Ordinary Differential Equations
Sci. Comp. 2020

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Accuracy
- Stability
- Implicit methods
- Trapezoid method
- Runge Kutta

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Accuracy
- Stability
- Implicit methods
- Trapezoid method
- Runge Kutta

Ordinary diff. eq. Part1

Differential equations involve derivatives of unknown solution function

Ordinary differential equation (ODE): all derivatives are with respect to single independent variable, often representing time

Solution of differential equation is continuous *function* in infinite-dimensional space of functions

Numerical solution of differential equations is based on finite-dimensional approximation

Differential equation is replaced by algebraic equation whose solution approximates that of given differential equation

Ordinary diff. eq. Part2

Order determined by highest-order derivative of solution function appearing in ODE

ODE with higher-order derivatives can be transformed into equivalent first-order system

We will discuss numerical solution methods only for first-order ODEs

Most ODE software is designed to solve only first-order equations

Ordinary diff. eq. Part3

For k-th order ODE

$$y^{(k)}(t) = f(t, y, y', \ldots, y^{(k-1)})$$

define k new unknown functions

$$u_1(t) = y(t),\ u_2(t) = y'(t),\ \ldots,\ u_k(t) = y^{(k-1)}(t)$$

Then original ODE is equivalent to first-order system

$$\begin{bmatrix} u_1'(t) \\ u_2'(t) \\ \vdots \\ u_{k-1}'(t) \\ u_k'(t) \end{bmatrix} = \begin{bmatrix} u_2(t) \\ u_3(t) \\ \vdots \\ u_k(t) \\ f(t, u_1, u_2, \ldots, u_k) \end{bmatrix}$$

Ordinary diff. eq. Part4

Newton's Second Law of Motion, $F = ma$, is second-order ODE, since acceleration a is second derivative of position coordinate, which we denote by y

Thus, ODE has form

$$y'' = F/m$$

where F and m are force and mass, respectively

Defining $u_1 = y$ and $u_2 = y'$ yields equivalent system of two first-order ODEs

$$\begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} u_2 \\ F/m \end{bmatrix}$$

We can now use methods for first-order equations to solve this system

First component of solution u_1 is solution y of original second-order equation

Second component of solution u_2 is velocity y'

Ordinary diff. eq. Part5

General first-order system of ODEs has form

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y})$$

where $\boldsymbol{y} : \mathbb{R} \to \mathbb{R}^n$, $\boldsymbol{f} : \mathbb{R}^{n+1} \to \mathbb{R}^n$, and $\boldsymbol{y}' = d\boldsymbol{y}/dt$ denotes derivative with respect to t,

$$\begin{bmatrix} y_1'(t) \\ y_2'(t) \\ \vdots \\ y_n'(t) \end{bmatrix} = \begin{bmatrix} dy_1(t)/dt \\ dy_2(t)/dt \\ \vdots \\ dy_n(t)/dt \end{bmatrix}$$

Function $\boldsymbol{f}$ is given and we wish to determine unknown function $\boldsymbol{y}$ satisfying ODE

For simplicity, we will often consider special case of single scalar ODE, $n = 1$

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Accuracy
- Stability
- Implicit methods
- Trapezoid method
- Runge Kutta

Initial value problem Part1

By itself, ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ does not determine unique solution function

This is because ODE merely specifies *slope* $\boldsymbol{y}'(t)$ of solution function at each point, but not actual value $\boldsymbol{y}(t)$ at any point

If $\boldsymbol{y}(t)$ is solution and $\boldsymbol{c}$ is any constant, then $\boldsymbol{y}(t) + \boldsymbol{c}$ is also a solution because $d(\boldsymbol{y}(t) + \boldsymbol{c})/dt = \boldsymbol{y}'(t) + \boldsymbol{0} = \boldsymbol{y}'(t)$

Infinite family of functions satisfies ODE, in general, provided $\boldsymbol{f}$ is sufficiently smooth

To single out particular solution, value $\boldsymbol{y}_0$ of solution function must be specified at some point t_0

Initial value problem Part2

Thus, part of given problem data is requirement that $\boldsymbol{y}(t_0) = \boldsymbol{y}_0$, which determines unique solution to ODE

Because of interpretation of independent variable t as time, we think of t_0 as initial time and $\boldsymbol{y}_0$ as initial value

Hence, this is termed *initial value problem*, or *IVP*

ODE governs evolution of system in time from its initial state $\boldsymbol{y}_0$ at time t_0 onward, and we seek function $\boldsymbol{y}(t)$ that describes state of system as function of time

