Lecture Notes to Accompany

Scientific Computing

An Introductory Survey

Second Edition

by Michael T. Heath

Chapter 9

Initial Value Problems for Ordinary Differential Equations

Differential Equations

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 *function* in infinite-dimensional space

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

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

Higher-Order ODEs

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

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

Given k-th order ODE

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

define k new unknowns

$$\begin{aligned} u_1(t) &= y, \\ u_2(t) &= y', \\ &\vdots \\ u_k(t) &= y^{(k-1)} \end{aligned}$$

Higher-Order ODEs, continued

Original ODE equivalent to first-order system

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

We restrict our attention to first-order ODEs in discussing numerical solution methods

Most ODE software designed to solve only first-order equations

Example: Newton's Second Law

Newton's Second Law of Motion, $F = ma$, is second-order ODE, since acceleration a is second derivative of position coordinate, denoted 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

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

Example Continued

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

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

In addition, we also get second component u_2, which is velocity y'

Ordinary Differential Equations

General first-order system of ODEs has form

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

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

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_n' \end{bmatrix} = \begin{bmatrix} dy_1/dt \\ dy_2/dt \\ \vdots \\ dy_n/dt \end{bmatrix}$$

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, $n = 1$

Initial Value Problems

ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ does not by itself 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

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

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

Thus, part of given problem data is requirement that

$$\boldsymbol{y}(t_0) = \boldsymbol{y}_0$$

Initial Value Problems, continued

This requirement determines unique solution to ODE, provided $\boldsymbol{f}$ is smooth

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

Example: Initial Value Problem

Consider scalar ODE

$$y' = y$$

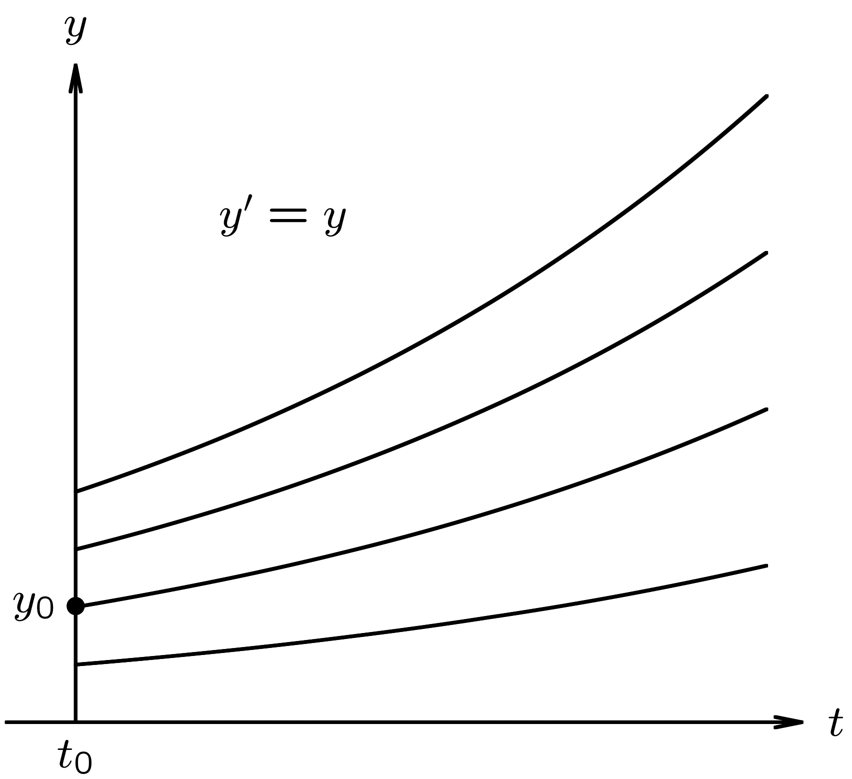
Family of solutions is given by $y = ce^t$, where c is any real constant

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 $y(t) = y_0 e^t$

Example: Initial Value Problem

Family of solutions for ODE $y' = y$



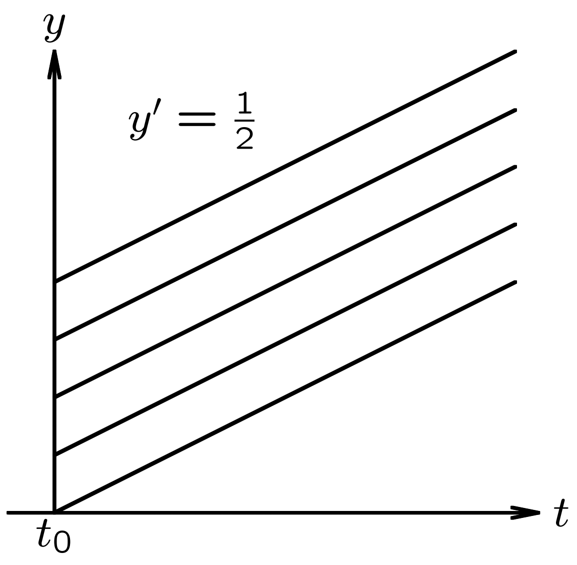
Stability of Solutions

Solution of ODE is

- *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

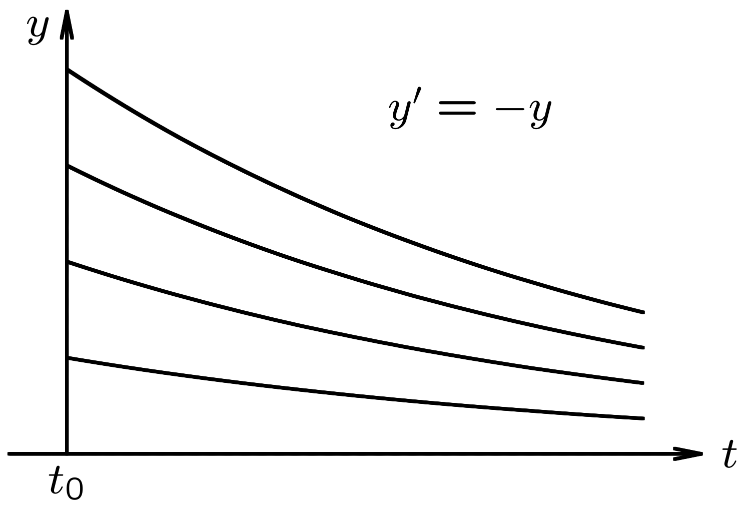
Example: Stable Solutions

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



Example: Asymptotically Stable Solutions

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



Example: Stability of Solutions

Consider scalar ODE

$$y' = \lambda y,$$

where λ is constant.

