Lecture # 23

03/06/2023

Idea: Store arrays:

t	y
$t_0 = 0$	y_0 (given)
$t_1 = \Delta t$	y_1
$t_2 = 2\Delta t$	y_2
$\vdots$	$\vdots$
$t_{final} = n\Delta t = T$	y_n

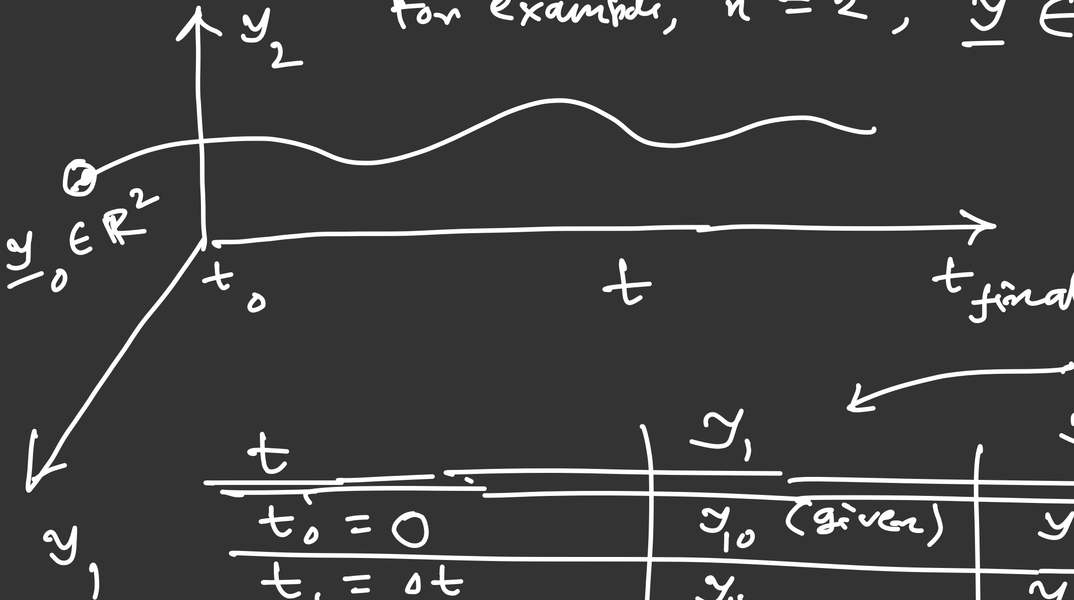
We can also have $\underline{y}$ as a vector:

$$\left.\frac{d\underline{y}}{dt} = \underline{f}(t, \underline{y}), \quad \underline{y}(t_0) = \underline{y}_0, \right\} \text{vector ODE IVP}$$

given

$\underline{y} \in \mathbb{R}^n$

For example, $n = 2$, $\underline{y} \in \mathbb{R}^2$



first component of $\underline{y}$

second " " "

t	y_1	y_2
$t_0 = 0$	y_{10} (given)	y_{20} (given)
$t_1 = \Delta t$	y_{11}	y_{21}
$t_2 = 2\Delta t$	y_{12}	y_{22}
$\vdots$	$\vdots$	$\vdots$
$t_{final} = n\Delta t = T$	y_{1n}	y_{2n}

Example: Any higher order ODE can be re-written as a system of first order ODEs:

$$\frac{d^2 y}{dt^2} = t + 3\frac{dy}{dt} + \underbrace{y\frac{dy}{dt}}_{\text{nonlinear}}$$

This is 2^{nd} order ODE because the highest order of derivative is 2

$$\boxed{\text{Let } \begin{aligned} y_1 &:= y \\ y_2 &:= \frac{dy}{dt} \end{aligned}}$$

$$\Leftrightarrow \frac{dy_1}{dt} = y_2 = f_1(t, y_1, y_2)$$

$$\frac{dy_2}{dt} = t + 3y_2 + y_1 y_2 = f_2(t, y_1, y_2)$$

Then define: $\underline{y} := \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \in \mathbb{R}^2$

$$\Leftrightarrow \frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ t + 3y_2 + y_1 y_2 \end{pmatrix}$$

$$\Leftrightarrow \frac{d}{dt}\underline{y} = \underline{f}(t, \underline{y})$$ ← 2×1 vector first order ODE

2×1 vectors

next pg.

