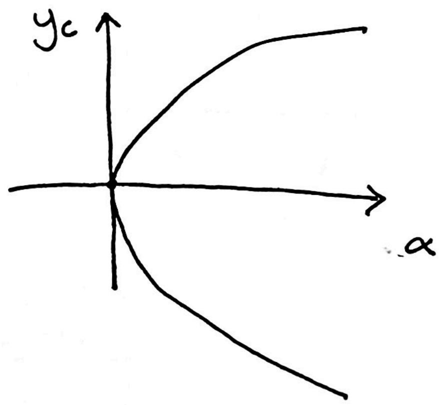
Bifurcations

Points where critical points change - need a parameter in system (for harvesting, that's H).

Ex: $y' = y^2 - \alpha$



Critical Points

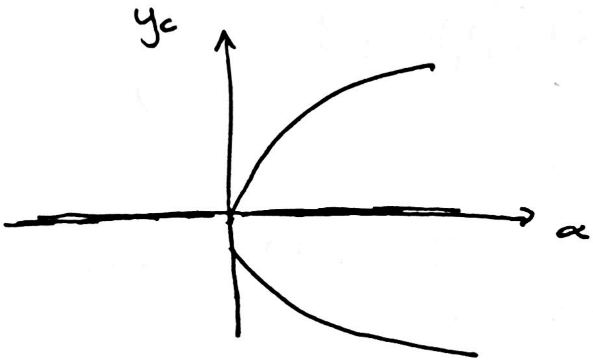
$\alpha < 0 \rightarrow$ none

$\alpha = 0 \rightarrow$ one $\quad y_c = 0$

$\alpha > 0 \rightarrow$ two $\quad y_c = \pm\sqrt{\alpha}$

The critical point at $\alpha = 0$ splits into two for $\alpha > 0$ is called a saddle node bifurcation.

Ex: $y' = y^3 - \alpha y$



Critical points

$\alpha = 0 \rightarrow$ one $\quad (y_c = 0)$

$\alpha < 0 \rightarrow$ one $\quad (y_c = 0)$

$\alpha > 0 \rightarrow$ three

This is called a pitchfork bifurcation.

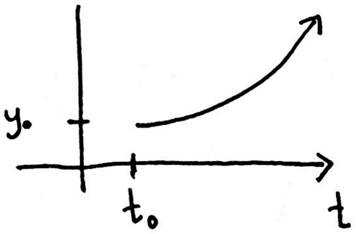
Ex: (Harvesting) $\rightarrow$ saddle-node

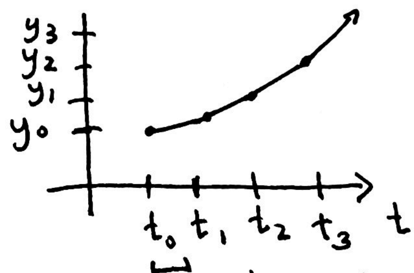
Numerics

Suppose we have:

$$y' = f(t,y) \quad \text{with} \quad y(t_0) = y_0.$$

Now the idea is to build our solution by connecting lines:





$y_i \approx y(t_i)$

h - step size $\quad (h \ll 1$ usually$)$

Algorithm

t_0 given

y_0 given

n steps (given)

for $j = 0:n$

$\quad y_{j+1} =$ some formula to make line (or curve)

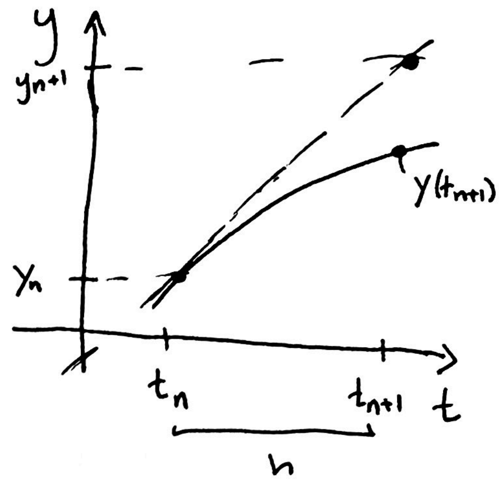
$\quad t_{j+1} = t_j + h$

end.

Note: Could also use while loop:

while $(t_i < T)$ where $T = t_0 + n \cdot h$

Ex: (Forward Euler)



If we zoom in, notice the line can be approximated by the tangent line.

Recall the point-slope formula:

$$y - y_n = m(t - t_n) \quad , \quad m = y'(t_n) = f(t_n, y_n)$$

And plug in (t_{n+1}, y_{n+1}):

$$y_{n+1} = y_n + f(t_n, y_n)(t_{n+1} - t_n) = y_n + h f(t_n, y_n)$$

This is Forward Euler.

