ODE – Initial Value Problems

IPCST
Seoul National University

Initial Value Problem

- $$\frac{dy(t)}{dt} = f(t, y(t))$$ with the initial condition $$y(t_0) = y_0$$
  $$\dot{y}(t) = f(t, y(t))$$

- Numerical solution
  $$y_n \approx y(t_n), \quad n = 0, 1, \ldots.$$
  - Time step-size $h = t_{n+1} - t_n$
    - Uniform → fixed step-size
    - Non-uniform → variable step-size

Systems of Equations

- Ex.) $$\ddot{x}(t) = -x(t)$$

$$y(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix} = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}$$

$$\dot{y}(t) = \begin{bmatrix} \dot{x}(t) \\ -x(t) \end{bmatrix} = \begin{bmatrix} y_2(t) \\ -y_1(t) \end{bmatrix}$$

- Most of numerical methods can treat only first-order ODEs.

❖ See also Wen Shen 9.7 & 9.8 for further study.

Single-step Methods

- Using only y_n to get y_{n+1}
  - Taylor series methods
    - Including the Forward Euler’s method
  - Euler’s method
  - Trapezoid rule (for ODE)
  - Runge-Kutta methods
    - Runge-Kutta methods of the wide meaning include the Euler’s method and the trapezoid rule.
    - Sometimes, implicit Runge-Kutta methods take multi-step. On the other hand, explicit Runge-Kutta methods are always single-step methods.

(Forward) Euler's Method

– Also known as Taylor series method of order 1

- $y_{n+1} = y_n + h \cdot f(t_n, y_n)$
- $t_{n+1} = t_n + h$

$$\dot{y}(t) = f(t, y(t))$$

- Total Error ~ $O(h)$
- Local Error ~ $O(h^2)$

❖ Cf.) Forward 2-point differentiation

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

Taylor Series Method of Order 2

- $y(t + h) \cong y(t) + h \cdot y'(t) + ½h^2 \cdot y''(t)$

- $s_1 = f(t_n, y_n)$
- $s_2 = df(t_n, y_n)/dt$
  $= \partial f(t_n, y_n)/\partial t + y'(t) \cdot \partial f(t_n, y_n)/\partial y$
  $= \partial f(t_n, y_n)/\partial t + f(t_n, y_n) \cdot \partial f(t_n, y_n)/\partial y$
- $y_{n+1} = y_n + s_1 \cdot h + ½s_2 \cdot h^2$
- $t_{n+1} = t_n + h$

- Total Error ~ $O(h^2)$

Errors

- Discretization error (truncation error)
  - property of the differential equation and the numerical method

- Roundoff error
  - property of the computer hardware and the program code
  - Related to floating-point arithmetic

Errors

- Discretization error
  - Local: error made at each step if the previous value is exact

    $O(h^{p+1})$ p: order of the method

  - Global: the maximum difference (in the interval) between the computed and the exact solution
    - A global error *may* be the sum of local errors or *not* (depending on $\partial f/\partial y$)
  - Total: the difference between the solutions at the end

    $O(h^p)$ or less

Errors

- Discretization error
  - Wen Shen pp. 179~180
  - Local error $e_n \doteq |y_{n+1} - y(t_n + h)|$
    - $y(t_n + h)$: exact on the assumption that $y(t_n)$ is exact

◆ Theorem for the Taylor series method of order m

$e_n \leq Mh^{m+1}$ for some constant M

- Rough proof

$$y(t_n + h) = y(t_n) + h \cdot y'(t_n) + \cdots + \frac{h^m}{m!} y^{(m)}(t_n) + \frac{h^{m+1}}{(m+1)!} y^{(m+1)}(\xi)$$

for some $\xi \in (t_n, t_{n+1})$

$$e_k = |y_{n+1} - y(t_n + h)| = \frac{h^{m+1}}{(m+1)!} \left| y^{(m+1)}(\xi) \right| \leq \frac{h^{m+1}}{(m+1)!} \widetilde{M} = Mh^{m+1}$$

Errors

- Discretization error
  - **Well-posed** ODE: stable in the sense that there exists a constant C such that
$$|y(t) - \tilde{y}(t)| \le C|y_0 - \tilde{y}_0|$$
    - $\tilde{y}$: solution for a perturbed initial condition $\tilde{y}_0$
  - Total error $E \doteq \left|y_N - y(t_f)\right|$
  - Theorem: for a well-posed ODE,
$$E \le Ch^m \text{ if } e_n \le Mh^{m+1}$$
    - Rough proof: from the well-posedness,
$$E \le \tilde{C} \sum_{n=1}^{N} \left|e_L^{(n)}\right| \le \tilde{C} \sum_{n=1}^{N} \frac{Mh^{m+1}}{(m+1)!} = \frac{\tilde{C}NMh^{m+1}}{(m+1)!} = \frac{\tilde{C}t_f Mh^m}{(m+1)!}$$

