colon \text{vector of polynomials} \end{array}$$

- Issue #1: order ω, step size h

$$\text{local error} = \mathcal{O}\left(h^{\omega+1}\right)$$

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

- Issue #1: order ω, step size h

$$\text{local error} \leqslant Kh^{\omega+1}$$

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

- Issue #1: order ω, step size h

  local error $\leqslant K h^{\omega+1}$ $\qquad$ K depends on y and I !!

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \quad \text{where} \quad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

- Issue #1: order ω, step size h

$$\text{local error} \leqslant Kh^{\omega+1} \qquad K \text{ depends on } y \text{ and } I \text{ !!}$$

Example: Euler method (Simplified)

$$\text{local error at step } i \leqslant \frac{1}{2}h^2 \left\| p'(y_i) \right\|_\infty$$

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

- Issue #1: order ω, step size h

$$\text{local error} \leqslant Kh^{\omega+1} \qquad K \text{ depends on } y \text{ and } I \text{ !!}$$

Example: Euler method (Simplified)

$$\text{local error} \leqslant \frac{1}{2}h^2 \left\|p'(y_i)\right\|_\infty \Rightarrow \mathcal{O}(1) = \max_{t\in I} \left\|p'(y(t))\right\|_\infty ?$$

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \quad \text{where} \quad \begin{array}{l} y\colon I \subseteq [0,1] \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

- Issue #1: order ω, step size h

$$\text{local error} \leqslant Kh^{\omega+1} \qquad K \text{ depends on } y \text{ and } I \text{ !!}$$

Example: Euler method (Simplified)

$$\text{local error} \leqslant \frac{1}{2}h^2 \left\|p'(y_i)\right\|_\infty \Rightarrow \mathcal{O}(1) = \max_{t\in I} \left\|p'(y(t))\right\|_\infty ?$$

Yes because $[0,1]$ is a compact set...

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\colon \text{vector of polynomials} \end{array}$$

- Issue #1: order ω, step size h

$$\text{local error} \leqslant Kh^{\omega+1} \qquad K \text{ depends on } y \text{ and } I \text{ !!}$$

Example: Typical assumptions

- $I \subseteq [0, 1]$
- p is a lipschitz function

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

- Issue #1: unrealistic assumptions

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\text{: vector of polynomials} \end{array}$$

- Issue #1: unrealistic assumptions

Idea: rescale!

If $I = [a, b]$, write $z(t) = y(a + (b - a)t)$, then:

$$z : [0, 1] \to \mathbb{R}^n \qquad \rightsquigarrow \qquad \begin{cases} z' &= (b - a)p(z) \\ z(t_0') &= z_0 \end{cases}$$

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\colon \text{vector of polynomials} \end{array}$$

- Issue #1: unrealistic assumptions

Idea: rescale!

If $I = [a, b]$, write $z(t) = y(a + (b - a)t)$, then:

$$z : [0, 1] \to \mathbb{R}^n \qquad \rightsquigarrow \qquad \begin{cases} z' &= (b - a)p(z) \\ z(t_0') &= z_0 \end{cases}$$

Still need lipschitz condition, now depends on p, a and b.

And so ?

Don't we know everything ? Not quite!

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases} \qquad \text{where} \qquad \begin{array}{l} y\colon I \to \mathbb{R}^n \\ p\colon \text{vector of polynomials} \end{array}$$

- Issue #1: unrealistic assumptions
- Issue #2: rescaling doesn't help

Computability

Theorem (Pieter Collins, Daniel Graça)

Let $I \subseteq \mathbb{R}$ open set, $t_0 \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}^n$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = f(y(t))$$

If y_0 is a computable real, p has computable coefficients and f is computable then y is a computable function.

Computability

Theorem (Pieter Collins, Daniel Graça)

Let $I \subseteq \mathbb{R}$ open set, $t_0 \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}^n$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = f(y(t))$$

If y_0 is a computable real, p has computable coefficients and f is computable then y is a computable function.

Remark

- f computable $\Rightarrow$ f continuous $\Rightarrow$ unique solution
- We have to assume the existence over I because finding I is undecidable.
- Absolutely terrible complexity

Complexity

Theorem (ICALP 2012)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, Y, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t)) \text{ and } \|y(t)\|_\infty \leqslant Y$$

If y_0 is a polytime computable real and p has polytime computable coefficients, then one can compute x such that $\|x - y(u)\|_\infty \leqslant 2^{-\mu}$ in time poly(μ, u, Y).