Numerically solving ODE IVPs in computer:

$$\frac{d\underline{y}}{dt} = \underline{f}(t, \underline{y}(t)), \quad \underline{y}(t_0) = \underline{y}_0 \text{ (given)}$$

$$\underbrace{\frac{d\underline{y}}{dt}}\ \approx \underbrace{\frac{\underline{y}_{k+1} - \underline{y}_k}{\Delta t}}_{\substack{\text{forward difference} \\ \text{approximation of} \\ \frac{d\underline{y}}{dt}}} + O(\Delta t), \quad \text{where } \underline{y}_k := \underline{y}(t_k) = \underline{y}(t_0 + k\Delta t)$$

for $k = 0, 1, 2, \ldots, n$

$$\Rightarrow \underline{y}_{k+1} = \underline{y}_k + (\Delta t)\,\underline{f}(t_k, \underline{y}_k) + \underbrace{(\Delta t)\,O(\Delta t)}_{O((\Delta t)^2)}$$

Forward Euler approximation

This is an explicit method

If we instead do backward approximation of the derivative :

$$\frac{\underline{y}_{k+1} - \underline{y}_k}{\Delta t} + O(\Delta t) = \underline{f}(t_{k+1}, \underline{y}_{k+1})$$

$$\Rightarrow \underline{y}_{k+1} = \underline{y}_k + (\Delta t)\underline{f}(t_{k+1}, \underline{y}_{k+1}) + \underbrace{(\Delta t) O(\Delta t)}_{O((\Delta t)^2)}$$

Backward Euler approximation

This is an implicit method

Because it is implicit method, we need to call a nonlinear equation solver / algorithm such as:
Newton's method / bisection / fixed pt. recursion

Example:

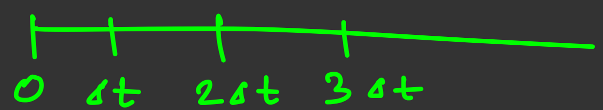
$$\frac{dy}{dt} = \underbrace{-y}_{=:\, f(t,y)}, \qquad y(0) = 1.$$

Exact solution:

$$y(t) = \exp(-t)$$

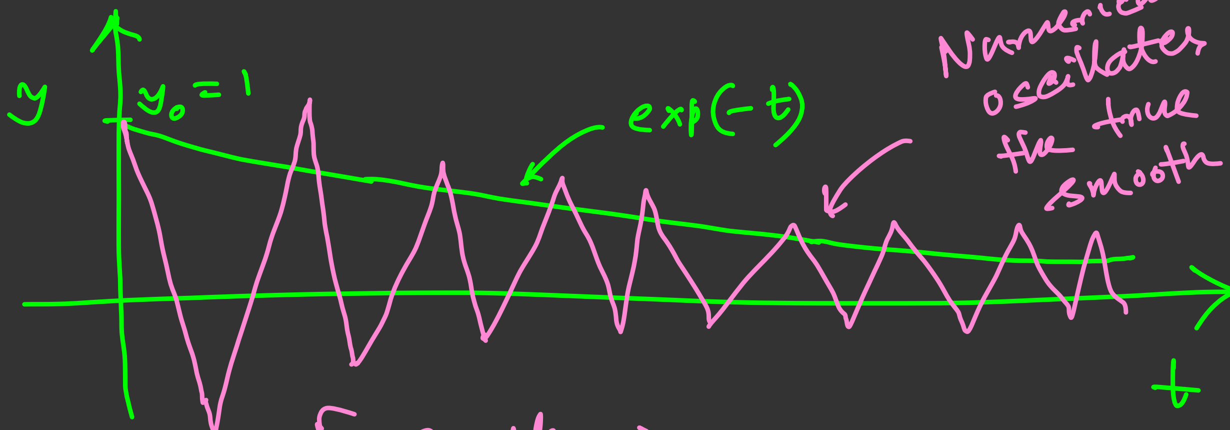
Forward Euler:

$$y_0 = 1$$



$$y_1 = y_0 + (\Delta t)(-y_0)$$
$$= 1 - (\Delta t)$$

$$y_2 = y_1 + (\Delta t)(-y_1)$$
$$= (1 - \Delta t)^2$$
$$\vdots$$
$$y_n = (1 - \Delta t)^n$$



For example, if $\Delta t = 1.5$, then
$$y_n = (-0.5)^n$$

Backward Euler:

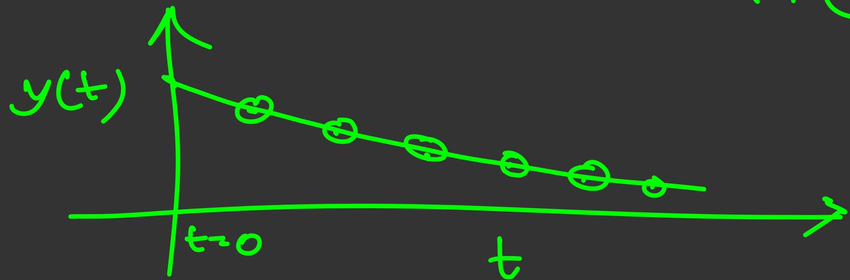
$$y_0 = 1$$

$$y_1 = y_0 - (\Delta t) y_1$$

$$\Rightarrow y_1 = \frac{1}{1 + \Delta t}$$

$\vdots$

$$y_n = \frac{1}{(1+\Delta t)^n}$$



Higher order explicit methods:

Runge-Kutta Methods:

RK2 (Second order Runge-Kutta methods):

IDEA:

$$\underline{y}_{k+1} = \underline{y}_k + \left(a\,\underline{k}_1 + b\,\underline{k}_2\right)$$

where

$$\underline{k}_1 := (\Delta t)\,\underline{f}(t_k, \underline{y}_k)$$

$$\underline{k}_2 := (\Delta t)\,\underline{f}\left(t_k + \alpha\Delta t,\ \underline{y}_k + \beta\,\underline{k}_1\right)$$

assuming that $\underline{y}(t) \in C^2([0,T])$

Notice that if we specialize:

$$a = 1, \quad b = 0$$

then we recover Forward Euler (explicit) method

Now we ask: determine the parameters a, b, α, β such that the truncation error becomes: $O((\Delta t)^3)$.

↳ Detailed derivation: CANVAS Supplementary Notes folder

<u>Final algorithm for RK2:</u>

$$\underline{y}_{k+1} = \underline{y}_k + \frac{1}{2}\left(\underline{k}_1 + \underline{k}_2\right)$$

where

$$\underline{k}_1 = (\Delta t)\, \underline{f}(t_k, \underline{y}_k)$$

$$\underline{k}_2 = (\Delta t)\, \underline{f}(t_k + \Delta t, \underline{y}_k + \underline{k}_1)$$

Similarly, we can derive RK4 (accurate upto 4th order):

determine parameters such that the truncation error becomes $\mathcal{O}((\Delta t)^5)$:

$$\underline{y}_{k+1} = \underline{y}_k + \frac{1}{6}\left(\underline{k}_1 + 2\underline{k}_2 + 2\underline{k}_3 + \underline{k}_4\right)$$

where:

$$\underline{k}_1 := (\Delta t)\,\underline{f}(t_k, \underline{y}_k)$$

$$\underline{k}_2 := (\Delta t)\,\underline{f}\left(t_k + \frac{\Delta t}{2}, \underline{y}_k + \frac{\underline{k}_1}{2}\right)$$

$$\underline{k}_3 := (\Delta t)\,\underline{f}\left(t_k + \frac{\Delta t}{2}, \underline{y}_k + \frac{\underline{k}_2}{2}\right)$$

$$\underline{k}_4 := (\Delta t)\,\underline{f}\left(t_k + \Delta t, \underline{y}_k + \underline{k}_3\right)$$

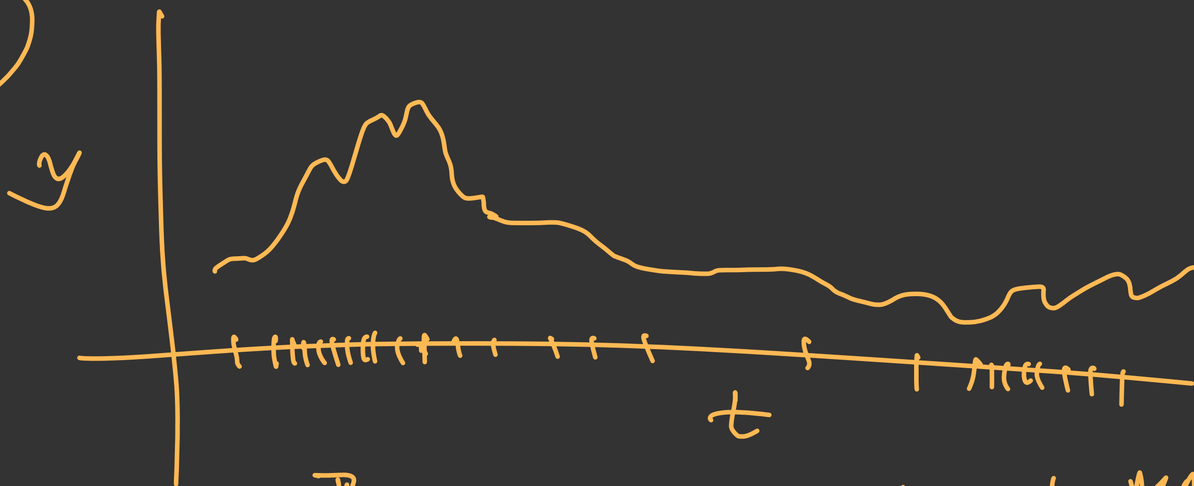
RK4 explicit

MATLAB has an in-built variable step-size fourth order accurate method

truncation error $O((\Delta t)^5)$

ode45

MATLAB command



These step-sizes are chosen by MATLAB adaptively during the execution of ode45