General Explicit Runge-Kutta Methods

- $s_1 = f(t_n, y_n)$
- $s_2 = f(t_n + \alpha_2 \cdot h, y_n + s_1 \cdot \beta_{2,1} \cdot h)$
- …
- $s_k = f(t_n + \alpha_k \cdot h, y_n + s_1 \cdot \beta_{k,1} \cdot h + \ldots + s_{k\text{-}1} \cdot \beta_{k,k\text{-}1} \cdot h)$
- $y_{n+1} = y_n + (\gamma_1 \cdot s_1 + \ldots + \gamma_k \cdot s_k) \cdot h$
- $t_{n+1} = t_n + h$

- Note1: Order = k if k<5. (k = 6 for order 5, k = 9 for order 7, …)
- Note2: h can be different at each step.

❖ $k = 1$ & $\gamma_1 = 1 \rightarrow$ forward Euler's method

Heun's Method

- Also known as *explicit Trapezoid rule* or *improved Euler's method*

- $s_1 = f(t_n, y_n)$
- $s_2 = f(t_n + h, y_n + s_1 \cdot h)$
- $y_{n+1} = y_n + [(s_1+s_2)/2] \cdot h$
- $t_{n+1} = t_n + h$

- Total Error ~ $O(h^2)$ ☞ See Wen Shen Theorem 9.3.

❖ Cf.) Trapezoid Quadrature rule: $T = h\dfrac{f(a) + f(b)}{2}$

The Classical RK4

- $s_1 = f(t_n, y_n)$
- $s_2 = f(t_n + h/2, y_n + s_1 \cdot h/2)$
- $s_3 = f(t_n + h/2, y_n + s_2 \cdot h/2)$
- $s_4 = f(t_n + h, y_n + s_3 \cdot h)$
- $y_{n+1} = y_n + (s_1 + 2s_2 + 2s_3 + s_4) \cdot h/6$
- $t_{n+1} = t_n + h$

❖ Cf.) Simpson's rule $\frac{h}{6}\left[f(t_n) + 4f\left(t_n + \frac{h}{2}\right) + f(t_n + h)\right]$

Do It Yourself

- Apply the forward Euler's, the Heun's, and the CRK4 to

$$y' = y + 2t - t^2, \qquad y(0) = -1$$

and get $y_1(= y(h))$ values for $h = 0.2$.

  - See Wen Shen 9.3 & 9.6.

- [After this class]: Make your codes for the solutions on uniform grids. The exact solution is $y(t) = t^2 - e^t$. Compare errors of your results. Plot your solutions.

General Adaptive Runge-Kutta Methods

- $s_1 = f(t_n, y_n)$
- $s_2 = f(t_n+\alpha_2 \cdot h, y_n+s_1 \cdot \beta_{2,1} \cdot h)$
- …
- $s_k = f(t_n+\alpha_k \cdot h, y_n+s_1 \cdot \beta_{k,1} \cdot h+\ldots +s_{k\text{-}1} \cdot \beta_{k,k\text{-}1} \cdot h)$
- $y_{n+1} = y_n + (\gamma_1 \cdot s_1 + \ldots + \gamma_k \cdot s_k) \cdot h$
- $s_{k+1} = f(t_n+\alpha_{k+1} \cdot h, y_n+s_1 \cdot \beta_{k+1,1} \cdot h+\ldots +s_k \cdot \beta_{k+1,k} \cdot h)$
- $z_{n+1} = y_n + (\gamma^*_1 \cdot s_1 + \ldots + \gamma^*_k \cdot s_{k+1}) \cdot h$
- $t_{n+1} = t_n + h$
- Error: $e_{n+1} = y_{n+1} - z_{n+1} = (\delta_1 \cdot s_1 + \ldots + \delta_k \cdot s_k) \cdot h$

Adaptive Runge-Kutta-Fehlberg 45

- $s_1 = f(t_n, y_n)$
- $s_2 = f(t_n + \frac{1}{4}h,\ y_n + s_1 \cdot \frac{1}{4}h)$
- $s_3 = f(t_n + \frac{3}{8}h,\ y_n + s_1 \cdot \frac{3}{32}h + s_2 \cdot \frac{9}{32}h)$
- $s_4 = f(t_n + \frac{12}{13}h,\ y_n + s_1 \cdot \frac{1932}{2197}h - s_2 \cdot \frac{7200}{2197}h + s_3 \cdot \frac{7296}{2197}h)$
- $s_5 = f(t_n + h,\ y_n + s_1 \cdot \frac{439}{216}h - s_2 \cdot 8h + s_3 \cdot \frac{3680}{513}h - s_4 \cdot \frac{845}{4104}h)$
- $t_{n+1} = t_n + h$
- $y_{n+1} = y_n + (\frac{25}{216}s_1 + \frac{1408}{2565}s_3 + \frac{2197}{4104}s_4 - \frac{1}{5}s_5) \cdot h$ 4th order
- $s_6 = f(t_n + \frac{1}{2}h,\ y_n - s_1 \cdot \frac{8}{27}h + s_2 \cdot 2h - s_3 \cdot \frac{3544}{2565}h + s_4 \cdot \frac{1859}{4104}h - \frac{11}{40}s_5 \cdot h)$
- $z_{n+1} = y_n + (\frac{16}{135}s_1 + \frac{6656}{12825}s_3 + \frac{28561}{56430}s_4 - \frac{9}{50}s_5 + \frac{2}{55}s_5) \cdot h$ 5th order
- $e_{n+1} = y_{n+1} - z_{n+1}$

