Initial Value Problem

CSS 455

Winter 2012

IVP

Turner uses dy/dx = f(x,y) instead of dy/dt = f(t,y)

- Consider the differential equation:

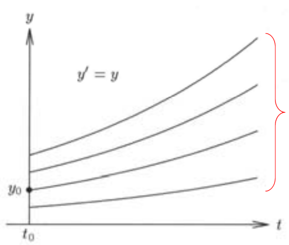
$$\frac{dy(t)}{dt} = ay(t) \text{ or } y'(t) = ay(t)$$

By inspection, its solution is: $y(t) = ce^{at}$

Each choice of *c* is a different solution, and together they form the *family* of solutions.

More generally: $y'(t) = f(t, y(t))$

Family of solutions $y(t) = ce^{at}$



- Case: $a=1$.

Unstable family of solutions due to the nature of the ODE being solved.

Family of solutions $y(t) = ce^{at}$

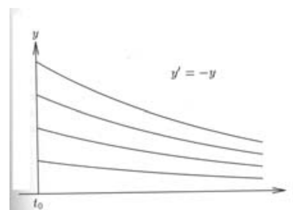


FIGURE 9.2
The family of solution curves for the ODE $y' = -y$.

- Case: $a=-1$.

Stable family of solutions due to the nature of the ODE being solved.

Family of solutions for $y'(t) = a$

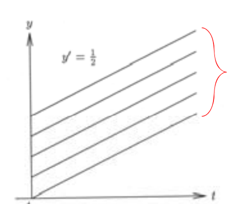


FIGURE 9.3
The family of solution curves for the ODE $y' = \frac{1}{2}$.

- Case: $a=1/2$.
  $y(t) = (1/2)t+c$

Neutrally stable family of solutions due to the nature of the ODE being solved.

Family of solutions $y(t) = ce^{at}$

Initial Value Problems specify the family member by specifying the initial values of t and y: $\{t_0, y_0\}$

- For example: at $t = 0$, $y = 1$. *(requires that $c = 1$).*

Taylor Series Expansion for *y(t)*

The value of *y(t)* at points near t_0 can be expressed in terms of its derivatives:

$$y(t_0+h) = y(t_0) + y'(t_0)h + \left(\tfrac{1}{2}\right)y''(t)h^2 + \ldots$$

In the Euler method only the first two terms are retained in the approximation:

$$y(t_0+h) \approx y(t_0) + y'(t_0)h$$

$$y(t_0+h) \approx y(t_0) + f(t_0, y(t_0))h$$

Looks like a fixed point iteration

$$y(t_1) \approx y(t_0) + f(t_0, y(t_0))h$$

$$y_1 \approx y_0 + f(t_0, y_0)h \qquad y_2 \approx y_1 + f(t_1, y_1)h$$

$$y_{n+1} \approx y_n + f(t_n, y_n)h_n$$

Local truncation error is error introduced *in a single step* by truncation of the series:

$LTE = \frac{h^2}{2}y''(\eta)$, for some $\eta \in [t_n, t_{n+1}]$

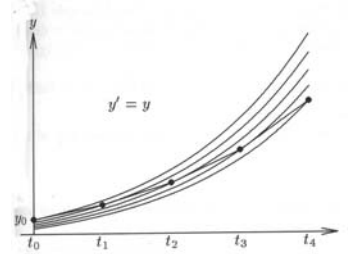
k-order method: LTE is proportional to h^{k+1}

Euler is 1st order method. we will see why later

Global Truncation Error

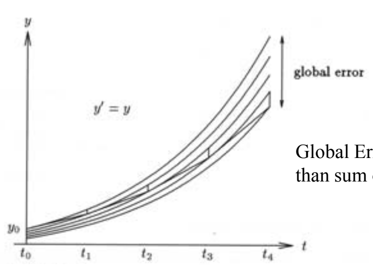
- The errors at each step move the approximate solution among the family members.
- The Global Truncation Error is the total error at time t_n *relative to the exact solution starting from the initial value.*
- GTE may be smaller or larger than sum of LTE's
- For stable IVP's, the GTE can be controlled by limiting the LTE's.