Complexity

Theorem (ICALP 2012)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, Y, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t)) \text{ and } \|y(t)\|_\infty \leqslant Y$$

If y_0 is a polytime computable real and p has polytime computable coefficients, then one can compute x such that $\|x - y(u)\|_\infty \leqslant 2^{-\mu}$ in time poly(μ, u, Y).

Complexity

Theorem (ICALP 2012)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, Y, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t)) \text{ and } \|y(t)\|_\infty \leqslant Y$$

If y_0 is a polytime computable real and p has polytime computable coefficients, then one can compute x such that $\|x - y(u)\|_\infty \leqslant 2^{-\mu}$ in time $\text{poly}(\mu, u, Y)$.

Remark

- Impossible to bound complexity without Y or something similar
- If $I \subseteq [0, 1]$, this is "polytime" in $\text{poly}(\mu)$
- Very inefficient in practice

Goal

- Complexity of practical adaptive algorithms ?
- Theoretical power of adaptiveness ?

Goal

- Complexity of practical adaptive algorithms ?⇒Too ambitious
- Theoretical power of adaptiveness ?Yes!

Our result

Theorem (CCA 2013)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, Y, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

If y_0 is a polytime computable real and p has polytime computable coefficients, then one can compute x such that $\|x - y(u)\|_\infty \leqslant 2^{-\mu}$ in time $\text{poly}(\mu, u, Z)$ where

$$Z \approx \int_{t_0}^{u} \text{poly}(\|y(\xi)\|_\infty) d\xi$$

Our result

Theorem (CCA 2013)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, Y, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

If y_0 is a polytime computable real and p has polytime computable coefficients, then one can compute x such that $\|x - y(u)\|_\infty \leqslant 2^{-\mu}$ in time $\text{poly}(\mu, u, Z)$ where

$$Z \approx \int_{t_0}^{u} \text{poly}(\|y(\xi)\|_\infty) d\xi$$

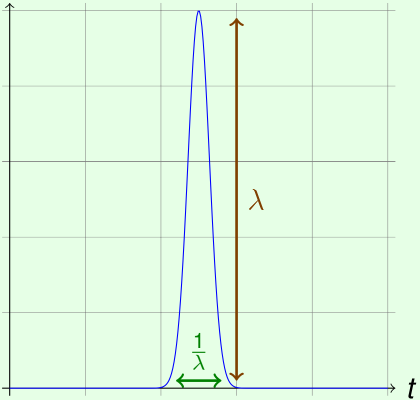
Remark

- Always better than our previous result
- Doesn't need an *a priori* bound on the solution

Example: why is this better ?

Example

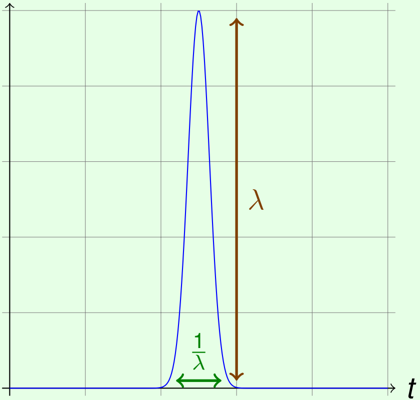
$$f_{\lambda,u}(t) = \lambda e^{-\lambda^2(u-t)^2}$$



Example: why is this better ?

Example

$$f_{\lambda,u}(t) = \lambda e^{-\lambda^2(u-t)^2}$$



Previous method (ICALP 2012)

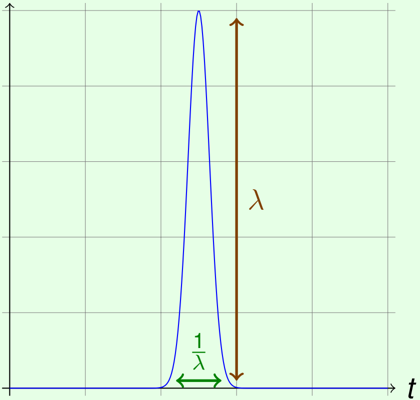
Complexity: poly(t, l_λ)

$$l_\lambda = \max_{t\in I} \|y(t)\|_\infty = \lambda$$

Example: why is this better ?