Adaptive Runge-Kutta-Fehlberg 45

- Algorithm
  1. Input: $t_0, t_f, y_0, h_0, n_{max}, e_{min}, e_{max}, h_{min}, h_{max}$
  2. Initialization: t, y, h & n(iteration number)
  3. Loop until $t \geq t_f$ or $n \geq n_{max}$
     ① Adjust h so that $h_{min} \leq h \leq h_{max}$
     ② Compute $s_1, s_2, \ldots, s_6, y_{n+1}, z_{n+1}$, & $e = |e_{n+1}|$
     ③ If ($e > e_{max}$ & $h > h_{min}$): $h = h/2$
     else: $n = n + 1$, $t = t + h$, $y_{n+1} = z_{n+1}$
     if ($e < e_{min}$): $h = h \times 2$

參考: Advanced Adaptive Algorithm

1. Initialize the internal parameters
   - Determine initial h carefully.
2. Main loop (from t_0 to t_f)
   1. Adjust h so that $h_{\min} \le h \le h_{\max}$
   2. Compute s_1, s_2, …, s_k, y_{n+1}, & z_{n+1}
   3. Compute e_{n+1} and check $|e_{n+1}| \le \max$ {abs. tol. , rel. tol. · Max($|y_{n+1}|$, $|y_n|$)}. (You can use abs. tol. only)
   4. Compute new h by multiplying $\{(\text{tol.})/(\text{err.})\}^{1/p}$ (p: y_{n+1}'s order) but avoid too much change (so multiply 0.8 or 0.9 and set $h_{\max}$ and $h_{\min}$ for the first loop step).

參考: Advanced Adaptive Algorithm

- For vectors $|y_{n+1}|$ and e_{n+1}, choose the maximum element, respectively.
- If error>tolerance, do not change t and redo the steps 1~3 with the new h.

- Optimal initial step size h,
  - I. Gladwell *et al*., J. Comp. Appl. Math. **18,** 175-192 (1987).
    - In most case, this is enough (*p*: y_{n+1}'s order)

$$h_0 = (0.8 \text{ or } 0.9) \cdot \min\{|t_0 - t_f|, (\text{tol.})^{1/p} / \| f(t_0, y_0)/\max(|y_0|, \textbf{threshold}) \|_\infty\}$$

參考: Step-size Strategy

- Fixed (= constant) step-size
  - Trial and error
    - Consider global errors
    - Try *h* from a large value to small one
      (ex.: 0.1, 0.01, 0.001, ……)
    - The tested *h* is okay if $y(t)$ is the same (in the desired accuracy) for smaller step-size values.
    - h = 0.005 is enough for many cases, but do not skip testing.

參考: Step-size Strategy

- Variable step-size
  - Process of step-size change
    1. Estimate local error for the given h
    2. Accept h if |local error| < (tolerance)
    3. (new h) = (old h) $\cdot$ $\min[5, \mathrm{s}\cdot\{(\mathrm{tol.})/|\mathrm{l.e.}|\}^{1/(p+1)}]$ (p: order)

       s: safety factor (0 < s < 1. Usually 0.8 or 0.9)
    - Instead of min[5, ...], you can use min[3, ...] or min[4, ...]

參考: Step-size Strategy

- Variable step-size
  - References
    - F. T. Krogh, "Algorithms for Changing Step Size", SIAM J. Numer. Anal. 10, 949 (1973).
    - M. L. Michelsen, "An Efficient General Purpose Method for the Integration of Stiff Ordinary Differential Equations", A. I. Ch. E. J. 22, 594 (1976).
    - J. Villadsen and M. L. Michelsen, *Solution of Differential Equation Models by Polynomial Approximation,* Prentice-Hall, Englewood Cliffs, N.J.
    - R. L. Johnston, *Numerical Methods-A Software Approach,* Wiley, New York

參考: Step-size Strategy

- Variable step-size
  - References
    - L. F. Shampine and A. Witt, "A simple step size selection algorithm for ODE codes", J. Comput. Appl. Math. 58, 345 (1995).
    - A. Usman and G. Hall, "Alternative stepsize strategies for Adams predictor-corrector codes", J. Comput. Appl. Math. 116, 105 (2000).
    - L. F. Shampine, "Error Estimation and Control for ODEs", J. Sci. Comput. 25, 3 (2005).
    - E. Alberdi Celaya *et al*., "Implementation of an Adaptive BDF2 Formula and Comparison with the MATLAB Ode15s", Procedia Comput. Sci. 29, 1014 (2014)

參考: Step-size Strategy

- Variable step-size
  - Advantages
    - More efficient for most ODEs than fixed step-size
    - Automatic, thus more convenient to users
  - Disadvantages
    - If the system is very complex (too many equations, for example), local error estimation demands huge time.
    - Implantation of parallel computing is impossible or very difficult.