$\boxed{\text{Algorithm}}$

Given y_0, t_0, n, h.

for $j = 0:n$

$\quad y_{j+1} = y_j + h f(t_j, y_j)$

~~end~~ $t_{j+1} = t_j + h$

end.

We may use "k-notation" (later):

for $j = a:n$

$\quad k_0 = f(t_j, y_j)$

$\quad y_{j+1} = y_j + h k_0$

$\quad t_{j+1} = t_j + h$

end

Error:

* Local error

$$y(t_{n+1}) = y(t_n) + h\,y'(t_n) + \underbrace{\frac{h^2 y''(t_n)}{2}}_{\text{error}} + \ldots$$

Note that $h^3 \ll h^2$ for small h, so my error is

$$\frac{h^2 y''(t_n)}{2} = O(h^2),$$ call big-O notation. $\left(O(h^2) \le C \cdot h^2\right)$

We can see why Euler has local error of $O(h^2)$:

$$y(t_{n+1}) = y_n + h\,y'(t_n) + \underline{\underline{O(h^2)}}$$
$$\approx y_n + h\,f(t_n, y_n) = y_{n+1}$$

* Cumulative Error

Each step taken, accumulates a local error. So,

Take $t = [0,1]$, $n = \frac{1}{h}$, so h steps taken. Then,

$$err_{[0,1]} \approx \sum_{i=1}^{n} e_i = O(h^2)\,n = \underbrace{O(h)}_{\text{cumulative error for Euler}}$$

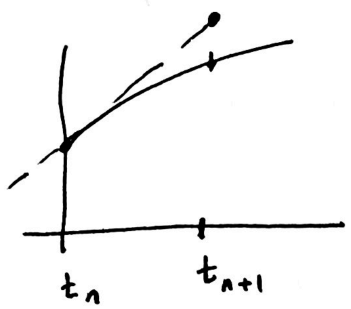
This is the most commonly used metric. As we see, we've lost an ~~of~~ order of accuracy from local error.

cut h in half $\rightarrow$ step twice as much

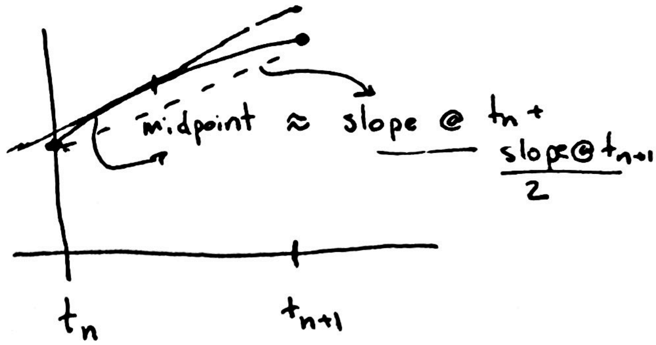
$\downarrow$ $\qquad\qquad\qquad\qquad$ $\downarrow$

half the error $\longrightarrow$ same amount of work

Second Order Schemes



Euler



Improved Euler

$$y_{n+1} = y_n + \frac{h}{2}\left(\text{slope}(t_n) + \text{slope}(t_{n+1})\right) = y_n + \frac{h}{2}(k_0 + k_1)$$

$k_0 = f(y_n, t_n)$ is known

$k_1 = f(y_{n+1}, t_n)$ which we don't know

↳ predict using Euler $y_{n+1} \approx y_n + h f(y_n, t_n) = y_n + h k_0$

$\Rightarrow k_1 = f(y_{n+1}, t_{n+1}) = f(y_n + h k_0, t_{n+1})$.

Algorithm

t_0, y_0, n given

for $j = 0 : n$

$\quad k_0 = f(y_j, t_j)$

$\quad u = y_j + h k_0$

$\quad k_1 = f(u, t_{j+1})$

$\quad y_{j+1} = y_j + \frac{h}{2}(k_0 + k_1)$

$\quad t_{j+1} = t_j + h$

end

Error (Improved Euler)

Local $\sim O(h^3)$

Cumulative $O(h^2)$

Cut h in half $\Rightarrow$ double the work

$\Downarrow$ $\Downarrow$

error down y 4 $\Rightarrow$ overall gain

* Improved Euler gives a relative estimate of error:

$y_{LOW} = y_i + hk_0$ Euler

$y_{HIGH} = y_i + \frac{h}{2}(k_0 + k_1)$ Improved Euler

$|y_{HIGH} - y_{LOW}| \approx$ local error (as $h \ll 1$)

This allows for <u>adaptive</u> schemes.

<u>Idea</u>: Determine step-size h based on estimate of error!

y_{LOW} is called the low order predictor

y_{HIGH} is called the high order corrector.

if $|y_{HIGH} - y_{LOW}| >$ tolerance

↳ cut h

or $|y_{HIGH} - y_{LOW}| <$ really small tolerance

↳ raise h.

