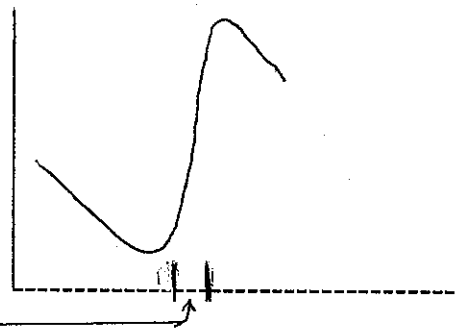
Adaptive Methods

Motivation



abrupt change: need "adaptive step-size control" based on local truncation error at each step

Embedded RK method (RK-Fehlberg method)

ϵ = | result of RKi - result of RKj |

(1) **ode23**

- second- and third-order RK (3 function values)

$y_{i+1} = y_i + \frac{h}{9}(2k_1 + 3k_2 + 4k_3)$,
where

$k_1 = f(t_i, y_i)$

$k_2 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$

$k_3 = f(t + \frac{3}{4}h, y_i + \frac{3}{4}k_2 h)$

$E_{i+1} = \frac{1}{72}(-5k_1 + 6k_2 + 8k_3 - 9k_4)h$ // $k_4 = f(t_{i+1}, y_{i+1})$

Error $\leq$ max (RelTol x |y|, AbsTol) // criterion to accept y_{i+1}
(10^{-3}) (10^{-6})

(2) **ode45**

- fourth- and fifth-order RK (6 function values)

Ex 21.2

$$\begin{cases} \dfrac{dy}{dt} = 10\, e^{-(t-2)^2/[2(0.075)^2]} - 0.6y \\ y(0) = 0.5 \end{cases}$$

$t = [0, 4]$.

Solution dydt.m

```
function yp = dydt(t, y)
yp = 10*exp(-(t-2)*(t-2)/(2*0.075^2)) - 0.6*y;
```

```
>> ode23(@dydt, [0 4], 0.5);
```

or

```
>> options = odeset('RelTol', 1e-4);
>> ode23(@dydt, [0,4], 0.5, options);
```

FIGURE 21.4
Solution of ODE with MATLAB. For (*b*), a smaller relative error tolerance is used and hence many more steps are taken.

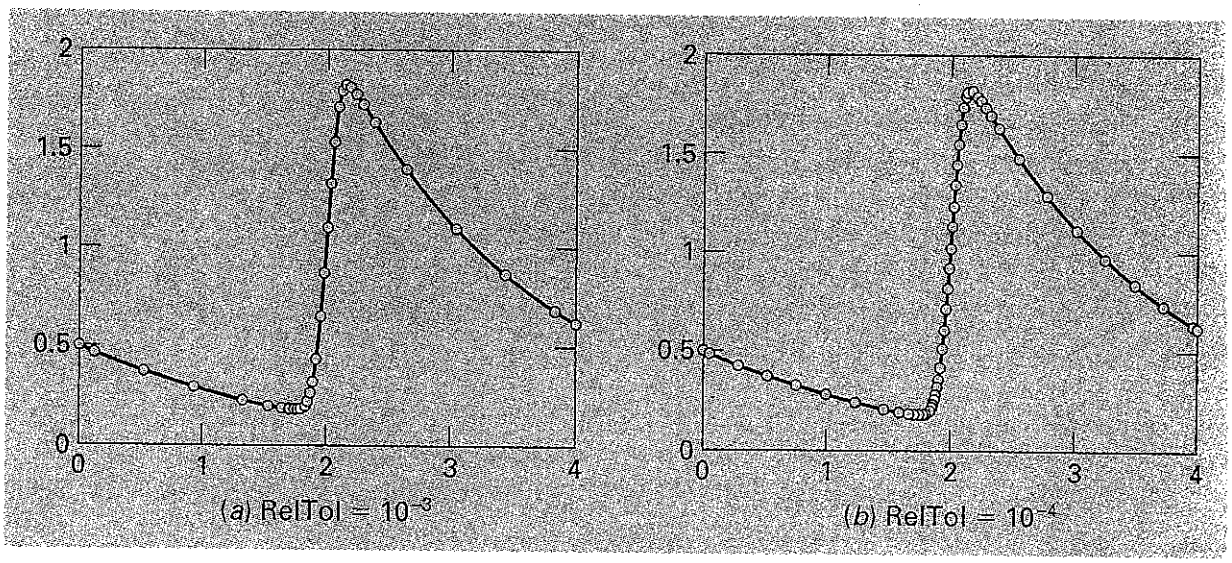


Table 12.6 MATLAB's Differential Equation Solvers

Ordinary Differential Equation Solver Function	Type of Problems Likely to be Solved with This Technique	Numerical Solution Method	Comments
ode45	nonstiff differential equations	Runge–Kutta	Best choice for a first-guess technique if you don't know much about the function. Uses an explicit Runge–Kutta (4,5) formula called the Dormand–Prince pair.
ode23	nonstiff differential equations	Runge–Kutta	This technique uses an explicit Runge–Kutta (2,3) pair of Bogacki and Shampine. If the function is "mildly stiff," this may be a better approach than **ode 45**.
ode113	nonstiff differential equations	Adams	Unlike **ode45** and **ode23**, which are single-step solvers, this technique is a multistep solver.
ode15s	stiff differential equation and differential algebraic equations	NDFs (BDFs)	Uses numerical differentiation formulas (NDFs) or backward differentiation formulas (BDFs). It is difficult to predict which technique will work best on a stiff differential equation.
ode23s	stiff differential equations	Rosenbrock	Modified second-order Rosenbock formulation.
ode23t	moderately stiff differential equations and differential algebraic equations	trapezoid rule	Useful if you need a solution without numerical damping.
ode23tb	stiff differential equations	TR–BDF2	This solver uses an implicit Runge–Kutta formula with the trapezoid rule (TR) and a second-order backward differentiation formula (BDF2).
ode15i	fully implicit differential equations	BDF	This solver uses a backward difference formula (BDF) to solve implicit differential equations of the form $f(y, y', t) = 0$.