Brief Discussion of Numerical Integration of ODEs

The Forward Euler Method

Differential equation $\frac{dy}{dt} = f(t, y)$

Remember definition of the derivative: $\frac{dy}{dt} = \lim_{h \to 0} \left(\frac{y(t+h) - y(t)}{h} \right)$

Approximate derivative using a small, but finite, value of *h* : $\frac{dy}{dt} \approx \frac{y(t+h) - y(t)}{h}$

The differential equation gives us the value of *dy*/*dt*, so substitute: $f(t, y) \approx \frac{y(t+h) - y(t)}{h}$

Rearrange:

$$y(t+h) \approx y(t) + h \cdot f(t, y)$$

Starting from an initial *t* and *y*, take small time steps of length *h*, updating *t* and *y* as we go along

The Forward Euler Method

Initial condition $y = y_0$ at $t = t_0$ Time step h

Calculate y values y_1, y_2, y_3, y_4, … at times $t_1 = t_0 + h$, $t_2 = t_0 + 2h$, $t_3 = t_0 + 3h$, …
using $$y_{j+1} = y_j + h f\left(t_j, y_j\right)$$

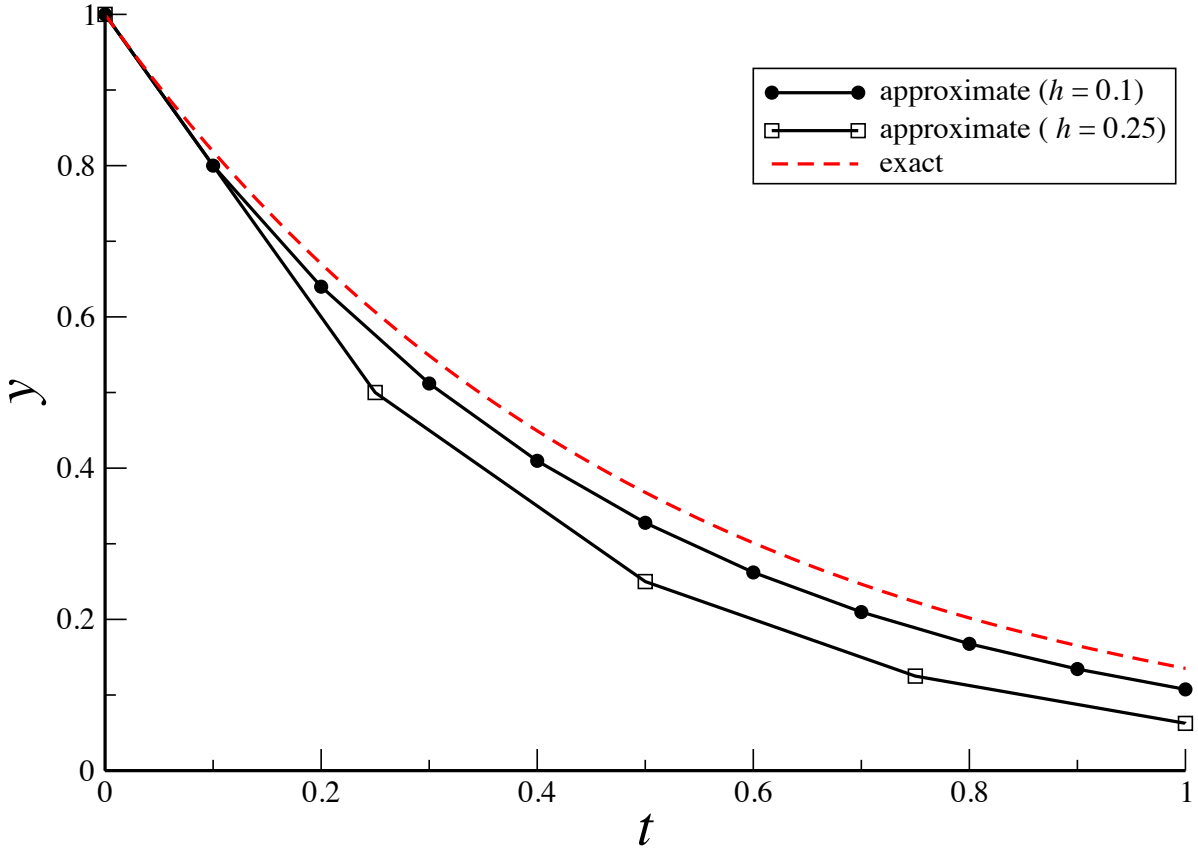
An "explicit" method: have an explicit formula for y_{j+1}