Example

$$f_{\lambda,u}(t) = \lambda e^{-\lambda^2(u-t)^2}$$



Previous method (ICALP 2012)

Complexity: poly(t, l_λ)

$$l_\lambda = \max_{t\in I} \|y(t)\|_\infty = \lambda$$

Adaptive method (CCA 2013)

Complexity: poly(t, K_λ)

$$K_\lambda = \int_{t\in I} \|y(t)\|_\infty \, dt = \mathcal{O}(1)$$

Euler method

Idea

$$y(t+h) \approx y(t) + hy'(t) \approx y(t) + hp(y(t))$$

Euler method

Idea

$$y(t+h) \approx y(t) + hy'(t) \approx y(t) + hp(y(t))$$

- Discretise: make N time steps

Euler method

Idea

$$y(t+h) \approx y(t) + hy'(t) \approx y(t) + hp(y(t))$$

- Discretise: make N time steps
- Do a linear approximation at each step

Euler method

Idea

$$y(t+h) \approx y(t) + hy'(t) \approx y(t) + hp(y(t))$$

- Discretise: make N time steps
- Do a linear approximation at each step

$$x_0 = y_0 \qquad x_{n+1} = x_n + h\,p(x_n) \qquad t = Nh + t_0$$

Euler method

Idea

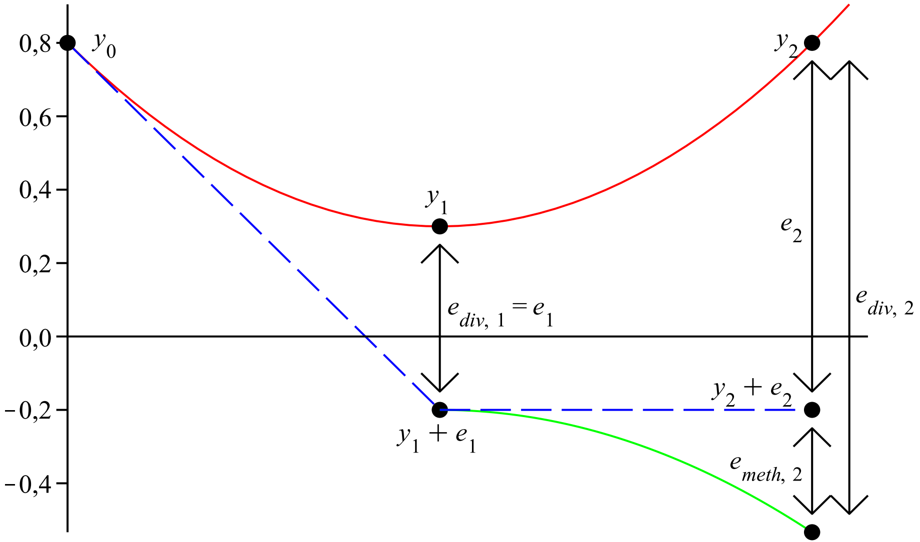
$$y(t+h) \approx y(t) + hy'(t) \approx y(t) + hp(y(t))$$

- Discretise: make N time steps
- Do a linear approximation at each step

$$x_0 = y_0 \qquad x_{n+1} = x_n + h\,p(x_n) \qquad t = Nh + t_0$$

Doesn't work very well !

Euler method (2)



Taylor method

Idea

$$y(t+h) \approx y(t) + \sum_{i=1}^{\omega} h^i y^{(i)}(t) \qquad y^{(i)}(t) = \text{poly}_i(y(t))$$

Taylor method

Idea

$$y(t+h) \approx y(t) + \sum_{i=1}^{\omega} h^i y^{(i)}(t) \qquad y^{(i)}(t) = \mathrm{poly}_i(y(t))$$

- Discretise: make N time steps

Taylor method

Idea

$$y(t+h) \approx y(t) + \sum_{i=1}^{\omega} h^i y^{(i)}(t) \qquad y^{(i)}(t) = \mathrm{poly}_i(y(t))$$

- Discretise: make N time steps
- Do a ω-th order approximation at each step

Taylor method

Idea

$$y(t+h) \approx y(t) + \sum_{i=1}^{\omega} h^i y^{(i)}(t) \qquad y^{(i)}(t) = \text{poly}_i(y(t))$$

- Discretise: make N time steps
- Do a ω-th order approximation at each step

$$x_0 = y_0 \qquad x_{n+1} = x_n + \sum_{i=1}^{\omega} h^i \,\text{poly}_i(x_n) \qquad t = Nh + t_0$$

Taylor method

Idea

$$y(t+h) \approx y(t) + \sum_{i=1}^{\omega} h^i y^{(i)}(t) \qquad y^{(i)}(t) = \text{poly}_i(y(t))$$

- Discretise: make N time steps
- Do a ω-th order approximation at each step

$$x_0 = y_0 \qquad x_{n+1} = x_n + \sum_{i=1}^{\omega} h^i \,\text{poly}_i(x_n) \qquad t = Nh + t_0$$

Works much better for $\omega \geqslant 3$. How to choose h and ω ?