Initial value problem Part3

Consider scalar ODE

$$y' = y$$

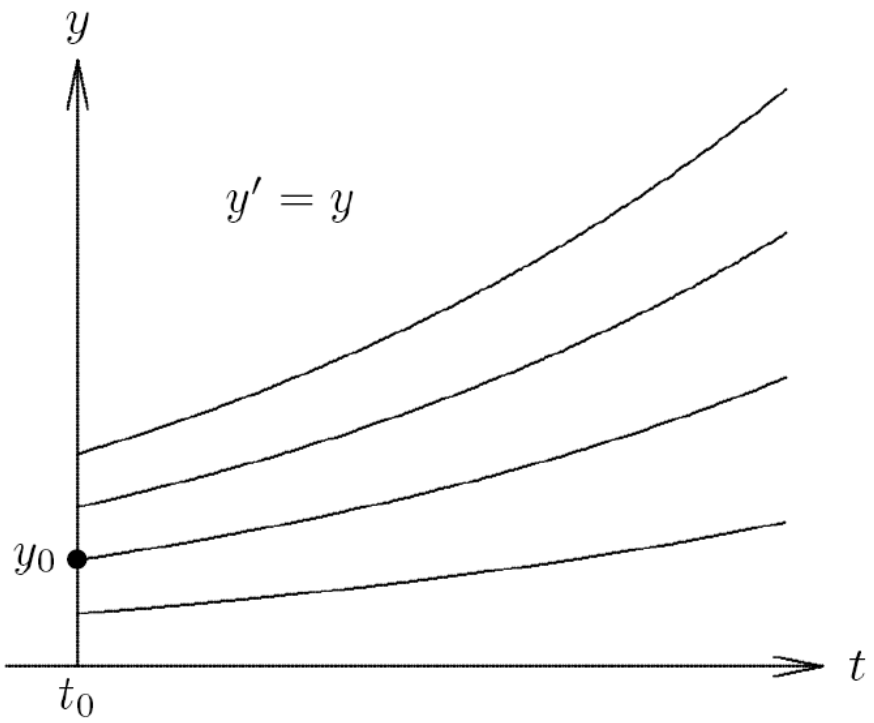
Family of solutions is given by $y(t) = c\,e^t$, where c is any real constant

Imposing initial condition $y(t_0) = y_0$ singles out unique particular solution

For this example, if $t_0 = 0$, then $c = y_0$, which means that solution is $y(t) = y_0 e^t$

Initial value problem Part4

Family of solutions for ODE $y' = y$



Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Accuracy
- Stability
- Implicit methods
- Trapezoid method
- Runge Kutta

Stability Part1

Stable if solutions resulting from perturbations of initial value remain close to original solution

Asymptotically stable if solutions resulting from perturbations converge back to original solution

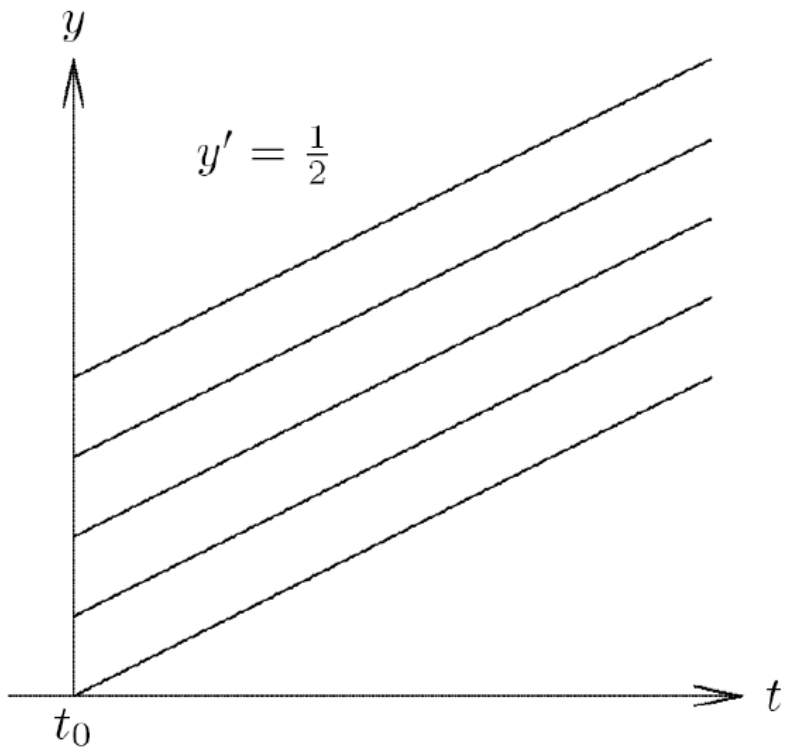
Unstable if solutions resulting from perturbations diverge away from original solution without bound