Solution given by

$$y(t) = y_0 \, e^{\lambda t},$$

where $t_0 = 0$ is initial time and $y(0) = y_0$ is initial value

If $\lambda > 0$, then all nonzero solutions grow exponentially, so every solution is unstable

If $\lambda < 0$, then all nonzero solutions decay exponentially, so every solution is not only stable, but asymptotically stable

If λ is complex, then solutions are unstable if $\mathrm{Re}(\lambda) > 0$, asymptotically stable if $\mathrm{Re}(\lambda) < 0$, and stable but not asymptotically stable if $\mathrm{Re}(\lambda) = 0$

Example: Linear System of ODEs

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

Example Continued

- 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 $\text{Re}(\lambda_i) \leq 0$ for every eigenvalue, and asymptotically stable if $\text{Re}(\lambda_i) < 0$ for every eigenvalue, but unstable if $\text{Re}(\lambda_i) > 0$ for some eigenvalue

Stability of Solutions, cont.

For general nonlinear ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$, determining stability of solutions is more complicated

ODE can be linearized locally about solution $\boldsymbol{y}(t)$ by truncated Taylor series, yielding linear ODE

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

where $\boldsymbol{J_f}$ is Jacobian matrix of $\boldsymbol{f}$ with respect to $\boldsymbol{y}$

Eigenvalues of $\boldsymbol{J_f}$ determine stability locally but conclusions drawn may not be valid globally

Numerical Solution of ODEs

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 increment h

Numerical Solution of ODEs, cont.