Euler's method for unstable ODE



Euler's method for the ODE $y' = y$.

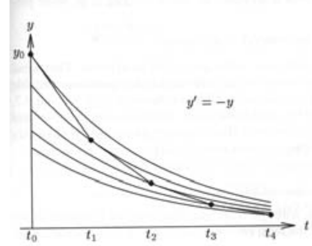
Euler's method for unstable ODE



Global Error is larger than sum of local errors

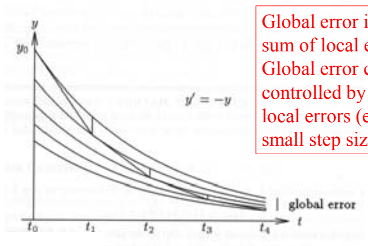
Local and global errors in Euler's method for the ODE $y' = y$.

Euler's method for Stable ODE



Euler's method for the ODE $y' = -y$.

Euler's method for Stable ODE



Local and global errors in Euler's method for the ODE $y' = -y$.

Global error is less than sum of local errors. Global error can be controlled by limiting local errors (e.g. through small step size)

Methods also have stability

- With unstable methods, small perturbations initial conditions cause diverging results.
- Apply Euler method to

$$y'(t) = -10y(t)$$

$$y_{n+1} = y_n - 10y_n h = (1-10h)y_n$$

$$y(t) = y(0) * e^{-10*(t-t_0)}$$

If h >1/5, |1-10*h*| > 1, then the error will grow from one step to the next, *even though the IPV itself is stable.* (small *h* is required) *demo1ch6.m*

The family of solutions is stable here. But, the method is not always stable.

Effect of step size

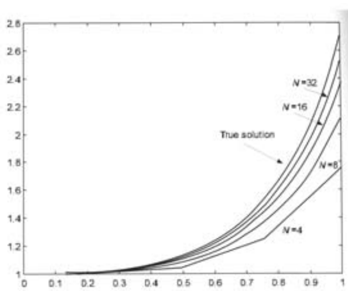


Figure 6.3 Euler's method for (6.5)

Backward Euler (implicit) Method

- Use estimate of derivative at *next* interval rather than present one to step forward.

$$y_{n+1} = y_n + f(t_{n+1}, y_{n+1})h_n$$

In some simple cases, we know $f(t_{n+1}, y_{n+1})$:

$$f(t_{n+1}, y_{n+1}) = y'(t_{n+1}) = ay(t_{n+1}) = ay_{n+1}$$

$$y_{n+1} = y_n + ah_n y_{n+1} \quad \text{or} \quad y_{n+1} = \frac{y_n}{1 - ah_n}$$

Still 1st order, but more generally stable.

Backward Euler (implicit) Method

- For the demo case, y'(t) = -10*y

$$y_{n+1} = \frac{y_n}{1 + 10h_n}$$

More general

- Use equation solution methods to solve the equation for y_{n+1}:

$$y_{n+1} = y_n + f(t_{n+1}, y_{n+1})h_n$$

See *demo2ch6.m*

Note that it is stable for a wider range of step sizes, even when they are very crude. The cost is that in general, you must solve for y_{n+1} in some way on each iteration.

Improvement?

- We have two first order methods that estimate the slope by:
  - the present point: $f(t_n,y_n)$ or f_n
  - The next point: $f(t_{n+1},y_{n+1})$ or f_{n+1}
- What might be an improvement, similar to the improvements we made in estimating derivatives?
- Average the two.

Average of forward and backward

$$y_{n+1} = y_n + \frac{f_n + f_{n+1}}{2} h$$

For the demo problem

$$f_n = -10 y_n$$
$$f_{n+1} = -10 y_{n+1}$$

Substitute these values, and solve for y_{n+1}

$$y_{n+1} = y_n \left(\frac{1-5h}{1+5h} \right)$$

see demo3ch6.m
second order

Runge-Kutta Methods

- Higher order methods that use intermediate points between t_n and t_{n+1} to estimate the slope of y over the interval.
- Look at the Taylor expansion for y_{n+1}:

$$y_{n+1} \approx y_n + hf_n + \frac{h^2}{2} y_n'' +$$

The idea is to improve upon the previous methods by estimating the 2nd derivative above.