t_0

Stability Part2

Stable Solutions

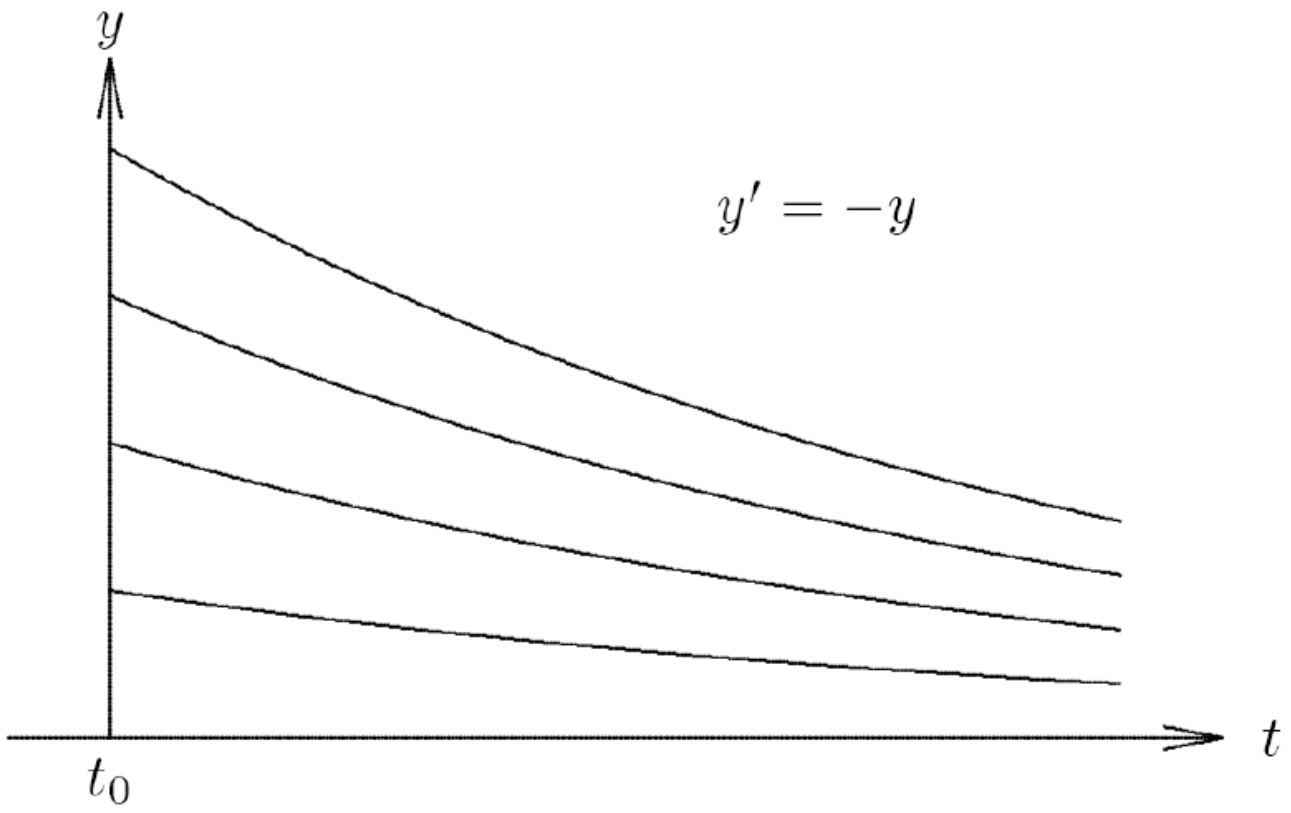
Family of solutions for ODE $y' = \frac{1}{2}$



Stability Part3

Asymptotically Stable Solutions

Family of solutions for ODE $y' = -y$



Stability Part4

- Consider scalar ODE $y' = \lambda y$, where λ is constant.
- Solution is given by $y(t) = y_0\, e^{\lambda t}$, where $t_0 = 0$ is initial time and $y(0) = y_0$ is initial value
- For real λ
  - $\lambda > 0$: all nonzero solutions grow exponentially, so every solution is unstable
  - $\lambda < 0$: all nonzero solutions decay exponentially, so every solution is not only stable, but asymptotically stable
- For complex λ
  - $\mathrm{Re}(\lambda) > 0$: unstable
  - $\mathrm{Re}(\lambda) < 0$: asymptotically stable
  - $\mathrm{Re}(\lambda) = 0$: stable but not asymptotically stable

Stability Part5

Linear, homogeneous system of ODEs with constant coefficients has form

$$\boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y}$$

where $\boldsymbol{A}$ is $n \times n$ matrix, and initial condition is $\boldsymbol{y}(0) = \boldsymbol{y}_0$

Suppose $\boldsymbol{A}$ is diagonalizable, with eigenvalues λ_i and corresponding eigenvectors $\boldsymbol{v}_i$, $i = 1, \ldots, n$

Express $\boldsymbol{y}_0$ as linear combination $\boldsymbol{y}_0 = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i$

Then

$$\boldsymbol{y}(t) = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i e^{\lambda_i t}$$

is solution to ODE satisfying initial condition $\boldsymbol{y}(0) = \boldsymbol{y}_0$

Stability Part6

Linear, homogeneous system of ODEs with constant coefficients has form

$$\boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y}$$

where $\boldsymbol{A}$ is $n \times n$ matrix, and initial condition is $\boldsymbol{y}(0) = \boldsymbol{y}_0$

Suppose $\boldsymbol{A}$ is diagonalizable, with eigenvalues λ_i and corresponding eigenvectors $\boldsymbol{v}_i$, $i = 1, \ldots, n$

Express $\boldsymbol{y}_0$ as linear combination $\boldsymbol{y}_0 = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i$

Then

$$\boldsymbol{y}(t) = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i e^{\lambda_i t}$$

is solution to ODE satisfying initial condition $\boldsymbol{y}(0) = \boldsymbol{y}_0$

Eigenvalues of $\boldsymbol{A}$ with *positive* real parts yield exponentially *growing* solution components