Backward Euler's Method

- $y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})$
- $t_{n+1} = t_n + h$

- 1st order *Implicit method*
  – To solve it at each step requires an iterative method.

  (You must set the limit of iteration number, usually order of 10.)

- Cf.) Backward 2-point differentiation

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O\left(h\right)$$

Backward Euler's Method

- Iterative method: usually, fixed-point iteration or modified Newton's method.

- If $f(t_{n+1}, y_{n+1})$ is just $\lambda \cdot y_{n+1} + g(t_{n+1})$, you can solve it algebraically.
  - For vector-valued functions, use a linear algebra library or module.

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\Lambda\boldsymbol{y}_{n+1} + h\boldsymbol{g}(t_{n+1})$$

$$\rightarrow (\boldsymbol{I} - h\Lambda)\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{g}(t_{n+1})$$

參考: (Implicit) Trapezoidal Rule

– *implicit Trapezoid rule* (2nd order method)

- $s_1 = f(t_n, y_n)$
- $s_2 = f(t_{n+1}, y_{n+1})$
- $y_{n+1} = y_n + [(s_1+s_2)/2]\cdot h$
- $t_{n+1} = t_n + h$

Cf.) $s_2 = f(t_{n+1}, y_n + s_1\cdot h)$ for the explicit version

- Averaging forward and backward Euler methods
- Iteration method: Fixed-point iteration or modified Newton's method

參考: General Implicit RK Methods

- $s_1 = f(t_n + \alpha_1 \cdot h,\ y_n + s_1 \cdot \beta_{1,1} \cdot h + \ldots + s_k \cdot \beta_{1,k} \cdot h)$
- $s_2 = f(t_n + \alpha_2 \cdot h,\ y_n + s_1 \cdot \beta_{2,1} \cdot h + \ldots + s_k \cdot \beta_{2,k} \cdot h)$
- …
- $s_k = f(t_n + \alpha_k \cdot h,\ y_n + s_1 \cdot \beta_{k,1} \cdot h + \ldots + s_k \cdot \beta_{k,k} \cdot h)$
- $y_{n+1} = y_n + (\gamma_1 \cdot s_1 + \ldots + \gamma_k \cdot s_k)\cdot h$
- $t_{n+1} = t_n + h$

- Generally, the order of a method has weak dependence on k. (Even $k = 1$ doesn't guarantee order 1.)

Stiff ODEs

- *An ODE* is ***stiff*** if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method is **numerically unstable** unless very small steps are taken.
  - There is no exact definition for stiff ODEs.

✓ You should use implicit methods.

❖ Newton’s method converges better for stiff equations than fixed-point iteration.

Stiff ODEs

– Wen Shen pp. 209~210

- Ex.) $y' = -ay$, $y(0) = 1$
  - a: large positive constant
  - – Forward Euler method gives

$$y_0 = 1 \text{ \& } y_{n+1} = y_n - ahy_n = (1 - ah)y_n$$

$$\rightarrow \quad y_n = (1 - ah)^n$$

  - – This solution does not diverge as $n \to \infty$ only if

$$h < 2/a$$

  - – Huge a means necessity of tiny h. → stiff

Stiff ODEs

– Wen Shen pp. 209~210

- Ex.) $y' = -ay$, $y(0) = 1$
  - a: large positive constant
  - – Backward Euler method gives