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

Euler's Method

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

Example: Euler's Method

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

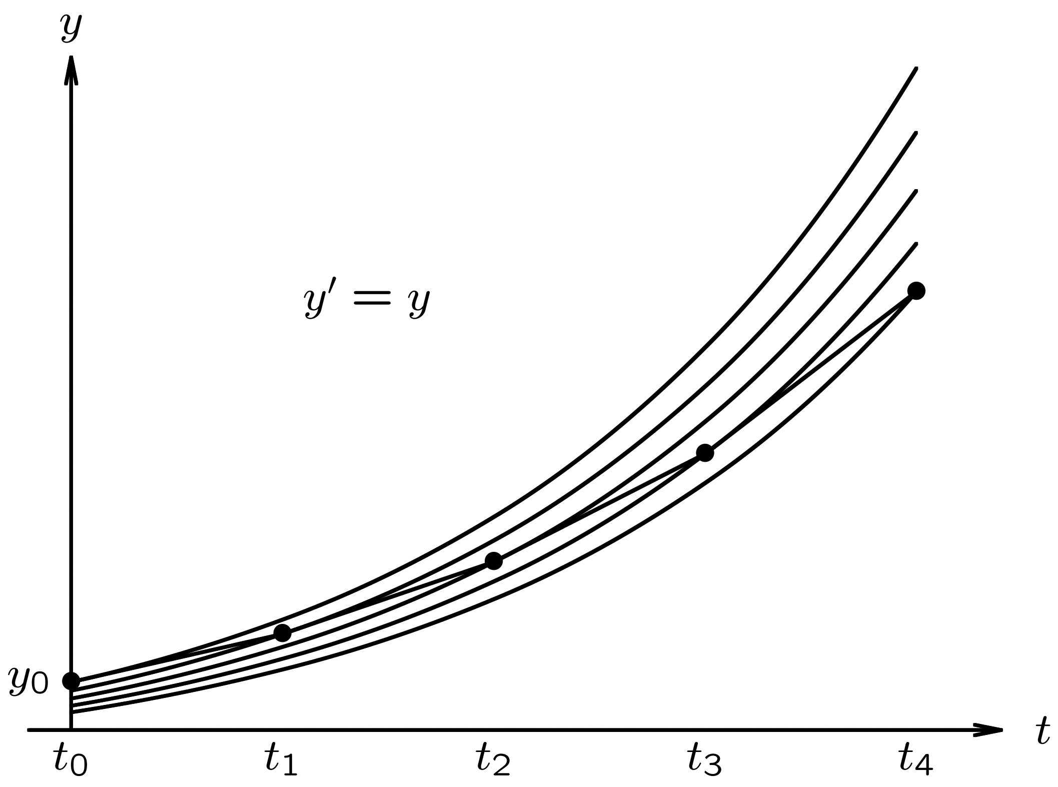
Example Continued

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

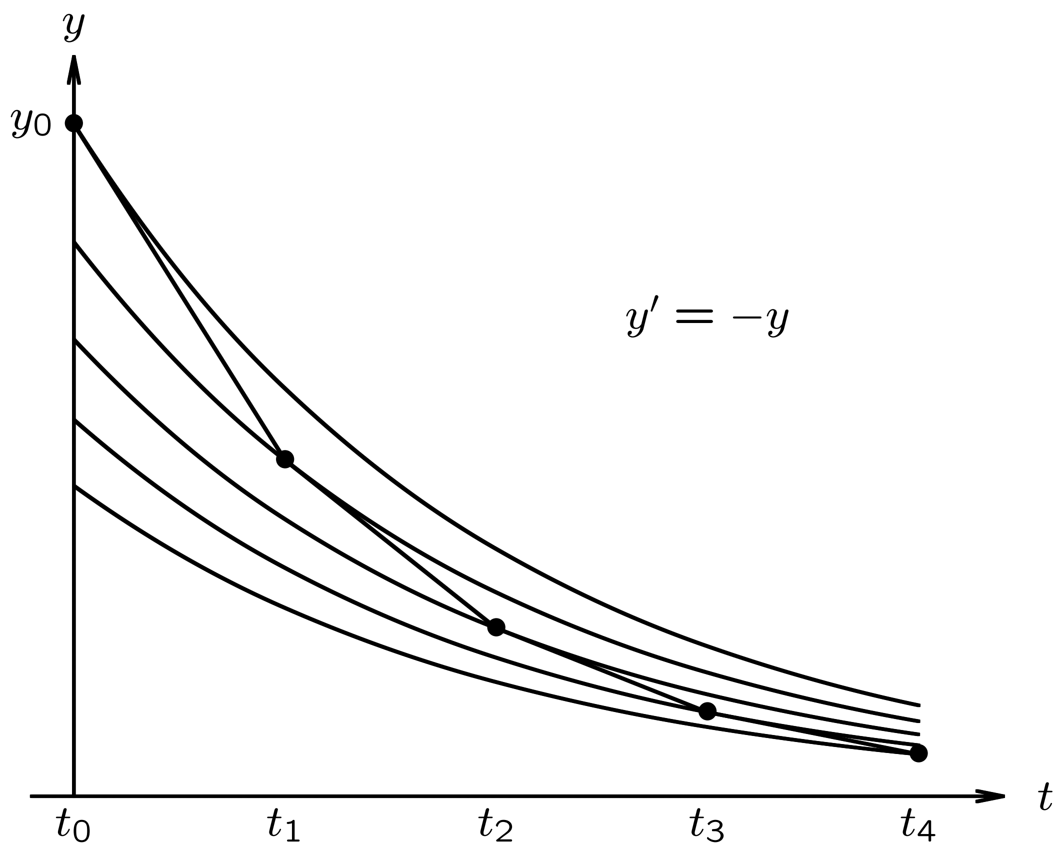
Example Continued



For unstable solutions, errors in numerical solution grow with time

Example Continued

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



Errors in Numerical Solution of ODEs

Numerical methods for solving ODEs suffer from two distinct sources of error:

- *Rounding* error, which is due to finite precision of floating-point arithmetic
- *Truncation* (or discretization) error, which is due to 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, and we shall henceforth ignore rounding error

Global Error and Local Error

Truncation error can be broken down into:

- *Global* error, which is difference between computed solution and true solution determined by initial data at t_0:

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

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

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

  where $\boldsymbol{u}_{k-1}$ is solution through $(t_{k-1}, \boldsymbol{y}_{k-1})$

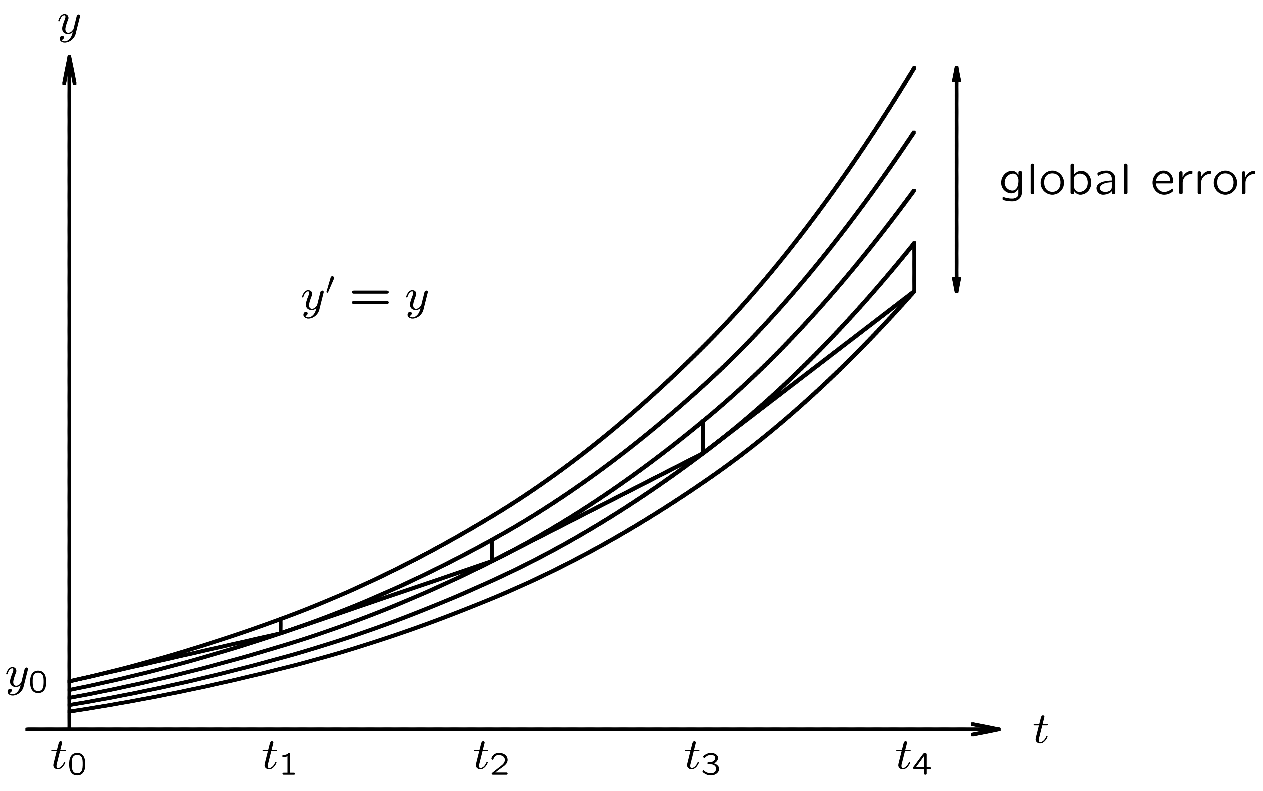
Global Error and Local Error, cont.