Eigenvalues with *negative* real parts yield exponentially *decaying* solution components

Eigenvalues with *zero* real parts (i.e., pure imaginary) yield *oscillatory* solution components

Solutions stable if $\mathrm{Re}(\lambda_i) \leq 0$ for every eigenvalue, and asymptotically stable if $\mathrm{Re}(\lambda_i) < 0$ for every eigenvalue, but unstable if $\mathrm{Re}(\lambda_i) > 0$ for any eigenvalue

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Accuracy
- Stability
- Implicit methods
- Trapezoid method
- Runge Kutta

Numerical approach Part1

Analytical solution of ODE is closed-form formula that can be evaluated at any point t

Numerical solution of ODE is table of approximate values of solution function at discrete set of points

Numerical solution is generated by simulating behavior of system governed by ODE

Starting at t_0 with given initial value $\boldsymbol{y}_0$, we track trajectory dictated by ODE

Evaluating $\boldsymbol{f}(t_0, \boldsymbol{y}_0)$ tells us slope of trajectory at that point

We use this information to predict value $\boldsymbol{y}_1$ of solution at future time $t_1 = t_0 + h$ for some suitably chosen time increment h

Numerical approach Part2

Approximate solution values are generated step by step in increments moving across interval in which solution is sought

In stepping from one discrete point to next, we incur some error, which means that next approximate solution value lies on *different* solution from one we started on

Stability or instability of solutions determines, in part, whether such errors are magnified or diminished with time

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Errors, stability, etc
- Implicit methods
- Trapezoid method
- Runge Kutta

Euler's method Part1

For general system of ODEs $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$, consider Taylor series

$$\begin{aligned} \boldsymbol{y}(t+h) &= \boldsymbol{y}(t) + h\boldsymbol{y}'(t) + \frac{h^2}{2}\boldsymbol{y}''(t) + \cdots \\ &= \boldsymbol{y}(t) + h\boldsymbol{f}(t, \boldsymbol{y}(t)) + \frac{h^2}{2}\boldsymbol{y}''(t) + \cdots \end{aligned}$$

Euler's method results from dropping terms of second and higher order to obtain approximate solution value

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h_k \boldsymbol{f}(t_k, \boldsymbol{y}_k)$$

Euler's method advances solution by extrapolating along straight line whose slope is given by $\boldsymbol{f}(t_k, \boldsymbol{y}_k)$

Euler's method is *single-step* method because it depends on information at only one point in time to advance to next point

Euler's method Part2

Applying Euler's method to ODE $y' = y$ with step size h, we advance solution from time $t_0 = 0$ to time $t_1 = t_0 + h$

$$y_1 = y_0 + hy_0' = y_0 + hy_0 = (1 + h)y_0$$

Value for solution we obtain at t_1 is not exact, $y_1 \neq y(t_1)$

For example, if $t_0 = 0$, $y_0 = 1$, and $h = 0.5$, then $y_1 = 1.5$, whereas exact solution for this initial value is $y(0.5) = \exp(0.5) \approx 1.649$

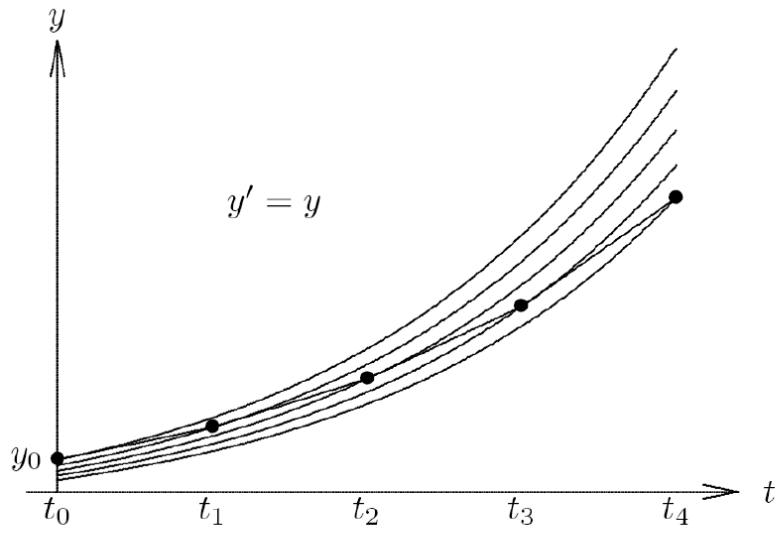
Thus, y_1 lies on different solution from one we started on

To continue numerical solution process, we take another step from t_1 to $t_2 = t_1 + h = 1.0$, obtaining
$y_2 = y_1 + hy_1 = 1.5 + (0.5)(1.5) = 2.25$