$$y_0 = 1 \;\&\; y_{n+1} = y_n - ahy_{n+1}$$

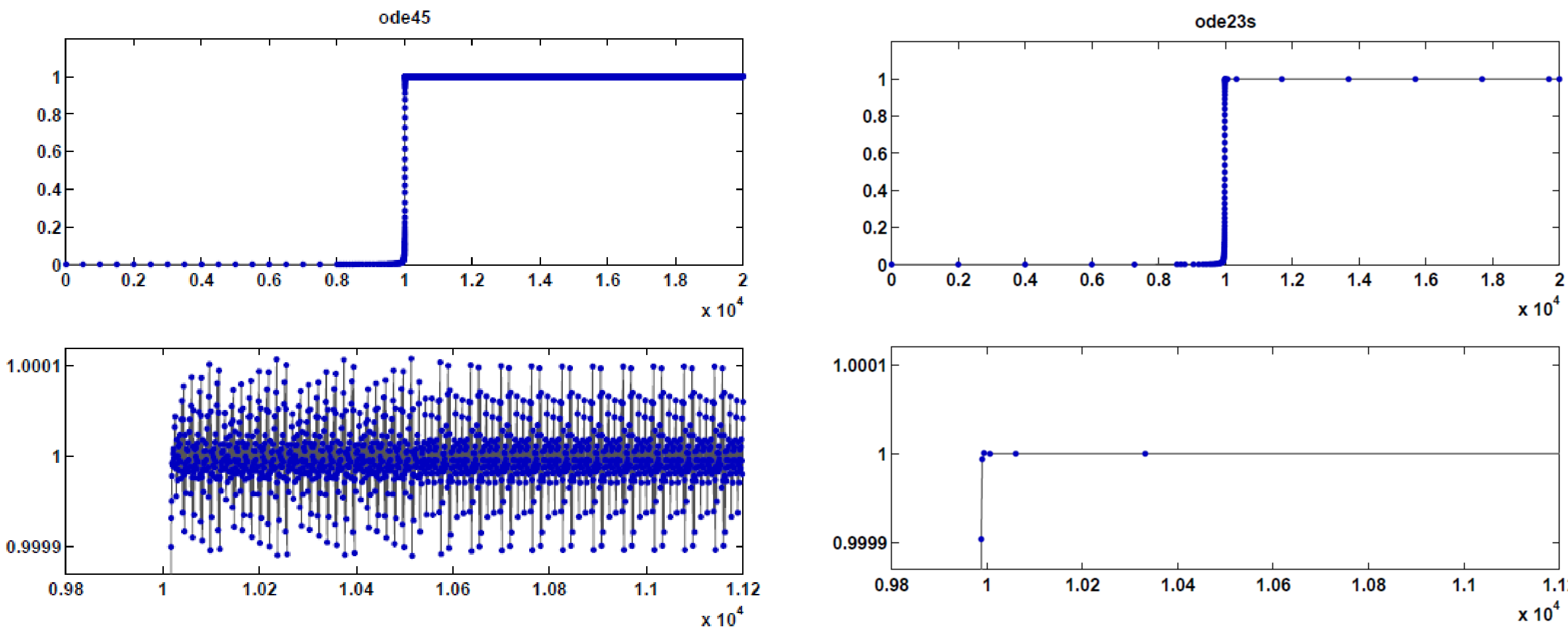
$$\rightarrow \quad y_n = \left(\frac{1}{1+ah}\right)^n$$

  - – Since $ah > 0$, this solution does not diverge for any choice of $h(> 0)$. [unconditionally stable]

Stiff ODEs

– Moler 7.9

- Ex.) $\dot{y} = y^2 - y^3 \qquad y(0) = 0.00001$



Figures from Moler

Stiff ODEs

– Wen Shen pp. 211~212

- Ex.) $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -20x - 19y \\ -19x - 20y \end{pmatrix} = \begin{pmatrix} -20 & -19 \\ -19 & -20 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$, $x(0) = 2,\ y(0) = 0.$
  - Eigenvalues of the matrix: -1 & -39.

  → condition number = 39 (rather large)
  - Exact solution

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} e^{-39t} + e^{-t} \\ e^{-39t} - e^{-t} \end{pmatrix}$$

    - For large t, the term e^{-t} dominates, and the term e^{-39t} is negligible and called the *transient term*.

Stiff ODEs

– Wen Shen pp. 211~212

- Ex.) $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -20x - 19y \\ -19x - 20y \end{pmatrix} = \begin{pmatrix} -20 & -19 \\ -19 & -20 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$, $x(0) = 2,\ y(0) = 0.$
  - Forward Euler method gives
  $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$ & $\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n + h(-20x_n - 19y_n) \\ y_n + h(-19x_n - 20y_n) \end{pmatrix}$
  $$\Rightarrow \begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} (1-39h)^n + (1-h)^n \\ (1-39h)^n - (1-h)^n \end{pmatrix}$$
  - Stability conditions: $h < 2/39$ & $h < 2$
  → $h < 2/39$ (The transient term restricts h.)

Stiff ODEs

- Wen Shen pp. 211~212

- Ex.) $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -20x - 19y \\ -19x - 20y \end{pmatrix} = \begin{pmatrix} -20 & -19 \\ -19 & -20 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$, $x(0) = 2,\ y(0) = 0.$
  - Backward Euler method gives
    $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$ & $\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n - 20x_{n+1} - 19y_{n+1} \\ y_n - 19x_{n+1} - 20y_{n+1} \end{pmatrix}$
    $(\mathbf{I} - h\mathbf{A}) \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} \Longrightarrow \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = (\mathbf{I} - h\mathbf{A})^{-1} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$
    - where $\mathbf{A} = \begin{pmatrix} -20 & -19 \\ -19 & -20 \end{pmatrix}$
  - Unconditionally stable because $\|(\mathbf{I} - h\mathbf{A})^{-1}\|_2 < 1$

