17. ODEs IV: Taylor methods

Last time

- Error control
- Variable step size algorithm

Goals for today

- Transforming non-autonomous to autonomous ODEs
- Taylor series solutions
- Taylor method
- Picard iteration

Non-autonomous ODEs

- A **non-autonomous** ODE is $\dot{x}(t) = f(t, x(t))$ where f depends *explicitly* on t
- E.g. $\dot{x}(t) = -x(t) + \cos(t)$
- We can transform this into an autonomous ODE:
- Introduce a new variable z with $\dot{z} = 1$ and $z(0) = 0$
- Then $z(t) = t$
- Get autonomous system

$$\dot{x} = -x + \cos(z)\dot{z} = 1$$

Taylor series solutions

- Methods we have seen reconstruct Taylor series
- But avoid derivative calculations (in a clever way)

Taylor series solutions

- Methods we have seen reconstruct Taylor series
- But avoid derivative calculations (in a clever way)
- Can we just calculate the Taylor series directly?
- Yes: **Taylor method**

Taylor method

- Solve $\dot{x}(t) = f(x(t))$, initial condition $x(t = t_0) = x_0$
- Take $t_0 = 0$ for simplicity

Taylor method

- Solve $\dot{x}(t) = f(x(t))$, initial condition $x(t = t_0) = x_0$
- Take $t_0 = 0$ for simplicity
- Suppose f is **analytic**
- i.e. is equal to its Taylor expansion

$$f(x) = \tilde{f}_0 + \tilde{f}_1 x + \tilde{f}_2 x^2 + \cdots$$

Taylor method II

- Then can show that solution $x(t)$ exists and has power series in t:

$$x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$$

Taylor method II

- Then can show that solution $x(t)$ exists and has power series in t:

$$x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$$

- We want to calculate the Taylor coefficients x_i
- Note: we have $x(0) = x_0$; $\dot{x}(0) = x_1$; $\ddot{x}(0) = 2x_2$, ...

Taylor method II

- Then can show that solution $x(t)$ exists and has power series in t:

$$x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$$

- We want to calculate the Taylor coefficients x_i
- Note: we have $x(0) = x_0$; $\dot{x}(0) = x_1$; $\ddot{x}(0) = 2x_2$, ...

- How can we calculate the x_i?

Taylor method III

- Let's **substitute** the Taylor series for $x(t)$ into the ODE:
  - on left-hand side, need $\dot{x}(t)$
  - on right-hand side, need $f(x(t))$

Taylor method III

- Let's **substitute** the Taylor series for $x(t)$ into the ODE:
  - on left-hand side, need $\dot{x}(t)$
  - on right-hand side, need $f(x(t))$
- Both of these give *new Taylor series*:

$$\dot{x}(t) = x_1 + 2x_2 t + 3x_3 t^2 + \cdots$$

- Substituting $x(t)$ into $f(x(t))$ gives series *in* t:

$$f(x(t)) = f_0 + f_1 t + f_2 t^2 + \cdots$$

Taylor method IV

- Since these two series are equal for all t *coefficient of* t^n *must be equal for each* n
- To prove this e.g. differentiate repeatedly

Taylor method IV

- Since these two series are equal for all t *coefficient of t^n must be equal for each n*
- To prove this e.g. differentiate repeatedly
- Equate coefficients of each power t^n:

$$\begin{aligned} x_1 &= f_0 \\ 2x_2 &= f_1 \\ &\vdots \\ nx_n &= f_{n-1} \end{aligned}$$

- Gives **recurrence relations**: x_n in terms of f_{n-1}

Taylor method V

- For f_{n-1}: insert Taylor series for $x(t)$ into $f(x)$
- f_{n-1} is coefficient of t^{n-1}
- So f_{n-1} can depend only on x_0 up to x_{n-1}
- So from coefficients up to x_{n-1}, obtain x_n!
- *Recursively* generates all coefficients x_n in the Taylor expansion one by one