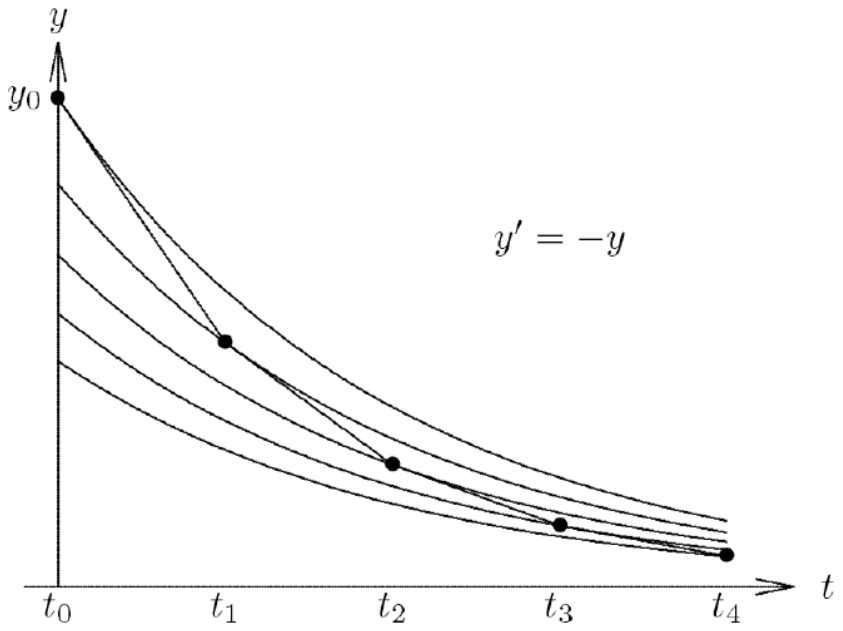
Now y_2 differs not only from true solution of original problem at $t = 1$, $y(1) = \exp(1) \approx 2.718$, but it also differs from solution through previous point (t_1, y_1), which has approximate value 2.473 at $t = 1$

Thus, we have moved to still another solution for this ODE

Euler's method Part3



For unstable solutions, errors in numerical solution grow with time



For stable solutions, errors in numerical solution may diminish with time

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Errors, stability, etc
- Implicit methods
- Trapezoid method
- Runge Kutta

Errors, stability, etc Part1

- Numerical methods for solving ODEs incur two distinct types of error
  - *Rounding error*, which is due to finite precision of floating-point arithmetic
  - *Truncation error* (*discretization error*), which is due to approximation method used and would remain even if all arithmetic were exact
- In practice, truncation error is dominant factor determining accuracy of numerical solutions of ODEs, so we will henceforth ignore rounding error

Errors, stability, etc Part2

Truncation error at any point t_k can be broken down into

- *Global error*: difference between computed solution and true solution $\boldsymbol{y}(t)$ passing through initial point $(t_0, \boldsymbol{y}_0)$

$$\boldsymbol{e}_k = \boldsymbol{y}_k - \boldsymbol{y}(t_k)$$

- *Local error*: error made in one step of numerical method

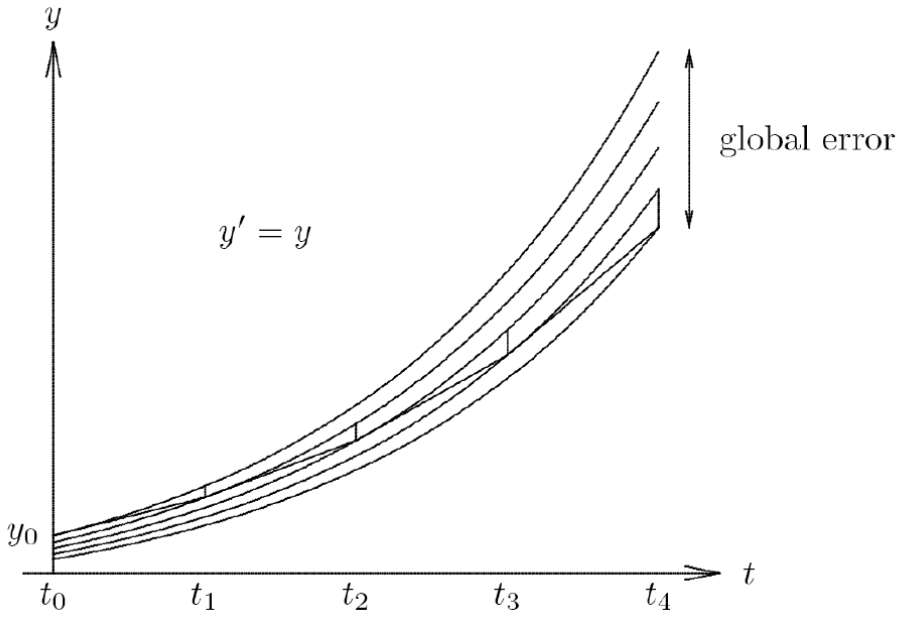
$$\ell_k = \boldsymbol{y}_k - \boldsymbol{u}_{k-1}(t_k)$$

  where $\boldsymbol{u}_{k-1}(t)$ is true solution passing through previous point $(t_{k-1}, \boldsymbol{y}_{k-1})$