Example: find an approximate solution of $\frac{dy}{dt} = -2y$, with $y(0) = 1$, using $h = 0.25$

(why use this example?)

j	t_j	y_j	$f(t_j, y_j)$
0	0	1	-2
1	0.25	$1 + 0.25(-2) = 0.5$	-1
2	0.5	$0.5 + 0.25(-1) = 0.25$	-0.5
3	0.75	$0.25 + 0.25(-0.5) = 0.125$	-0.25
4	1	$0.125 + 0.25(-0.25) = 0.0625$	-0.125
5	1.25	$0.0625 + 0.25(-0.125) = 0.03125$	

Forward Euler Example: smaller step size

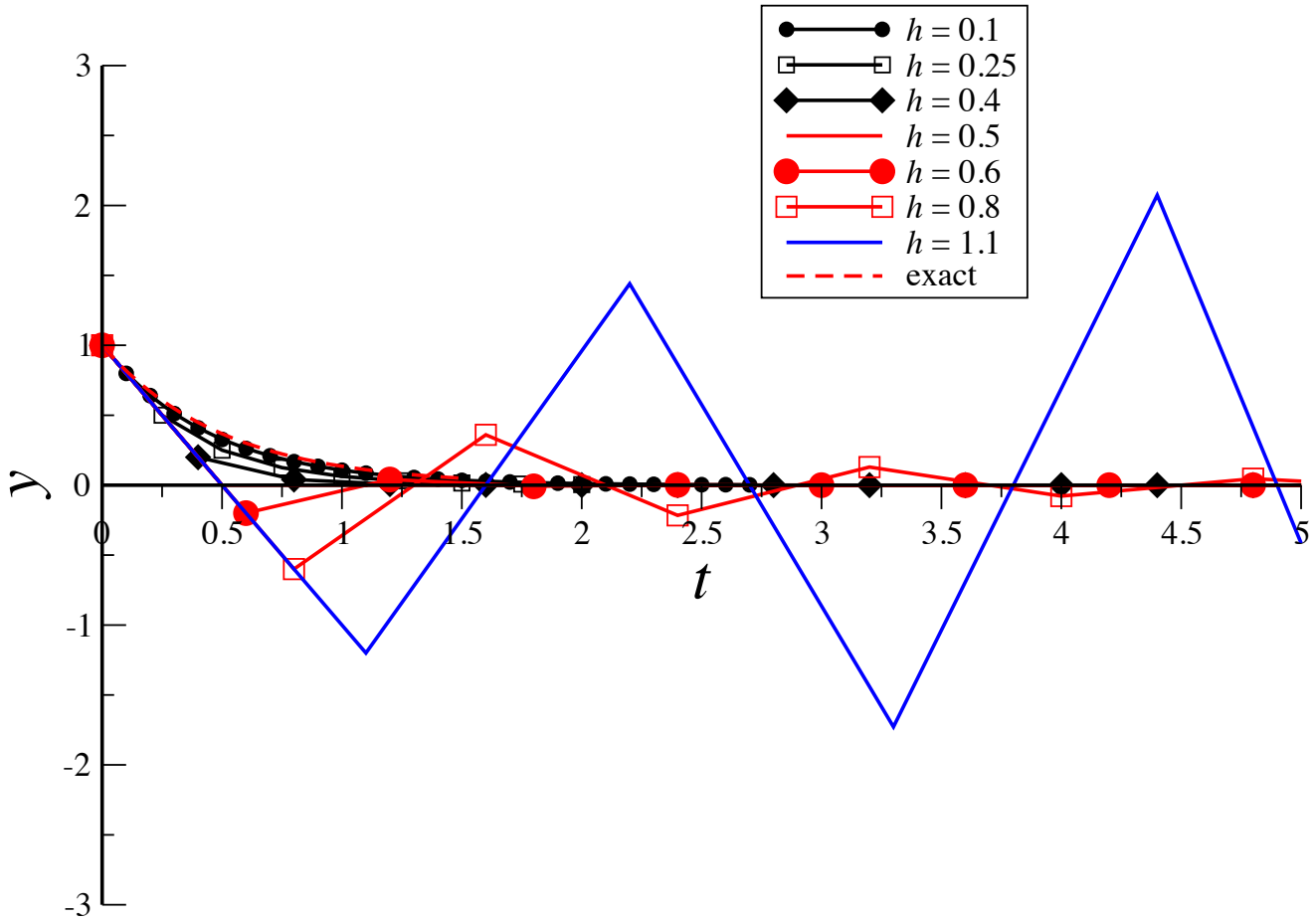


We get a better approximation using a smaller step size

However, there is a systematic underestimate of the true solution in both approximate curves. Why?

Problems with Forward Euler

Forward Euler method is badly behaved if the step size becomes too large



Numerical instabilities … Forward Euler is a good method to explain idea behind numerical integration, but not a great method to use in practice

Improvements on Forward Euler

1. Tolerance checks
Smaller step size improved accuracy, but increases computation time.
What is the appropriate step size?
Could compare prediction with that obtained using a smaller step size or a different method: if they are "close enough" then our step size is small enough, otherwise reduce it

2. Variable (adaptive) step size
Step size needs to be small if $f(t, y)$ changes rapidly
Forward Euler doesn't require h to be fixed, so we could take smaller or larger steps, depending on how rapidly $f(t, y)$ is changing at a particular value of y

3. Higher order techniques
Forward Euler uses a linear approximation to obtain the next value of y. Could use a quadratic (or higher order) approximation

4. Base the method on a different formula for the derivative
Forward Euler uses a "forward difference" approximation to dy/dt. Could use one of several alternatives (e.g. "backward difference")

The Backward Euler Method

Alternative formula for derivative:

$$\left.\frac{dy}{dt}\right|_t = \lim_{h\to 0}\left(\frac{y(t)-y(t-h)}{h}\right)$$