參考: Stability of a Method

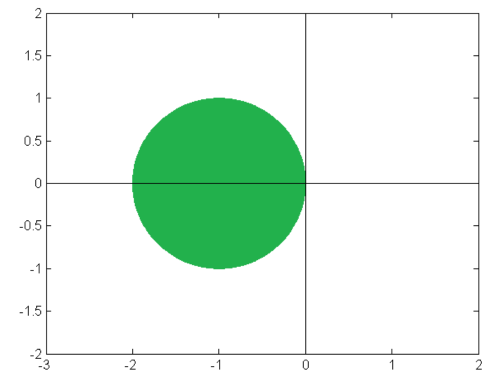
- Absolute stability
  - A numerical method is absolutely stable if its numerical solution $y_n \to 0$ as $n \to \infty$ for a test ODE $\dot{y} = ky$.
  - (Definition of stability may be slightly different in other literature.)
- Stability region
  - Absolutely stable region of $z = hk$ in complex plane for a numerical method

參考: Stability of a Method

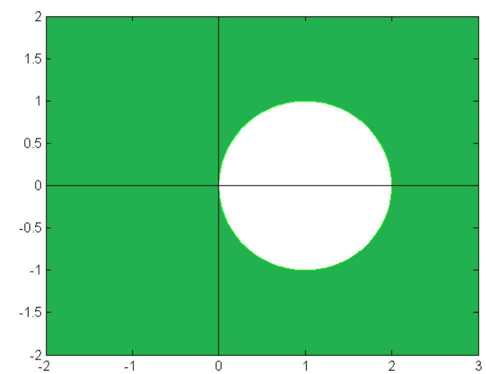
- Stability function
  - When a numerical method's solution for a test ODE $\dot{y} = ky$ is expressed as $y_{n+1} = \varphi(hk)\cdot y_n$, the function φ is the stability function.
- A-stability
  - A numerical method is A-stable if it is absolutely stable for $\mathrm{Re}(hk) < 0$, *i.e.*, $|\varphi(z)| < 1$ for $\mathrm{Re}(z) < 0$.
- L-stability
  - A-stability with $|\varphi(z)| \to 0$ as $z \to -\infty$.

參考: Stability of a Method

- Forward Euler method
  - $y_{n+1} = y_n + h{\cdot}k{\cdot}y_n$ for $\dot{y} = ky$
  - Stability function: $\varphi(z) = 1 + z$
  - Stability region: $|1 + z| < 1$

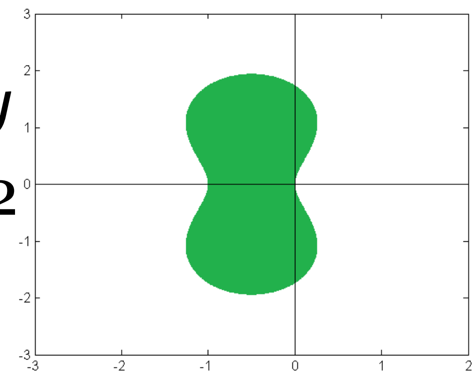


- Backward Euler method (L-stable)
  - $y_{n+1} = y_n + h{\cdot}k{\cdot}y_{n+1}$ for $\dot{y} = ky$
  - Stability function: $\varphi(z) = 1/(1 - z)$
  - Stability region: $|1 - z| > 1$

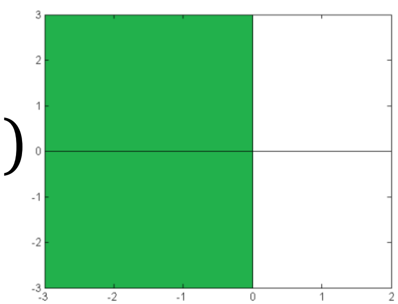


參考: Stability of a Method

- Heun method
  - $y_{n+1} = y_n + (1+ k{\cdot}h){\cdot}k{\cdot}h{\cdot}y_n/2$ for $\dot{y} = ky$
  - Stability function: $\varphi(z) = 1+(1 + z){\cdot}z/2$
  - Stability region: $|1 + (1 + z){\cdot}z/2| < 1$



- Trapezoidal method (A-stable, not L-stable)
  - $y_{n+1} = y_n + (y_{n+1} + y_n){\cdot}k{\cdot}h/2$ for $\dot{y} = ky$
  - Stability function: $\varphi(z) = (1+z/2)/(1-z/2)$
  - Stability region: $|1 + z/2| < |1 - z/2|$



Multistep Methods

- *Longer memory*
- Using y_{n-p+1}, y_{n-p+2}, … y_{n-1}, y_n to get y_{n+1}
  - Self-starting is impossible. (Starting with single-step methods)
  - Good for problems with smooth solutions and high accuracy requirements.
    - Ex.) the orbits of planets