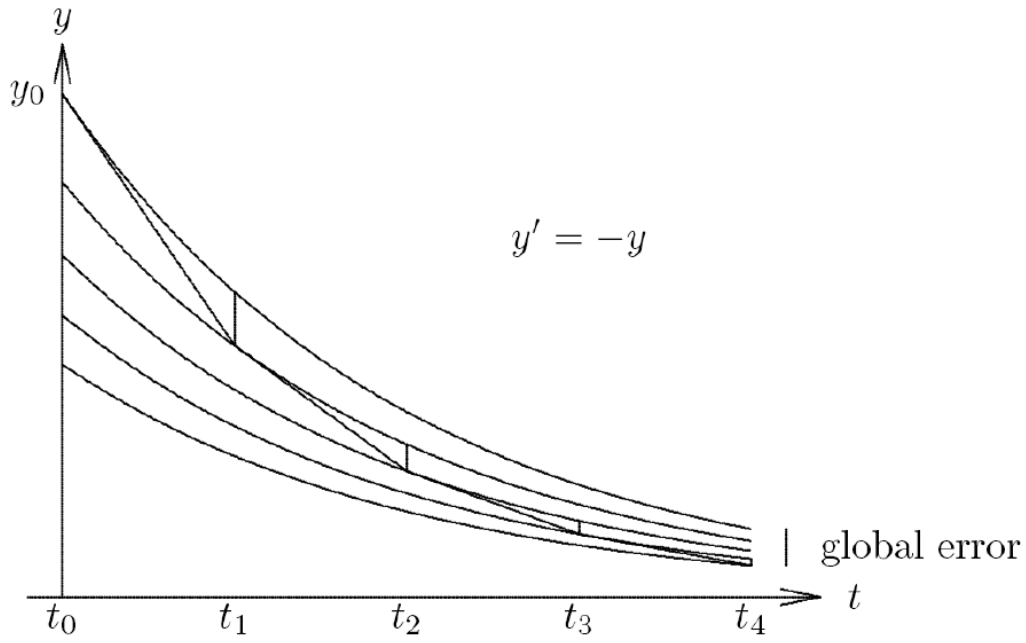
- Global error is not necessarily sum of local errors
- Global error is generally greater than sum of local errors if solutions are unstable, but may be less than sum if solutions are stable
- We want small global error, but we can control only local error directly

Errors, stability, etc Part3

Unstable Solutions



Stable Solutions



Errors, stability, etc Part4

Order of accuracy of numerical method is p if

$$\ell_k = \mathcal{O}(h_k^{p+1})$$

Then local error per unit step, $\ell_k / h_k = \mathcal{O}(h_k^p)$

Under reasonable conditions, $\boldsymbol{e}_k = \mathcal{O}(h^p)$, where h is average step size

Numerical method is *stable* if small perturbations do *not* cause resulting numerical solutions to diverge from each other without bound

Such divergence of numerical solutions could be caused by instability of solutions to ODE, but can also be due to numerical method itself, even when solutions to ODE are stable

Errors, stability, etc Part5

Determining Stability and Accuracy

Simple approach to determining stability and accuracy of numerical method is to apply it to scalar ODE $y' = \lambda y$, where λ is (possibly complex) constant

Exact solution is given by $y(t) = y_0 e^{\lambda t}$, where $y(0) = y_0$ is initial condition

Determine stability of numerical method by characterizing growth of numerical solution

Determine accuracy of numerical method by comparing exact and numerical solutions

Errors, stability, etc Part6

Determining Stability and Accuracy

Applying Euler's method to $y' = \lambda y$ using fixed step size h,

$$y_{k+1} = y_k + h\lambda y_k = (1 + h\lambda)y_k$$

which means that

$$y_k = (1 + h\lambda)^k y_0$$

If $\mathrm{Re}(\lambda) < 0$, exact solution decays to zero as t increases, as does computed solution if

$$|1 + h\lambda| < 1$$

which holds if $h\lambda$ lies inside circle in complex plane of radius 1 centered at -1

If λ is real, then $h\lambda$ must lie in interval $(-2, 0)$, so for $\lambda < 0$, we must have

$$h \leq -\frac{2}{\lambda}$$

for Euler's method to be stable

Growth factor $1 + h\lambda$ agrees with series expansion

$$e^{h\lambda} = 1 + h\lambda + \frac{(h\lambda)^2}{2} + \frac{(h\lambda)^3}{6} + \cdots$$

through terms of first order in h, so Euler's method is first-order accurate

Errors, stability, etc Part7

Determining Stability and Accuracy

For general system of ODEs $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$, consider Taylor series

$$\begin{aligned} \boldsymbol{y}(t+h) &= \boldsymbol{y}(t) + h\boldsymbol{y}'(t) + \mathcal{O}(h^2) \\ &= \boldsymbol{y}(t) + h\boldsymbol{f}(t, \boldsymbol{y}(t)) + \mathcal{O}(h^2) \end{aligned}$$

If we take $t = t_k$ and $h = h_k$, we obtain

$$\boldsymbol{y}(t_{k+1}) = \boldsymbol{y}(t_k) + h_k \boldsymbol{f}(t_k, \boldsymbol{y}(t_k)) + \mathcal{O}(h_k^2)$$

Subtracting this from Euler's method,

$$\begin{aligned} \boldsymbol{e}_{k+1} &= \boldsymbol{y}_{k+1} - \boldsymbol{y}(t_{k+1}) \\ &= [\boldsymbol{y}_k - \boldsymbol{y}(t_k)] + h_k[\boldsymbol{f}(t_k, \boldsymbol{y}_k) - \boldsymbol{f}(t_k, \boldsymbol{y}(t_k))] - \mathcal{O}(h_k^2) \end{aligned}$$