Global error is not necessarily sum of local errors

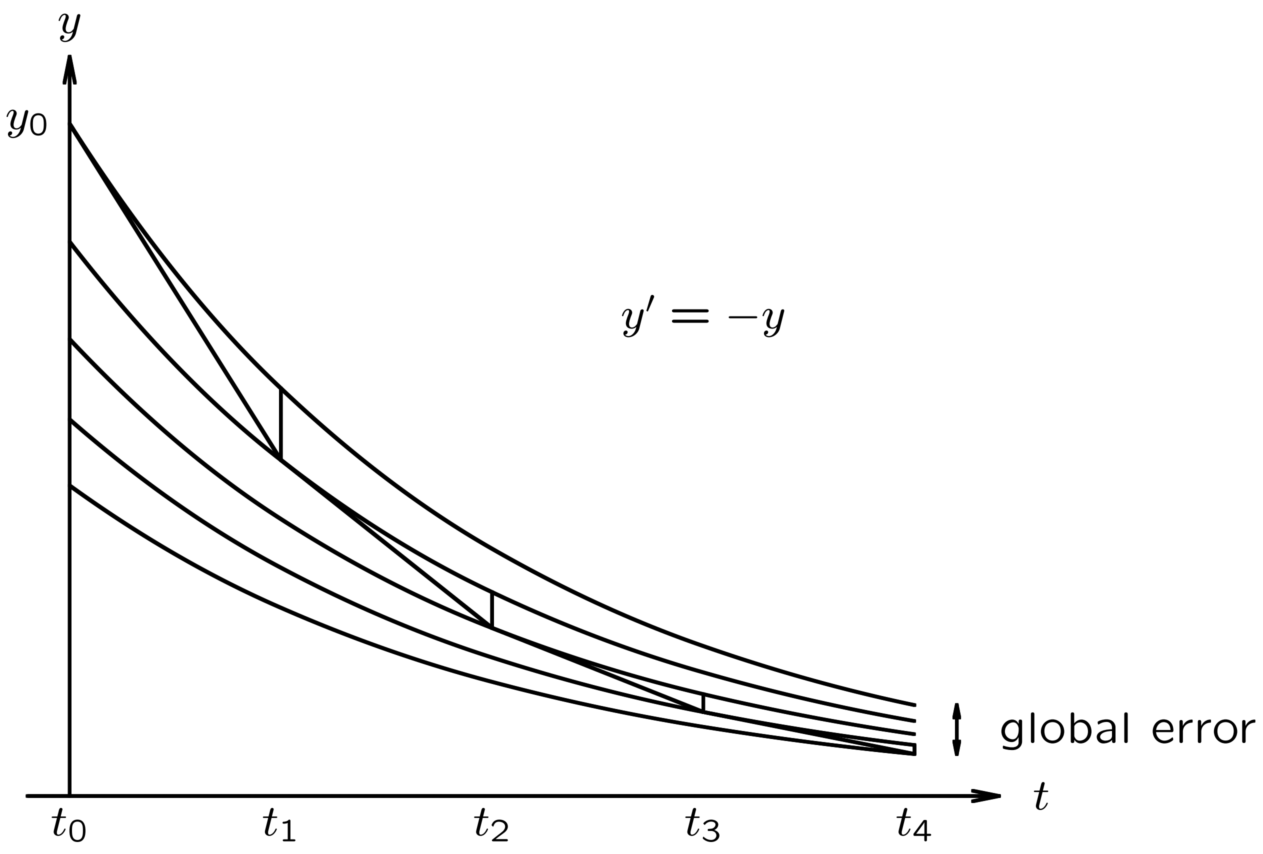
Global error generally greater than sum of local errors if solutions unstable, but may be less than sum if solutions stable

Having small global error is what we want, but we can control only local error directly

Global Error and Local Error, cont.



Global and Local Error, cont.



Accuracy and Stability of Numerical Methods for ODEs

Accuracy of numerical method is of *order* p if

$$\boldsymbol{\ell}_k = \mathcal{O}(h_k^{p+1})$$

Local error per unit step, $\boldsymbol{\ell}_k / h_k = \mathcal{O}(h_k^p)$

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 solution to ODE, but can also be due to numerical method itself, even when solutions to ODE are stable

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

For given numerical method, we can

- Determine stability by characterizing growth of numerical solution
- Determine accuracy by comparing exact and numerical solutions

Example: Euler's Method

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

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

which means that

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

If $\text{Re}(\lambda) < 0$, 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 $h \leq -2/\lambda$ for Euler's method to be stable

Euler's Method, continued

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

Euler's Method, continued

For general ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$, 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, we get

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

Euler's Method, continued

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

This means that Euler's method is first-order accurate

Euler's Method, continued

From previous derivation, global error is sum of local error and *propagated* error

From Mean Value Theorem, we have

$$\boldsymbol{f}(t_k, \boldsymbol{y}_k) - \boldsymbol{f}(t_k, \boldsymbol{y}(t_k)) = \boldsymbol{J}_f(t_k, \boldsymbol{\xi})(\boldsymbol{y}_k - \boldsymbol{y}(t_k))$$

for some (unknown) value $\boldsymbol{\xi}$, where $\boldsymbol{J}_f$ is Jacobian matrix of $\boldsymbol{f}$ with respect to $\boldsymbol{y}$