Approximate derivative using a small, but finite, value of *h* :

$$\frac{dy}{dt} \approx \frac{y(t-h)-y(t)}{h}$$

The differential equation gives us the value of *dy*/*dt*, so substitute:

$$f\left(t, y(t)\right) \approx \frac{y(t-h)-y(t)}{h}$$

Rearrange:

$$y(t) \approx y(t-h) + h \cdot f\left(t, y(t)\right)$$

$$\boxed{y(t+h) \approx y(t) + h \cdot f\left(t+h, y(t+h)\right)}$$

Notice: value of *y* inside the function *f* is its value at the **end** of the time step – this is the "new" value of *y*

We no longer get an explicit formula for *y*. This is an **implicit** method: at each time step we get an equation that has to be solved for the "new" value of *y*

Implicit methods are more difficult to implement, but are better behaved numerically

Numerical Analysis

Numerical integration of differential equations is a big subject: whole books, entire courses, entire academic careers, …

We won't worry too much about details here. We will typically use pre-packaged numerical integration routines (e.g. web applets or routines in MATLAB) that others have written for us

What is important for us is having some idea of what is involved **and some idea of the potential pitfalls**
(e.g. numerical instability – view numerical results with some caution)

ODE Solvers in MATLAB

Nice article at
https://blogs.mathworks.com/loren/2015/09/23/ode-solver-selection-in-matlab/

discusses MATLAB's ODE solvers:

ode45 , ode23 , ode113 , ode15s , ode23s , ode23t , ode23tb

and some reasons why you might choose one over the other

Talks about naming system: ode45 uses both a fourth order and a fifth order Runge-Kutta method, providing an estimate of error.

Variable step-length employed, guided by error estimate.

Discusses **stiff systems** and that certain routines (ode15s, ode23s, ode23t, ode23tb) are better suited for these

Very short version of guidance says ode45, but ode15s for stiff [see article for additional comments/recommendations]