Adaptive variable-order Taylor method

Idea

Change the time step and the order at each step.

Adaptive variable-order Taylor method

Idea

Change the time step and the order at each step.

$$x_0 = y_0 \qquad x_{n+1} = x_n + \sum_{i=1}^{\omega_n} {h_n}^i \operatorname{poly}_i(x_n) \qquad t = \sum_{i=1}^{N} h_i + t_0$$

where

Adaptive variable-order Taylor method

Idea

Change the time step and the order at each step.

$$x_0 = y_0 \qquad x_{n+1} = x_n + \sum_{i=1}^{\omega_n} {h_n}^i \operatorname{poly}_i(x_n) \qquad t = \sum_{i=1}^{N} h_i + t_0$$

where

$$h_n = \frac{1}{\operatorname{poly}(\|x_n\|_\infty)} \qquad \omega_n = \log_2 \operatorname{poly}\left(\|x_n\|_\infty, K, \frac{1}{\varepsilon}\right) \qquad N = \operatorname{poly}(K)$$

$$\varepsilon = \text{output precision} \qquad K \geqslant \int_{t_0}^{t} \operatorname{poly}(\|y(u)\|_\infty) du$$

Adaptive variable-order Taylor method

Idea

Change the time step and the order at each step.

$$x_0 = y_0 \qquad x_{n+1} = x_n + \sum_{i=1}^{\omega_n} {h_n}^i \operatorname{poly}_i(x_n) \qquad t = \sum_{i=1}^{N} h_i + t_0$$

where

$$h_n = \frac{1}{\operatorname{poly}(\|x_n\|_\infty)} \qquad \omega_n = \log_2 \operatorname{poly}\left(\|x_n\|_\infty, K, \frac{1}{\varepsilon}\right) \qquad N = \operatorname{poly}(K)$$

$$\varepsilon = \text{output precision} \qquad K \geqslant \int_{t_0}^{t} \operatorname{poly}(\|y(u)\|_\infty)du$$

Remark

We need to know $\int_{t_0}^{t} \operatorname{poly}(\|y(u)\|_\infty)du$

Complexity

Theorem (Complexity)

If y_0 and p are polytime computable, $\mathcal{A}(t_0, y_0, p, K, u, \mu)$ has running time poly$(u - t_0, K, \mu)$.

Complexity

Theorem (Complexity)

If y_0 and p are polytime computable, $\mathcal{A}(t_0, y_0, p, K, u, \mu)$ has running time poly$(u - t_0, K, \mu)$.

Proof ideas

- Show that derivatives of y can be computed quickly from p
- Tedious computations

A crucial property

Theorem (Algorithm is correct)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, K, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

There exist an algorithm $\mathcal{A}$ such that

$$K \geqslant \int_{t_0}^{t} \text{poly}(\|y(\xi)\|_\infty) d\xi \quad \Rightarrow \quad \|\mathcal{A}(t_0, y_0, p, K, u, \mu) - y(u)\|_\infty \leqslant e^{-\mu}$$

A crucial property

Theorem (Algorithm is correct)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, K, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

There exist an algorithm $\mathcal{A}$ such that

$$K \geqslant \int_{t_0}^{t} \text{poly}(\|y(\xi)\|_\infty) d\xi \quad \Rightarrow \quad \|\mathcal{A}(t_0, y_0, p, K, u, \mu) - y(u)\|_\infty \leqslant e^{-\mu}$$

Proof ideas

- Bound dependency in the initial condition
- Tedious error analysis

A crucial property

Theorem (Algorithm is correct)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, K, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

There exist an algorithm $\mathcal{A}$ such that

$$K \geqslant \int_{t_0}^{t} \operatorname{poly}(\|y(\xi)\|_\infty) d\xi \quad \Rightarrow \quad \|\mathcal{A}(t_0, y_0, p, K, u, \mu) - y(u)\|_\infty \leqslant e^{-\mu}$$

Remark

What if we give $\mathcal{A}$ a K which is not big enough ?

A crucial property

Theorem (Algorithm is correct)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, K, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

There exist an algorithm $\mathcal{A}$ such that

$$K \geqslant \int_{t_0}^{t} \text{poly}(\|y(\xi)\|_\infty) d\xi \quad \Rightarrow \quad \|\mathcal{A}(t_0, y_0, p, K, u, \mu) - y(u)\|_\infty \leqslant e^{-\mu}$$

Remark

What if we give $\mathcal{A}$ a K which is not big enough ?