So we can express global error at step $k+1$ as

$$\boldsymbol{e}_{k+1} = (\boldsymbol{I} + h_k \boldsymbol{J}_f)\boldsymbol{e}_k + \boldsymbol{\ell}_{k+1}$$

Thus, global error is multiplied at each step by *growth factor* $\boldsymbol{I} + h_k \boldsymbol{J}_f$

Errors do not grow if spectral radius

$$\rho(\boldsymbol{I} + h_k \boldsymbol{J}_f) \leq 1,$$

which holds if all eigenvalues of $h_k \boldsymbol{J}_f$ lie inside circle in complex plane of radius 1 centered at -1

Stability of Numerical Methods for ODEs

In general, growth factor depends on

- Numerical method, which determines form of growth factor
- Step size h
- ODE, which determines Jacobian $\boldsymbol{J}_f$

Step Size Selection

In choosing step size for advancing numerical solution of ODE, want to take large steps to reduce computational cost, but must also take into account both stability and accuracy

To yield meaningful solution, step size must obey any stability restrictions

In addition, local error estimate is needed to ensure that desired accuracy is achieved

With Euler's method, for example, local error is approximately $(h_k^2/2)\boldsymbol{y}''$, so choose step size to satisfy

$$h_k \leq \sqrt{2\ tol/\|\boldsymbol{y}''\|}$$

Step Size Selection, continued

We do not know value of $\boldsymbol{y}''$, but we can estimate it by difference quotient

$$\boldsymbol{y}'' \approx \frac{\boldsymbol{y}'_k - \boldsymbol{y}'_{k-1}}{t_k - t_{k-1}}$$

Other methods of obtaining error estimates are based on difference between results obtained using methods of different orders or different step sizes

Implicit Methods

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, continued

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

Example: 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 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 an explicit method. Euler's method gives $y_1 = y_0 - 0.5y_0^3 = 0.5$

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

Implicit Methods, continued

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

Answer is that implicit methods generally have significantly larger stability region

Backward Euler Method

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

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

or

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

so that

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

Thus, for backward Euler to be stable we must have

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

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

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

Backward Euler Method, cont.

Growth factor

$$\frac{1}{1 - h\lambda} = 1 + h\lambda + (h\lambda)^2 + \cdots$$

agrees with expansion for $e^{\lambda h}$ through terms of order h, so backward Euler method is first-order accurate

Growth factor for backward Euler method for general ODE is $(\boldsymbol{I} - h\boldsymbol{J}_f)^{-1}$, whose spectral radius is less than 1 provided all eigenvalues of $h\boldsymbol{J}_f$ lie outside circle in complex plane of radius 1 centered at 1

Thus, stability region backward Euler is entire left half of complex plane for system of equations

For computing stable solution, backward Euler is stable for any positive step size, which means that it is *unconditionally* stable

Unconditionally Stable Methods

Great virtue of unconditionally stable method is that desired accuracy is only constraint on choice of step size

Thus, we may be able to take much larger steps than for explicit method of comparable order and attain much higher overall efficiency despite requiring more computation per step

Although backward Euler method is unconditionally stable, its accuracy is only of first order, which severely limits its usefulness

Trapezoid Method

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

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

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

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

which implies that

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

Method is stable if

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

which holds for any $h > 0$ when $\text{Re}(\lambda) < 0$, so trapezoid rule is unconditionally stable

Trapezoid Method, continued

Growth factor

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

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

More generally, trapezoid method has growth factor $(\boldsymbol{I}+\frac{1}{2}h\boldsymbol{J}_f)(\boldsymbol{I}-\frac{1}{2}h\boldsymbol{J}_f)^{-1}$, whose spectral radius is less than 1 provided eigenvalues of $h\boldsymbol{J}_f$ lie in left half of complex plane

Implicit Methods, continued

We have now seen two examples of implicit methods that are unconditionally stable, but not all implicit methods have this property

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

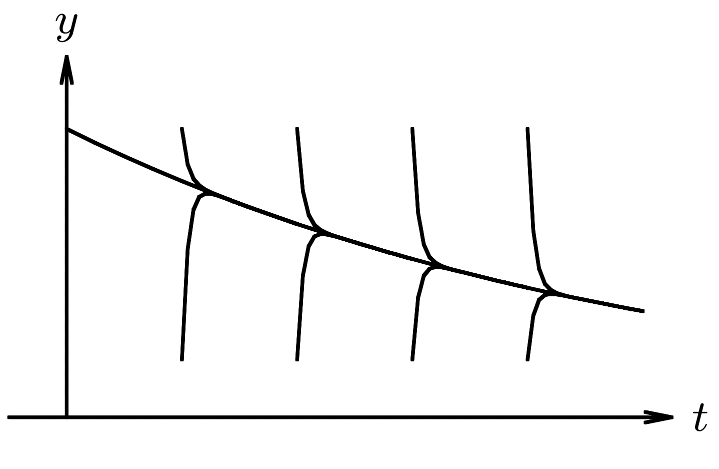
Implicitness alone is not sufficient to guarantee stability

Stiff Differential Equations

Asymptotically stable solutions converge with time, and this has favorable property of damping errors in numerical solution

But if convergence of solutions is too rapid, then difficulties of different type may arise

Such ODE is said to be *stiff*



Stiff ODEs, continued

Stiff ODE corresponds to physical process whose components have disparate time scales or whose time scale is small compared to interval over which it is studied

ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ is stiff if its Jacobian $\boldsymbol{J_f}$ has eigenvalues that differ greatly in magnitude

There may be eigenvalues with

- large negative real parts, corresponding to strongly damped components of solution, or
- large imaginary parts, corresponding to rapidly oscillating components of solution

Stiff ODEs, continued

Some numerical methods are inefficient for stiff ODEs because rapidly varying component of solution forces very small step sizes to maintain stability

