Euler Method for (IVP) $\begin{cases} y' = f(t,y), & t_0 < t < t_{end} \\ y(t_0) = y_0 \end{cases}$

It is the simplest, most basic ODE solver.

Idea: approximate $y'(t_n)$ by Forward Finite Difference:

$$y'(t_n) \approx \frac{y(t_{n+1}) - y(t_n)}{\Delta t}$$ the smaller Δt the better the approximat

so ODE $\Rightarrow$ $\frac{y(t_{n+1}) - y(t_n)}{\Delta t} \approx f(t_n, y(t_n)), \quad n = 0, 1, \ldots, N-1$

$\Rightarrow y(t_{n+1}) \approx y(t_n) + \Delta t \cdot f(t_n, y(t_n)), \quad n = 0 : N-1$

knowing current $Y_n \approx y(t_n)$ can approximate $Y_{n+1} \approx y(t_{n+1})$
starting with IC: $Y_0 = y_0 = y(t_0)$.

Euler scheme: $Y_0 = y_0$

$$Y_{n+1} = Y_n + \Delta t \cdot f(t_n, Y_n), \quad n = 0 : N-1$$

It is 1st order: error $= O(\Delta t)$

It is single-step method, using one f-evaluation per time step.

e.g. $\begin{cases} y' = y, & 0 < t < 1 \\ y(0) = 1 \end{cases}$ Here $f(t,y) = y$, $t_0 = 0$, $y_0 = 1$

Exact solution: $y = e^t$

Apply Euler with $N = 4$: $\Delta t = \frac{1-0}{4} = \frac{1}{4}$, so $t_0 = 0$, $t_1 = \frac{1}{4}$, $t_2 = \frac{2}{4} = \frac{1}{2}$, $t_3 = \frac{3}{4}$, $t_4 = 1$

$Y_0 = y_0 = 1$

$Y_1 = Y_0 + \Delta t \cdot f(t_0, Y_0) = 1 + \frac{1}{4} \cdot 1 = \frac{5}{4} \approx y(\frac{1}{4})$

$Y_2 = Y_1 + \Delta t \cdot f(t_1, Y_1) = \frac{5}{4} + \frac{1}{4} \cdot \frac{5}{4} = \frac{25}{16} = 1.5625 \approx y(\frac{1}{2})$

$Y_3 = Y_2 + \Delta t \cdot f(t_2, Y_2) = \frac{25}{16} + \frac{1}{4} \cdot \frac{25}{16} = \frac{25}{16}(1 + \frac{1}{4}) = \frac{125}{64} = 1.953125 \approx y(\frac{3}{4})$

$Y_4 = Y_3 + \Delta t \cdot f(t_3, Y_3) = \frac{125}{64}(1 + \frac{1}{4}) = \frac{625}{256} = 2.44140625 \approx y(1) = e$

error at $t_4 = 1$: $y(1) - Y_4 \approx 0.2769$

Taking $N = 8$, $\Delta t = \frac{1}{8}$: $Y_8 = 2.5658$, error $= |y(1) - Y_8| \approx 0.1525$ (roughly half of previou

Forward Euler scheme for $\begin{cases} y' = f(t,y), \quad t_0 < t < t_{end} \\ y(t_0) = y_0 \end{cases}$

Algorithm:

choose N = # of timesteps

set $\Delta t = \dfrac{t_{end} - t_0}{N}$

$t_n = t_0 + n \cdot \Delta t$, $n = 0:N$

$Y_0 = y_0$

$Y_{n+1} = Y_n + \Delta t \cdot f(t_n, Y_n)$, $n = 0 : N-1$

Coding:

$f(t,y)$ in subprogram FCN.m :

```
function yprime = FCN(t,y)
    yprime = ... formula for f(t,y) ...
end
```

Main code:

inputs: Nsteps, t0, y0, tend, dtout
print them out

```
%--- set timestep:
    dt = (tend - t0) / Nsteps

%--- initialize:
    tn = t0
    Yn = y0
    errMax = 0
    tout = max( dtout, dt)

%--- timestepping loop:
for n = 1:Nsteps
    Yn = Yn + dt*FCN(tn, Yn)
    tn = t0 + n*dt

    errn = abs( Yn - y_exact(tn))  if known
    errMax = max( errMax, errn)
    %--- time to output?
    if ( tn >= tout)
        OUTPUT ( tn, Yn, errMax)
        tout = tout + dtout
    end %if
end %for
if ( tn >= tend)
    print: DONE at time = tn , errMax = ...
end %if
```

Types of ODE solvers (categories)

1. explicit vs implicit

2. single step vs multistep

3. RK - multistep
   Runge-Kutta BDF for stiff

4. symplectic integrators

5. ETD = Exponential Time Differencing schemes

6. PEER methods (RK type, very high order)

7. SSP = Strong Stability Preserving, for Conservation laws

Other simple methods

1. Backward Euler

Using backward finite difference for y':

$$y'(t) \approx \frac{y(t) - y(t-h)}{h} = f(t, y(t))$$

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

$$\Rightarrow \quad Y_{n+1} = Y_n + \Delta t \cdot f(t_{n+1}, Y_{n+1}), \quad n=0,1,\ldots, \quad Y_0 = y(t_0)$$

scheme is <u>implicit</u> ; must solve for Y_{n+1}, at each timestep

Again <u>1st order</u>, but unconditionally absolutely stable, as we'll see...
does better on stiff problems

2. Midpoint method: use centered FD: $y'(t) \approx \frac{y(t+h) - y(t-h)}{2h} = f(t, y(t))$

$$Y_{n+1} = Y_{n-1} + \Delta t \cdot f(t_n, Y_n), \quad n=1,2,\ldots$$

needs Y_{n-1}, Y_n to produce Y_{n+1}: <u>2-step</u> method, <u>not</u> self-starting

<u>Explicit</u>, <u>2nd order</u>

Equivalent integral eqn: $y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(s, y(s))\, ds, \quad n=0,1,\ldots,N$

Approximating the integral by various quadrature rules we get various methods:

a. Rectangle Rule with left heights: $y(t_{n+1}) \approx y(t_n) + \Delta t \cdot f(t_n, y(t_n))$

$\Rightarrow$ Forward Euler

b. Rectangle Rule with right heights: $y(t_{n+1}) \approx y(t_n) + \Delta t \cdot f(t_{n+1}, y(t_{n+1}))$

$\Rightarrow$ Backward Euler

c. Trapezoidal Rule: $y(t_{n+1}) \approx y(t_n) + \frac{\Delta t}{2} \cdot \left[f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1})) \right]$

$$\Rightarrow Y_{n+1} = Y_n + \frac{\Delta t}{2} \left[f(t_n, Y_n) + f(t_{n+1}, Y_{n+1}) \right], \quad n=0:N$$

<u>implicit</u>, <u>single step</u>, <u>2nd order</u>