If there were no prior errors, then $\boldsymbol{y}_k = \boldsymbol{y}(t_k)$, and differences in brackets on right side would be zero, leaving only $\mathcal{O}(h_k^2)$ term, which is local error

So Euler's method is first-order accurate

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Errors, stability, etc
- Implicit methods
- Trapezoid method
- Runge Kutta

Implicit methods Part1

Euler's method is *explicit* in that it uses only information at time t_k to advance solution to time t_{k+1}

This may seem desirable, but Euler's method has rather limited stability region

Larger stability region can be obtained by using information at time t_{k+1}, which makes method *implicit*

Simplest example is *backward Euler method*

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h_k \boldsymbol{f}(t_{k+1}, \boldsymbol{y}_{k+1})$$

Method is implicit because we must evaluate $\boldsymbol{f}$ with argument $\boldsymbol{y}_{k+1}$ before we know its value

Implicit methods Part2

Euler's method is *explicit* in that it uses only information at time t_k to advance solution to time t_{k+1}

This may seem desirable, but Euler's method has rather limited stability region

Larger stability region can be obtained by using information at time t_{k+1}, which makes method *implicit*

Simplest example is *backward Euler method*

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h_k \boldsymbol{f}(t_{k+1}, \boldsymbol{y}_{k+1})$$

Method is implicit because we must evaluate $\boldsymbol{f}$ with argument $\boldsymbol{y}_{k+1}$ before we know its value

This means that we must solve algebraic equation to determine $\boldsymbol{y}_{k+1}$

Typically, we use iterative method such as Newton's method or fixed-point iteration to solve for $\boldsymbol{y}_{k+1}$

Good starting guess for iteration can be obtained from explicit method, such as Euler's method, or from solution at previous time step

Implicit methods Part3

Backward Euler Method

Consider nonlinear scalar ODE $y' = -y^3$ with initial condition $y(0) = 1$

Using backward Euler method with step size $h = 0.5$, we obtain implicit equation

$$y_1 = y_0 + hf(t_1, y_1) = 1 - 0.5y_1^3$$

for solution value at next step

This nonlinear equation for y_1 could be solved by fixed-point iteration or Newton's method

To obtain starting guess for y_1, we could use previous solution value, $y_0 = 1$, or we could use explicit method, such as Euler's method, which gives $y_1 = y_0 - 0.5y_0^3 = 0.5$

Iterations eventually converge to final value $y_1 \approx 0.7709$

Implicit methods Part4

Backward Euler Method

Given extra trouble and computation in using implicit method, one might wonder why we bother

Answer is that implicit methods generally have significantly larger stability region than comparable explicit methods

Implicit methods Part5

Backward Euler Method

To determine stability of backward Euler, we apply it to scalar ODE $y' = \lambda y$, obtaining

$$y_{k+1} = y_k + h\lambda y_{k+1}$$

$$(1 - h\lambda)y_{k+1} = y_k$$

$$y_k = \left(\frac{1}{1 - h\lambda}\right)^k y_0$$

Thus, for backward Euler to be stable we must have

$$\left|\frac{1}{1 - h\,\lambda}\right| \leq 1$$

which holds for *any* $h > 0$ when $\mathrm{Re}(\lambda) < 0$

So stability region for backward Euler method includes entire left half of complex plane, or interval $(-\infty, 0)$ if λ is real

Implicit methods Part6

Backward Euler Method

To determine stability of backward Euler, we apply it to scalar ODE $y' = \lambda y$, obtaining

$$y_{k+1} = y_k + h\lambda y_{k+1}$$

$$(1 - h\lambda)y_{k+1} = y_k$$

$$y_k = \left(\frac{1}{1 - h\lambda}\right)^k y_0$$

Thus, for backward Euler to be stable we must have

$$\left|\frac{1}{1 - h\,\lambda}\right| \leq 1$$

which holds for *any* $h > 0$ when $\mathrm{Re}(\lambda) < 0$

So stability region for backward Euler method includes entire left half of complex plane, or interval $(-\infty, 0)$ if λ is real

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Errors, stability, etc
- Implicit methods
- Trapezoid method
- Runge Kutta

Trapezoid method Part1

Higher-order accuracy can be achieved by averaging Euler and backward Euler methods to obtain implicit *trapezoid method*

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h_k \left(\boldsymbol{f}(t_k, \boldsymbol{y}_k) + \boldsymbol{f}(t_{k+1}, \boldsymbol{y}_{k+1})\right)/2$$