Stability restriction depends on rapidly varying component of solution, but accuracy restriction depends on slowly varying component, so step size may be more severely restricted by stability than by required accuracy

For example, Euler's method is extremely inefficient for solving stiff ODEs because of severe stability limitation on step size

Backward Euler method is suitable for stiff ODEs because of its unconditional stability

Stiff ODEs need not be difficult to solve numerically, provided suitable method is chosen

Example: Stiff ODE

Consider scalar ODE

$$y' = -100y + 100t + 101$$

with initial condition $y(0) = 1$

General solution is $y(t) = 1 + t + ce^{-100t}$, and particular solution satisfying initial condition is $y(t) = 1 + t$ (i.e., $c = 0$)

Since solution is linear, Euler's method is theoretically exact for this problem

However, to illustrate effect of using finite precision arithmetic, let us perturb initial value slightly

Example Continued

With step size $h = 0.1$, first few steps for given initial values are:

t	0.0	0.1	0.2	0.3	0.4
exact sol.	1.00	1.10	1.20	1.30	1.40
Euler sol.	0.99	1.19	0.39	8.59	−64.2
Euler sol.	1.01	1.01	2.01	−5.99	67.0

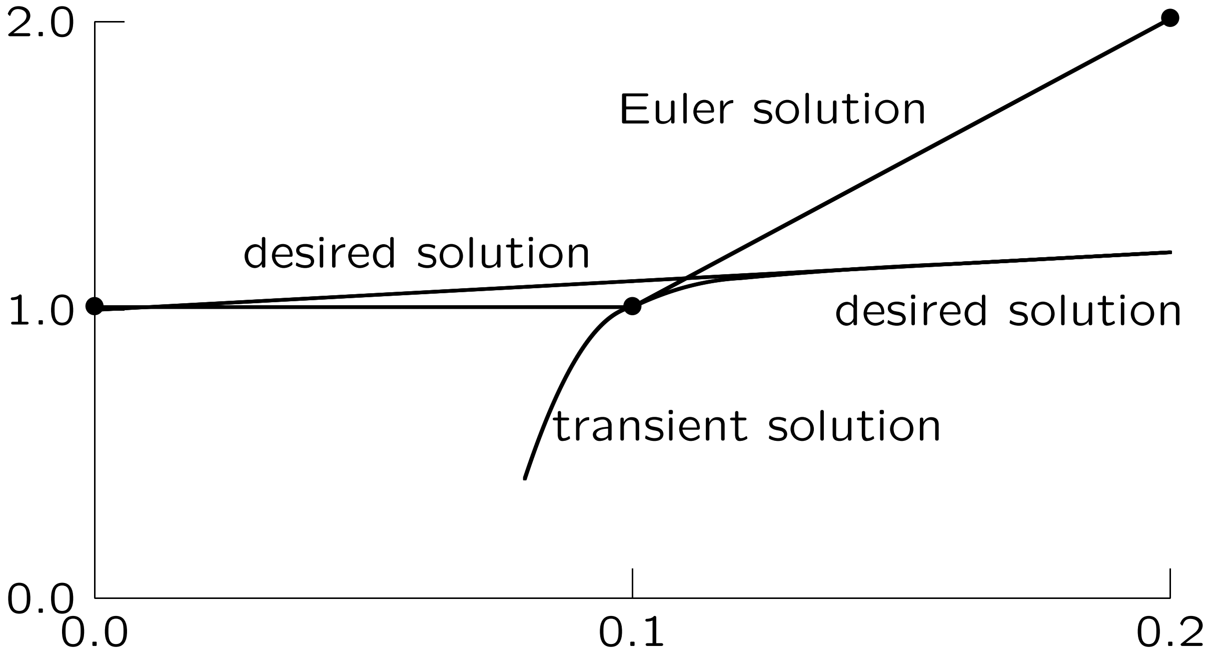
Computed solution is incredibly sensitive to initial value, as each tiny perturbation results in wildly different solution

Any point deviating from desired particular solution, even by only small amount, lies on different solution, for which $c \neq 0$, and therefore rapid transient of general solution is present

Example Continued

Euler's method bases its projection on derivative at current point, and resulting large value causes numerical solution to diverge radically from desired solution

Jacobian for this ODE is $J_f = -100$, so stability condition for Euler's method requires step size $h < 0.02$, which we are violating



Example Continued

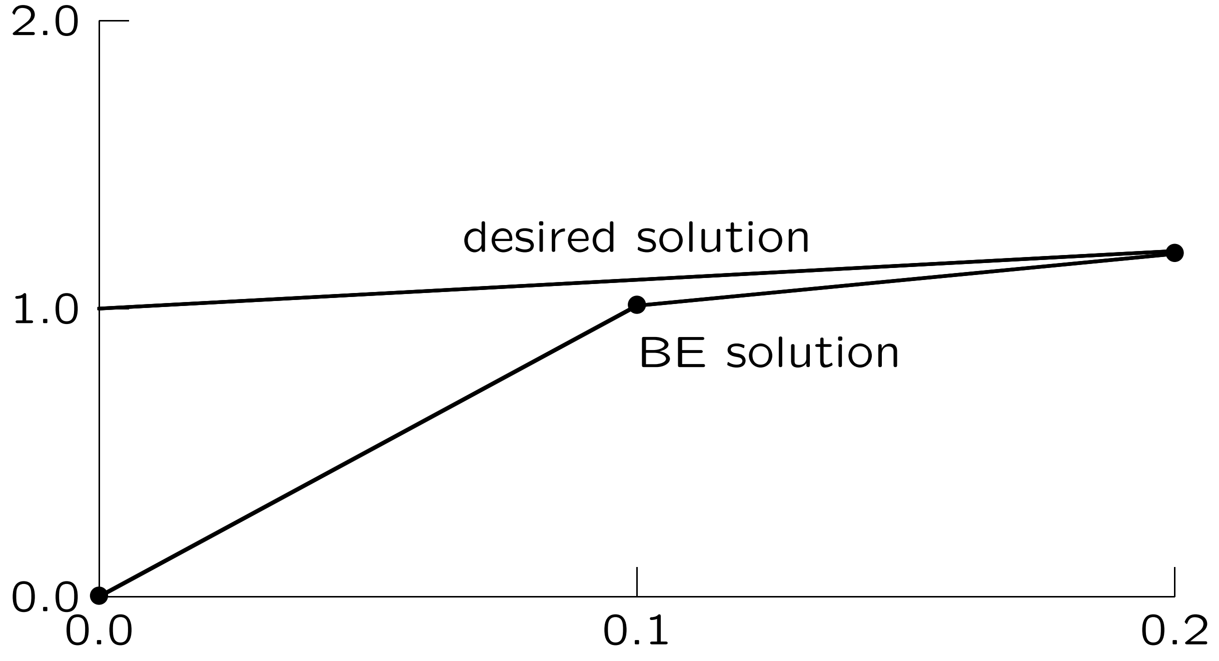
Backward Euler method has no trouble solving this problem and is extremely *insensitive* to initial value:

t	0.0	0.1	0.2	0.3	0.4
exact sol.	1.00	1.10	1.20	1.30	1.40
BE sol.	0.00	1.01	1.19	1.30	1.40
BE sol.	2.00	1.19	1.21	1.30	1.40

Even with very large perturbation in initial value, by using derivative at next point rather than current point, transient is quickly damped out and backward Euler solution converges to desired solution after only few steps

This behavior is consistent with unconditional stability of backward Euler method for stable solutions

Example Continued



Numerical Methods for ODEs

There are many different methods for solving ODEs Most of these are of one of following types:

- Taylor series
- Runge-Kutta
- Extrapolation
- Multistep
- Multivalue

We briefly consider each of these types of methods

Taylor Series Methods

Euler's method can be derived from Taylor series expansion

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

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

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

Taylor Series Methods, cont.

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 Methods

Runge-Kutta methods are single-step methods similar in motivation to Taylor series methods but 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}

Simplest example is second-order Runge-Kutta method

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

where

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

which is called *Heun's method*

Runge-Kutta Methods, continued

Heun's method is analogous to implicit trapezoid rule, 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}

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

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

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 Methods, continued

To proceed to time t_{k+1}, Runge-Kutta methods require no history of solution prior to time t_k, which makes them *self-starting* at beginning of integration, and also makes it easy to change step size during integration

These facts also make Runge-Kutta methods relatively easy to program, which accounts in part for their popularity

Unfortunately, classical Runge-Kutta methods provide no error estimate on which to base choice of step size

Runge-Kutta Methods, continued

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 relatively inefficient for stiff problems or when very high accuracy is required

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

Extrapolation Methods

Extrapolation methods are based on use of single-step method to integrate ODE over given interval, $t_k \le t \le t_{k+1}$, using several different step sizes h_i, and yielding results denoted by $\boldsymbol{Y}(h_i)$

This gives discrete approximation to function $\boldsymbol{Y}(h)$, where $\boldsymbol{Y}(0) = \boldsymbol{y}(t_{k+1})$

Interpolating polynomial or rational function $\hat{\boldsymbol{Y}}(h)$ is fit to these data, and $\hat{\boldsymbol{Y}}(0)$ is then taken as approximation to $\boldsymbol{Y}(0)$

Extrapolation methods are capable of achieving very high accuracy, but they are much less efficient and less flexible than other methods for ODEs, so they are not often used unless extremely high accuracy is required and cost is not significant factor

Multistep Methods

Multistep methods use information at more than one previous point to estimate solution at next point

Linear multistep methods have form

$$\boldsymbol{y}_{k+1} = \sum_{i=1}^{m} \alpha_i \boldsymbol{y}_{k+1-i} + h \sum_{i=0}^{m} \beta_i \boldsymbol{f}(t_{k+1-i}, \boldsymbol{y}_{k+1-i})$$

Parameters α_i and β_i are determined by polynomial interpolation

If $\beta_0 = 0$, method is explicit, but if $\beta_0 \neq 0$, method is implicit

Implicit methods are usually more accurate and stable than explicit methods, but require starting guess for $\boldsymbol{y}_{k+1}$

Multistep Methods, continued

Starting guess is conveniently supplied by explicit formula, so the two are used as *predictor-corrector* pair

One could use corrector repeatedly (i.e., fixed-point iteration) until some convergence tolerance is met, but it may not be worth expense

So fixed number of corrector steps, often only one, may be used instead, giving *PECE* scheme (predict, evaluate, correct, evaluate)

Although it has no effect on value of $\boldsymbol{y}_{k+1}$, second evaluation of $\boldsymbol{f}$ in PECE scheme yields improved value of $\boldsymbol{y}'_{k+1}$ for future use

Multistep Methods, continued

Alternatively, nonlinear equation for $\boldsymbol{y}_{k+1}$ given by implicit multistep method can be solved by Newton's method or similar method, again with starting guess supplied by solution at previous step or by explicit multistep method

Newton's method or close variant of it is essential when using an implicit multistep method designed for stiff ODEs, as fixed-point iteration fails to converge for reasonable step sizes

Examples: Multistep Methods

Simplest second-order accurate explicit two-step method is

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h}{2}(3\boldsymbol{y}'_k - \boldsymbol{y}'_{k-1})$$

Simplest second-order accurate implicit two-step method is trapezoid rule

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h}{2}(\boldsymbol{y}'_{k+1} + \boldsymbol{y}'_k)$$

One of most popular pairs of multistep methods is explicit fourth-order Adams-Bashforth predictor