Linear Multistep Methods

- $t_n = t_0 + n \cdot h$
- $s_k = f(t_{n-k}, y_{n-k})$
- $y_{n+1} = \alpha_p \cdot y_{n-p} + \alpha_{p-1} \cdot y_{n-p+1} + \ldots + \alpha_1 \cdot y_{n-1} + \alpha_0 \cdot y_n + (\beta_{-1} \cdot s_{-1} + \beta_0 \cdot s_0 + \beta_1 \cdot s_1 + \ldots + \beta_k \cdot s_k) \cdot h$
  - explicit if $\beta_{-1} = 0$
  - implicit if $\beta_{-1} \neq 0$

◆ A good way to find coefficients: Polynomial interpolation

Linear Multistep Methods

- Adams-Bashforth methods
  - From the Lagrange interpolation. Explicit methods
  - $s_k = f(t_{n\text{-}k}\,,\ y_{n\text{-}k})$
  - $y_{n+1} = y_n + (\beta_0 \cdot s_0 + \beta_1 \cdot s_1 + \ldots + \beta_k \cdot s_k\ ) \cdot h$

  - Order = 1 → Forward Euler ($\beta_0 = 1$, $\beta_k = 0$ for $k \neq 0$)
  - Order = 2 → $y_{n+1} = y_n + [3f(t_n\,,\ y_n) - f(t_{n\text{-}1}\,,\ y_{n\text{-}1})] \cdot h/2$
  - Order = 3 → $\beta_0 = 23/12$, $\beta_1 = -4/3$, $\beta_2 = 5/12$
  - Order = 4 → $\beta_0 = 55/24$, $\beta_1 = -59/24$, $\beta_2 = 37/24$, $\beta_3 = -3/8$

❖ AB methods include extrapolation reducing accuracy.

Linear Multistep Methods

- Adams-Moulton methods
  - From the Lagrange interpolation. Implicit methods
  - $s_k = f(t_{n-k}, y_{n-k})$
  - $y_{n+1} = y_n + (\beta_{-1} \cdot s_{-1} + \beta_0 \cdot s_0 + \beta_1 \cdot s_1 + \ldots + \beta_k \cdot s_k) \cdot h$
  - Order = 1 → Backward Euler ($\beta_{-1} = 1$, $\beta_k = 0$ for $k \neq -1$)
  - Order = 2 → Trapezoidal rule ($\beta_{-1} = 1/2$, $\beta_0 = 1/2$)
  - Order = 3 → $\beta_{-1} = 5/12$, $\beta_0 = 2/3$, $\beta_1 = -1/12$
  - Order = 4 → $\beta_{-1} = 3/8$, $\beta_0 = 19/24$, $\beta_1 = -5/24$, $\beta_2 = 1/24$

✓ Initial guess for iteration of each step: AB method of order 2 is the best.

參考: Linear Multistep Methods

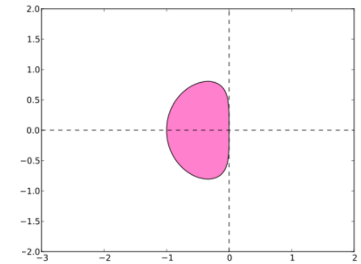
- Backward differentiation formulas (BDF)
  - Directly derived from backward finite difference formulas
  - Implicit methods
  - $s_k = f(t_{n-k}, y_{n-k})$
  - $y_{n+1} = \alpha_p \cdot y_{n-p} + \alpha_{p-1} \cdot y_{n-p+1} + \ldots + \alpha_1 \cdot y_{n-1} + \alpha_0 \cdot y_n + \beta_{-1} \cdot s_{-1} \cdot h$
  - Order = 1 → Backward Euler ($\alpha_0 = 1$, $\alpha_k = 0$ for $k \neq 0$, $\beta_{-1} = 1$)
  - Order = 2 → $\alpha_0 = 4/3$, $\alpha_1 = -1/3$, $\beta_{-1} = 2/3$
  - Order > 2: not A-stable. Order > 6: not zero-stable.
  - More efficient than Adams-Bashforth or Adams-Moulton

參考: Linear Multistep Methods

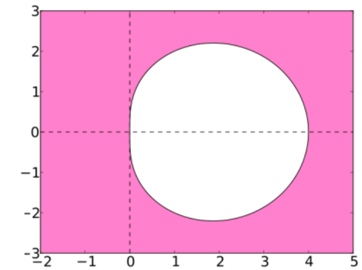
- Zero-stability
  - A numerical method's numerical solution $y_n \rightarrow$ (a finite value) as $n \rightarrow \infty$ for a test ODE $\dot{y} = 0$.
- Characteristic polynomial
  - From $y_{n+1} = \alpha_{\mathrm{p}} \cdot y_{n-p} + \alpha_{\mathrm{p-1}} \cdot y_{n-p+1} + \ldots + \alpha_1 \cdot y_{n-1} + \alpha_0 \cdot y_n + (\beta_{-1} \cdot s_{-1} + \beta_0 \cdot s_0 + \beta_1 \cdot s_1 + \ldots + \beta_k \cdot s_k) \cdot h$
  - By testing $\dot{y} = ky$ and substituting $hk \rightarrow z$ & $y_n \rightarrow x^n$
  - Characteristic polynomial $\Phi(x, z) = x^{n+1} - \alpha_{\mathrm{p}} \cdot x^{n-\mathrm{p}} - \alpha_{\mathrm{p-1}} \cdot x^{n-\mathrm{p}+1} - \ldots - \alpha_1 \cdot x^{n-1} - \alpha_0 \cdot x^n - (\beta_{-1} \cdot x^{n+1} + \beta_0 \cdot x^n + \beta_1 \cdot x^{n-1} + \ldots + \beta_k \cdot x^{n-k}) \cdot z$