Theorem (Algorithm is complete)

$\mathcal{A}$ can detect if K is not big enough.

A crucial property

Theorem (Algorithm is correct)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, K, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

There exist an algorithm $\mathcal{A}$ such that

$$K \geqslant \int_{t_0}^{t} \text{poly}(\|y(\xi)\|_\infty) d\xi \quad \Rightarrow \quad \|\mathcal{A}(t_0, y_0, p, K, u, \mu) - y(u)\|_\infty \leqslant e^{-\mu}$$

Theorem (Algorithm is complete)

$\mathcal{A}$ can detect if K is not big enough.

Proof ideas

- Clever bound on the number of steps

Enhanced algorithm

Idea

Start with $K = 1$. While $\mathcal{A}$ fails, double K.

Enhanced algorithm

Idea

Start with $K = 1$. While $\mathcal{A}$ fails, double K.

Theorem (CCA 2013)

Let $I \subseteq \mathbb{R}$ open set, $t_0, u \in I, y_0 \in \mathbb{R}^n, y : I \to \mathbb{R}^n, Y, \mu > 0$. Assume

$$y(t_0) = y_0 \quad \text{and} \quad \forall t \in I, y'(t) = p(y(t))$$

If y_0 is a polytime computable real and p has polytime computable coefficients, then one can compute x such that $\|x - y(u)\|_\infty \leqslant 2^{-\mu}$ in time $\text{poly}(\mu, u, Z)$ where

$$Z \approx \int_{t_0}^{u} \text{poly}(\|y(\xi)\|_\infty) d\xi$$

Conclusion

- Adaptive algorithm to solve polynomial initial value problem
- Proven complexity
- Theoretical power of adaptiveness

Conclusion

- Adaptive algorithm to solve polynomial initial value problem
- Proven complexity
- Theoretical power of adaptiveness

Conclusion

- Adaptive algorithm to solve polynomial initial value problem
- Proven complexity
- Theoretical power of adaptiveness

Future Work

- General study of explicit methods
- Study implicit methods
- Lower bound on complexity of solving initial value problem
- Lower bound on adaptive algorithms

Future Work

- General study of explicit methods
- Study implicit methods
- Lower bound on complexity of solving initial value problem
- Lower bound on adaptive algorithms

Future Work

- General study of explicit methods
- Study implicit methods
- Lower bound on complexity of solving initial value problem
- Lower bound on adaptive algorithms

Future Work

- General study of explicit methods
- Study implicit methods
- Lower bound on complexity of solving initial value problem
- Lower bound on adaptive algorithms

Questions ?

- Do you have any questions ?

Hidden table

Method	Max. Order At Point u	Guaranteed Hint	Number of steps
Previous (with hint I)*	$\mathcal{O}\left(\log \frac{I}{\varepsilon}\right)$	$k\Sigma p(t-t_0)\times \sup_{u\in[t_0,t]} (1+\\|y(u)\\|_\infty)^{k-1}$	$2I$
Fixed ω (with hint I)†	$\omega = \frac{1}{\lambda}$	$I \geqslant K_\lambda$	$1+(3I)^{\frac{\omega+1}{\omega-1}}\left(\frac{k+\lambda}{\varepsilon}\right)^{\frac{1}{1-\lambda}}$
Fixed ω (enhanced)†	$\omega = \frac{1}{\lambda}$	Not Applicable	$r+\left(3\cdot 2^{r+1}\right)^{\frac{\omega+1}{\omega-1}}\left(\frac{k+\lambda}{\varepsilon}\right)^{\frac{1}{1-\lambda}}$ where $r=\lceil \log_2 K_\lambda \rceil$
Variable (with hint I)	$\mathcal{O}\left(\log \frac{K\\|y(u)\\|_\infty}{\varepsilon}\right)$	$I \geqslant K_0$	$1+12(k+1)I$
Variable (enhanced)	$\mathcal{O}\left(\log \frac{K_0\\|y(u)\\|_\infty}{\varepsilon}\right)$	Not Applicable	$r+12(k+1)2^{r+1}$ where $r=\lceil \log_2 K_0 \rceil$

$$\text{where} \quad K_\lambda = \int_{t_0}^{t} k\Sigma p(1+\varepsilon+\|y(u)\|_\infty)^{k-1+\lambda}\,du$$

*This algorithm only works if the given hint is greater than the guaranteed hint, the result is otherwise undefined.

†This algorithm can detect if the hint is not large enough.