To determine its stability and accuracy, we apply it to scalar ODE $y' = \lambda y$, obtaining

$$y_{k+1} = y_k + h\left(\lambda y_k + \lambda y_{k+1}\right)/2$$

$$y_k = \left(\frac{1 + h\lambda/2}{1 - h\lambda/2}\right)^k y_0$$

Method is stable if

$$\left|\frac{1 + h\lambda/2}{1 - h\lambda/2}\right| < 1$$

which holds for any $h > 0$ when $\mathrm{Re}(\lambda) < 0$

Trapezoid method Part2

Thus, trapezoid method is unconditionally stable

Its growth factor

$$\begin{aligned}\frac{1+h\lambda/2}{1-h\lambda/2} &= \left(1+\frac{h\lambda}{2}\right)\left(1+\frac{h\lambda}{2}+\left(\frac{h\lambda}{2}\right)^2+\left(\frac{h\lambda}{2}\right)^3+\cdots\right) \\ &= 1+h\lambda+\frac{(h\lambda)^2}{2}+\frac{(h\lambda)^3}{4}+\cdots\end{aligned}$$

agrees with expansion of $e^{h\lambda}$ through terms of order h^2, so trapezoid method is second-order accurate

Implicit methods generally have larger stability regions than explicit methods, but allowable step size is not always unlimited

Implicitness alone is not sufficient to guarantee stability

Outline

- Ordinary diff. eq.
- Initial value problem
- Stability
- Numerical approach
- Euler’s method
- Errors, stability, etc
- Implicit methods
- Trapezoid method
- Runge Kutta

Runge-Kutta Part1

Euler's method can be derived from Taylor series expansion

By retaining more terms in Taylor series, we can generate single-step methods of higher-order

For example, retaining one additional term in Taylor series

$$\boldsymbol{y}(t+h) = \boldsymbol{y}(t) + h\,\boldsymbol{y}'(t) + \frac{h^2}{2}\,\boldsymbol{y}''(t) + \frac{h^3}{6}\,\boldsymbol{y}'''(t) + \cdots$$

gives second-order method

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h_k\,\boldsymbol{y}'_k + \frac{h_k^2}{2}\,\boldsymbol{y}''_k$$

Runge-Kutta Part2

This approach requires computation of higher derivatives of $\boldsymbol{y}$, which can be obtained by differentiating $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ using chain rule, e.g.,

$$\begin{aligned} \boldsymbol{y}'' &= \boldsymbol{f}_t(t, \boldsymbol{y}) + \boldsymbol{f}_y(t, \boldsymbol{y})\, \boldsymbol{y}' \\ &= \boldsymbol{f}_t(t, \boldsymbol{y}) + \boldsymbol{f}_y(t, \boldsymbol{y})\, \boldsymbol{f}(t, \boldsymbol{y}) \end{aligned}$$

where subscripts indicate partial derivatives with respect to given variable

As order increases, expressions for derivatives rapidly become too complicated to be practical to compute, so Taylor series methods of higher order have not often been used in practice

Runge-Kutta Part3

Runge-Kutta methods are single-step methods similar in motivation to Taylor series methods, but they do not require computation of higher derivatives

Instead, Runge-Kutta methods simulate effect of higher derivatives by evaluating $\boldsymbol{f}$ several times between t_k and t_{k+1}

Runge-Kutta Part4

Simplest example is second-order *Heun's method*

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h_k}{2}(\boldsymbol{k}_1 + \boldsymbol{k}_2)$$

where

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(t_k, \boldsymbol{y}_k) \\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_k + h_k, \boldsymbol{y}_k + h_k \boldsymbol{k}_1) \end{aligned}$$

Heun's method is analogous to implicit trapezoid method, but remains explicit by using Euler prediction $\boldsymbol{y}_k + h_k \boldsymbol{k}_1$ instead of $\boldsymbol{y}(t_{k+1})$ in evaluating $\boldsymbol{f}$ at t_{k+1}

Runge-Kutta Part5

Best-known Runge-Kutta method is classical fourth-order scheme

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h_k}{6}\left(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\right)$$

where

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(t_k, \boldsymbol{y}_k) \\
\boldsymbol{k}_2 &= \boldsymbol{f}(t_k + h_k/2, \boldsymbol{y}_k + (h_k/2)\boldsymbol{k}_1) \\
\boldsymbol{k}_3 &= \boldsymbol{f}(t_k + h_k/2, \boldsymbol{y}_k + (h_k/2)\boldsymbol{k}_2) \\
\boldsymbol{k}_4 &= \boldsymbol{f}(t_k + h_k, \boldsymbol{y}_k + h_k\boldsymbol{k}_3)
\end{aligned}$$

Runge-Kutta Part6

Advantages of classical Runge-Kutta methods

- No history of solution prior to time t_k required to proceed to time t_{k+1}
  - *Self-starting* at beginning of integration
  - Easy to change step size during integration
- Relatively easy to program, which accounts in part for their popularity

Disadvantages of classical Runge-Kutta methods

- Provide no error estimate on which to base choice of step size

Runge-Kutta Part7

Fehlberg devised *embedded Runge-Kutta* method that uses six function evaluations per step to produce both fifth-order and fourth-order estimates of solution, whose difference provides estimate for local error

Another embedded Runge-Kutta method is due to Dormand and Prince

This approach has led to automatic Runge-Kutta solvers that are effective for many problems, but which are still relatively inefficient for stiff problems or when very high accuracy is required

It is possible, however, to derive *implicit* Runge-Kutta methods with superior stability properties that are suitable for solving stiff ODEs

The final cut

Any questions