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h}{24}(55\boldsymbol{y}'_k - 59\boldsymbol{y}'_{k-1} + 37\boldsymbol{y}'_{k-2} - 9\boldsymbol{y}'_{k-3})$$

and implicit fourth-order Adams-Moulton corrector

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h}{24}(9\boldsymbol{y}'_{k+1} + 19\boldsymbol{y}'_k - 5\boldsymbol{y}'_{k-1} + \boldsymbol{y}'_{k-2})$$

Examples: Multistep Methods

Backward differentiation formulas form another important family of implicit multistep methods

BDF methods, typified by formula

$$\boldsymbol{y}_{k+1} = \frac{1}{11}(18\boldsymbol{y}_k - 9\boldsymbol{y}_{k-1} + 2\boldsymbol{y}_{k-2}) + \frac{6h}{11}\boldsymbol{y}'_{k+1},$$

are effective for solving stiff ODEs

Multistep Methods, continued

Stability and accuracy of some popular multi-step Adams methods are summarized below

Explicit Methods

Order	Stability threshold	Error constant
1	-2	$1/2$
2	-1	$5/12$
3	$-6/11$	$3/8$
4	$-3/10$	$251/720$

Implicit Methods

Order	Stability threshold	Error constant
1	$-\infty$	$-1/2$
2	$-\infty$	$-1/12$
3	-6	$-1/24$
4	-3	$-19/720$

Implicit methods are both more stable and more accurate than corresponding explicit methods of same order

Properties of Multistep Methods

They are not self-starting, since several previous values of $\boldsymbol{y}_k$ are needed initially

Changing step size is complicated, since interpolation formulas are most conveniently based on equally spaced intervals for several consecutive points

Good local error estimate can be determined from difference between predictor and corrector

They are relatively complicated to program

Properties of Multistep, continued

Being based on interpolation, they can efficiently provide solution values at output points other than integration points

Implicit methods have much greater region of stability than explicit methods, but must be iterated to convergence to enjoy this benefit fully (e.g., PECE scheme is actually explicit, though in a somewhat complicated way)

Although implicit methods are more stable than explicit methods, they are still not necessarily unconditionally stable (no multistep method of greater than second order is unconditionally stable, even if it is implicit)

Properly designed implicit multistep method can be very effective for solving stiff ODEs

Multivalue Methods

With multistep methods it is difficult to change step size, since past history of solution is most easily maintained at equally spaced intervals

Like multistep methods, *multivalue methods* are also based on polynomial interpolation, but avoid some implementation difficulties associated with multistep methods

One key idea motivating multivalue methods is observation that interpolating polynomial itself can be evaluated at any point, not just at equally spaced intervals

Equal spacing associated with multistep methods is artifact of representation as linear combination of successive solution and derivative values with fixed weights

Multivalue Methods, continued

Another key idea in implementing multivalue methods is representing interpolating polynomial so that parameters are values of solution and its derivatives at t_k

This approach is analogous to using Taylor, rather than Lagrange, representation of polynomial

Solution is advanced in time by simple transformation of representation from one point to next, and it is easy to change step size

Multivalue methods are equivalent to multistep methods but are more convenient and flexible to implement, so most modern software implementations are based on them

Example: Multivalue Method

Consider four-value method for solving scalar ODE

$$y' = f(t, y)$$

Instead of representing interpolating polynomial by its value at four different points, we represent it by its value and first three derivatives at single point t_k:

$$\mathbf{y}_k = \begin{bmatrix} y_k \\ hy'_k \\ (h^2/2)y''_k \\ (h^3/6)y'''_k \end{bmatrix}$$

By differentiating Taylor series

$$y(t_k + h) = y(t_k) + hy' + \frac{h^2}{2}y'' + \frac{h^3}{6}y'''_k + \cdots$$

three times, we see that corresponding values at next point $t_{k+1} = t_k + h$ are given approximately by transformation

Example: Multivalue Method

$$\hat{\mathbf{y}}_{k+1} = \boldsymbol{B}\mathbf{y}_k,$$

where matrix $\boldsymbol{B}$ is given by

$$\boldsymbol{B} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We have not yet used ODE, however, so we add correction term to foregoing prediction to obtain final value

$$\mathbf{y}_{k+1} = \hat{\mathbf{y}}_{k+1} + \alpha\boldsymbol{r},$$

where $\boldsymbol{r}$ is fixed 4-vector and

$$\alpha = h(f(t_{k+1}, y_{k+1}) - \hat{y}'_{k+1})$$

Example Continued

For consistency, i.e., for ODE to be satisfied, we must have $r_2 = 1$

Remaining three components of $\boldsymbol{r}$ can be chosen in various ways, resulting in different methods, analogous to different choices of parameters in multistep methods

For example, four-value method with

$$\boldsymbol{r} = [\,\tfrac{3}{8} \quad 1 \quad \tfrac{3}{4} \quad \tfrac{1}{6}\,]^T$$

is equivalent to implicit fourth-order Adams-Moulton method given earlier

Multivalue Methods, continued

It is easy to change step size with multivalue method: we need merely rescale components of $\mathbf{y}_k$ to reflect new step size

It is also easy to change order of method simply by changing components of $\boldsymbol{r}$

These two capabilities, combined with sophisticated tests and strategies for deciding when to change order and step size, have led to development of powerful and efficient software packages for solving ODEs based on variable-order/variable-step methods

Variable-Order/Variable-Step Solvers

Such routines are analogous to adaptive quadrature routines: they automatically adapt to given problem, varying order and step size of integration method as necessary to meet user-supplied error tolerance efficiently

Such routines often have options for solving either stiff or nonstiff problems, and some even detect stiffness automatically and select appropriate method accordingly

Ability to change order easily also obviates need for special starting procedures: one can simply start with first-order method, which requires no additional starting values, and let automatic order/step size selection procedure increase order as needed for required accuracy