Estimate y”

Define: $k_1 = y_n' \equiv f(x_n, y_n)$

for some intermediate point, $x_n + \alpha h$

$$k_2 = f(x_n + \alpha h, y_n + \alpha h k_1)$$

$$y_{n+1} \approx y_n + hf_n + \frac{h^2}{2} y_n'' +$$

$$y_{n+1} \approx y_n + hk_1 + \frac{h^2}{2} \left(\frac{k_2 - k_1}{\alpha h} \right)_n$$

Estimate y”

$$y_{n+1} \approx y_n + hk_1 + \frac{h^2}{2} \left(\frac{k_2 - k_1}{\alpha h} \right)_n$$

$$y_{n+1} = y_n + k_1 h \left(1 - \tfrac{1}{2\alpha}\right) + k_2 \left(\tfrac{h}{2\alpha}\right)$$

Choice of α provides a range of 2nd order accurate, self-starting methods.

4th order Runge-Kutta RK4

$$k_1 = f(x_n, y_n)$$
$$k_2 = f(x_n + h/2, y_n + hk_1/2)$$
$$k_3 = f(x_n + h/2, y_n + hk_2/2)$$
$$k_4 = f(x_n + h, y_n + hk_3)$$

$$y_{n+1} = y_n + \tfrac{h}{6}\left[k_1 + 2(k_2 + k_3) + k_4\right]$$

4th order accurate, self-starting, easy to program.

Multistep Methods

- Use more than one point to calculate y_{n+1}, but all these points are tabulated ones.
- Explicit: depend only on $y_{n,}$, y_{n-1}, etc. (*Adams-Bashforth)*
- Implicit: depend upon y_{n+1},y_n, y_{n-1},etc. (*Recall backward Euler*) If used independently, must solve the equation for y_{n+1} as in Euler. *One order more accurate than corresponding explicit method- but takes more calculations.*

Multistep Methods

- Predictor-Corrector:
  - Use explicit to get approximation to y_{n+1}.
  - Use this estimate in implicit method to get improved value for y_{n+1}.
  - Superior to some other algorithms of same order.
- see *demo4ch6.m*

Systems of Diffl Equations

- Multiple 1st order equations that are coupled. Solve in vector format, incrementing all elements of vector along with each time step.
- Second order initial value problem:

$$y'' = f(x, y, y'); y(x_0) = y_0; y'(x_0) = y'_0$$

Can be recast as a system of two 1st order differential equations that are coupled.

Boundary Value Problems

- Shooting Methods:

$$y'' = f(x, y, y'); y(a) = y_a; y(b) = y_b$$

For a value of y'(a), we can solve the 2nd order initial value problem:

$$y'' = f(x, y, y'); y(a) = y_a; y'(a) = z$$

This solution is denoted y(x;z)

The boundary condition is satisfied when $y(b;z)=y_b$

Define and solve: $F(z)=y(b;z)-y_b = 0$ (How?)

Boundary Value Problems

- Finite Difference Methods:

$$y'' + a(x)y' + b(x)y = f(x);$$
$$y(x_0) = y_0; y(x_N) = y_N$$

This solution inteval $[x_0,x_N]$ is divided into N steps.

Step size is *h*.

At each point x_k, use finite difference approximations to the derivatives, yielding an equation for each point.

Involve k-1 and k+1 steps in central differences.

Finite Difference Methods:

This solution interval $[x_0,x_N]$ is divided into N steps.

Step size is *h*.

At each point x_k, use finite difference approximations to the derivatives, yielding an equation for each point.

Involve k-1 and k+1 steps in central differences.

Results in *tridiagonal* system of linear equations.

$$(2 - ha_k)y_{k-1} + (2h^2 b_k - 4)y_k + (2 + ha_k)y_{k+1} = 2h^2 f_k$$

Large *k* needed for practical situations.