參考: Linear Multistep Methods

- Stability region
  - For all roots x_i's of $\Phi(x, z) = 0$, the region of z satisfying $|x_i| < 1$.
  - Ex.) Adams-Bashforth order 2
    - $\Phi(x, z) = x^2 - (1 + 3z/2)x + z/2$
    - Stability region: $\left|\frac{1}{2}\left(1+\frac{3}{2}z \pm \sqrt{1+z+\frac{9}{4}z^2}\right)\right| < 1$



  - Ex.) BDF order 2
    - $\Phi(x, z) = (1 - 2z/3)x^2 - 4x/3 + 1/3$
    - Stability region: $\left|\left(2 \pm \sqrt{1+2z}\right)\right| < \left|3 - 2z\right|$



Predictor-Corrector Method

- Predictor: explicit method (sometimes multistep)
- Corrector: implicit method (often multistep)

▪ Ex.) trapezoidal rule with forward Euler method
  (→ Heun method)

- Most popular pairs of multistep methods are
  Adams-Bashforth (predictor)
  + Adams-Moulton (corrector)

Predictor-Corrector Method

– Wen Shen p.193

- Ex.) 2nd order ABM method
  - $f_n = f(t_n, y_n)$

1. Predictor

$$y_{n+1}^* = y_n + h(3f_n - 2f_{n-1})/2$$
$$\rightarrow \; f_{n+1}^* = f(t_{n+1}, y_{n+1}^*)$$

2. Corrector

$$y_{n+1} = y_n + h(f_{n+1}^* + f_n)/2$$
$$\rightarrow \; f_{n+1} = f(t_{n+1}, y_{n+1})$$

✓ No need of iteration at each step

➢ This is the PECE mode.

參考: Predictor-Corrector Method

- Variants of predictor-corrector methods
  - PECE mode
    - Most frequently used one
    - Predict(y_{n+1}) – evaluate(f^*_{n+1}) – correct(y_{n+1}) – evaluate(f_{n+1})
  - PEC mode
    - Predict(y_{n+1}) – evaluate(f^*_{n+1}) – correct(y_{n+1})
  - P(EC)^k mode
  - P(EC)^{k}E mode
  - PE(CE)$^\infty$ mode
    - Equivalent to fixed-point iteration

Do It Yourself

- Write your codes to solve
  $y'' + y' + 0.75y = 0, \qquad y(0) = 3, y'(0) = -2.5$

  with the 2nd order AB and the 2nd order ABM (PC) method on uniform grids.
  - You may refer to Wen Shen 9.9 & p.203

- [After this class]: The exact solution is $y(t) = 2e^{-0.5t} + e^{-1.5t}$ & $y'(t) = -e^{-0.5t} - 1.5e^{-1.5t}$. Compare errors of your results. Plot your solutions.

參考: Error Estimation for ABM

- Milne's device
  - For AB and AM methods of the same order *p*, the local errors have the form

    $e_{AB} \approx C_{AB} \cdot h^{p+1} \cdot y^{(p+1)}(\xi_n)$

    $e_{AM} \approx C_{AM} \cdot h^{p+1} \cdot y^{(p+1)}(\xi_n)$

    - C_{AB}, C_{AM} : constant coefficients

  $|\text{ local error of } y\ | \approx |C_{AM}/(C_{AB} - C_{AM})| \cdot |\ y_{AB} - y_{AM}\ |$

  ✓ You must know the constant coefficients (from derivation of the formulas).

參考: Error Estimation for ABM

- Milne’s device
  - Error coefficients AB and AM methods

Order	1	2	3	4	5	6
C_{AB}	1/2	5/12	3/8	251/750	95/288	19087/60480
C_{AM}	-1/2	-1/12	-1/24	-19/720	-3/160	-863/60480

❖ For BDFs, see Brenan *et al*., “Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations”, SIAM.

References

- Wikipedia
- C. Moler,
  Numerical Computing with MATLAB
- Wen Shen,
  An Introduction to Numerical Computation

References

- K. Atkinson *et al*., “Numerical Solution of Ordinary Differential Equations”
- E. Suli, “Numerical Solution of Ordinary Differential Equations”
- Etc.

Further Study

- About the **Bogacki-Shampine (RK23)** Method

- About **variable order** methods