Example

- E.g. Solve $\dot{x} = x^2$ with $x_0 = 1$
- Start with all coefficients unknown *except* x_0:

$$x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$$

where {red denotes as-yet-unknown coefficients

Example

- E.g. Solve $\dot{x} = x^2$ with $x_0 = 1$
- Start with all coefficients unknown *except* x_0:

$$x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$$

where {red denotes as-yet-unknown coefficients

- So

$$\begin{aligned} f(x(t)) &= [x(t)]^2 \\ &= (x_0 + x_1 t + \cdots)^2 \\ &= x_0^2 + \mathcal{O}(t) \end{aligned}$$

Example II

- So $x_1 = f_0/1 = x_0^2$

Example II

- So $x_1 = f_0/1 = x_0^2$
- Now have $x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$

Example II

- So $x_1 = f_0/1 = x_0^2$
- Now have $x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$
- Substitute into $f(x)$ again:

$$f(x(t)) = x_0^2 + t(x_0 x_1 + x_1 x_0) + \mathcal{O}(t^2)$$

Example II

- So $x_1 = f_0/1 = x_0^2$
- Now have $x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$
- Substitute into $f(x)$ again:

$$f(x(t)) = x_0^2 + t(x_0 x_1 + x_1 x_0) + \mathcal{O}(t^2)$$

- Gives us f_1 = (coefficient of t) = $2x_0 x_1$

Example II

- So $x_1 = f_0/1 = x_0^2$
- Now have $x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$
- Substitute into $f(x)$ again:

$$f(x(t)) = x_0^2 + t(x_0 x_1 + x_1 x_0) + \mathcal{O}(t^2)$$

- Gives us f_1 = (coefficient of t) = $2x_0 x_1$
- Hence $x_2 = f_1/2 = x_0\, x_1$

Example II

- So $x_1 = f_0/1 = x_0^2$
- Now have $x(t) = x_0 + x_1 t + x_2 t^2 + \cdots$
- Substitute into $f(x)$ again:

$$f(x(t)) = x_0^2 + t(x_0 x_1 + x_1 x_0) + \mathcal{O}(t^2)$$

- Gives us f_1 = (coefficient of t) = $2x_0 x_1$
- Hence $x_2 = f_1/2 = x_0\, x_1$
- Repeat, including new coefficient to $x(t)$
- Note: previous f_i are recalculated – inefficient

Alternative viewpoint: Integrals

- Alternative viewpoint: integral formulation of the ODE:

$$x(t) = x_0 + \int_0^t f(x(s))\, ds$$

- Define nth order polynomial approximation:

$$x^{(n)}(t) := x_0 + \cdots + x_n t^n$$

- **Picard iteration** to calculate $x^{(n)}$ recursively:

$$x^{(n+1)} = x_0 + \int \hat{f}^{(n)} \left(x^{(n)} \right)$$

Example using integrals

- Back to example:

$$\dot{x} = x^2 \quad \text{with } x^{(0)} := x_0$$

Example using integrals

- Back to example:

$$\dot{x} = x^2 \quad \text{with } x^{(0)} := x_0$$

$$x^{(1)} = x_0 + \int \left(x^{(0)}\right)^2 = x_0 + \int_0^t x_0^2 \, ds = x_0 + t \, x_0^2$$

Example using integrals

- Back to example:

$$\dot{x} = x^2 \quad \text{with } x^{(0)} := x_0$$

$$x^{(1)} = x_0 + \int \left(x^{(0)}\right)^2 = x_0 + \int_0^t x_0^2 \, ds = x_0 + t \, x_0^2$$

$$x^{(2)} = x_0 + \int \left(x^{(1)}\right)^2 = x_0 + \int_0^t (x_0 + s \, x_0^2)^2 \, ds = x_0 + t \, x_0^2$$

Implementation

- How can we automate this in Julia?
- What operations do we need?

Implementation

- How can we automate this in Julia?
- What operations do we need?

- We are just manipulating polynomials!
- And truncating to a certain degree
- So define operations like $*$ on polynomials of degree n that return polynomials of the *same* degree

Implementation

- How can we automate this in Julia?
- What operations do we need?

- We are just manipulating polynomials!
- And truncating to a certain degree
- So define operations like $*$ on polynomials of degree n that return polynomials of the *same* degree
- These manipulations are done with *numeric* coefficients

Summary

- Can generate Taylor methods of arbitrary order
- Recursive calculation of coefficients
- Uses polynomial manipulation