$\boxed{\text{Algorithm}}$ (Improved Euler)

for $i = a:n$ (while loop)

$$k_0 = f(t_i, y_i)$$

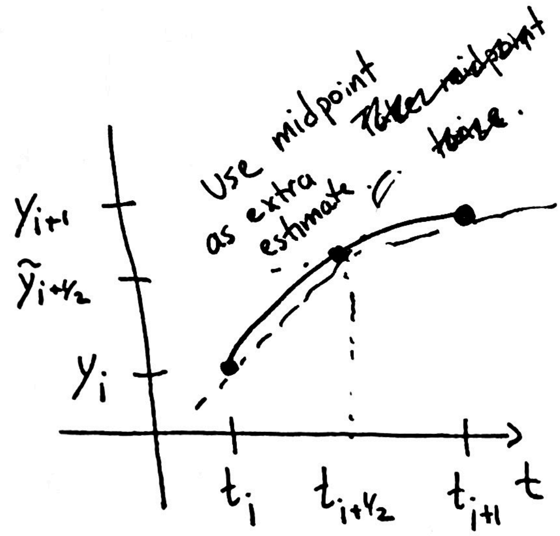
$$u = y_i + h k_0$$

$$k_1 = f(t_{i+1}, u)$$

$$y_{i+1} = y_i + \frac{h}{2}(k_0 + k_1)$$

adaptive
$$\begin{cases} |y_{HIGH} - y_{LOW}| > \text{tolerance} \\ \quad \rightarrow h = h/2 \\ |y_{HIGH} - y_{LOW}| < \text{small tolerance} \\ \quad h = 2h \end{cases}$$

end



RK4 Method

Do this ... twice.

$$k_0 = f(t_i, y_i)$$

$$u_{1/2} = y_i + \frac{h}{2} k_0$$

$$t_{i+1/2} = t_i + \frac{h}{2}$$

$$k_{1/2} = f(t_{i+1/2}, u_{1/2})$$

$$\bar{u}_{1/2} = y_i + \frac{h}{2} k_{1/2}$$

$$\bar{k}_{1/2} = f(t_{i+1/2}, \bar{u}_{1/2})$$

$$u_1 = y_i + h \tilde{k}_{1/2}$$

$$k_1 = f(t_{i+1}, u_1)$$

combining $\longrightarrow$

$$y_{i+1} = y_i + \frac{h}{6}(k_0 + 2k_{1/2} + 2\tilde{k}_{1/2} + k_1)$$

Error (cont'd)

For RK4, local error is $O(h^5)$ and cumulative is $O(h^4)$.

Half $h \Rightarrow$ { double work; error scales $1/16$.

Ex:

	Euler	I.E.	RK4
$h = 0.1$	1e-2	1e-2	2.3e-2
$h = 0.01$	2.1e-3	2.2e-4	1.7e-6
$h = 0.001$	1.1e-4	3e-6	4e-10

Much better error as you go down in h.

Trade-off → more computational work.

ODE45

5 - ~~local~~ global error & order of corrector

↳ i.e. y_{HIGH}

4 - ~~[illegible]~~ global error of predictor

↳ i.e. y_{LOW}

Improved Euler would be ~~[illegible]~~ ODE12, for example

↳ MATLAB doesn't use I.E., but another type of second order.

Other RK4 methods

Another one is the 3/8th's rule:

$$k_0 = f(t_n, y_n)$$
$$k_1 = f(t_n + \tfrac{h}{3}, y_n + k_0/3)$$
$$k_2 = f(t_n + \tfrac{2h}{3}, y_n - \tfrac{k_0}{3} + k_1)$$
$$k_3 = f(t_{n+1}, y_n + k_1 - k_2 + k_3)$$

$$y_{n+1} = y_n + \frac{h}{8}(k_0 + 3k_1 + 3k_2 + k_3)$$

Numerical Stability

Note to students: This is not critical point stability.

Suppose we have the test equation:

$$y' = -ay \quad ; \quad y(0) = y_0 \quad \rightarrow \quad y = y_0 e^{-at} \qquad , a > 0.$$

Now suppose we integrate with Euler:

$$y_1 = y_0 + hf(y_0) = y_0 - ahy_0 = (1-ah)y_0$$
$$y_2 = y_1 + hf(y_1) = y_1 - ahy_1 = (1-ah)y_1 = (1-ah)^2 y_0$$
$$\vdots$$
$$y_n = (1-ah)^n y_0 \quad , \text{ usually written } \quad y_n = z^n y_0.$$

Notice, as $t \to \infty$, $y \to 0$. However, in our scheme, this only happens when $|z| < 1$. This is the idea behind stability.

So, $|z|<1 \Rightarrow -1 < 1-ah < 1$

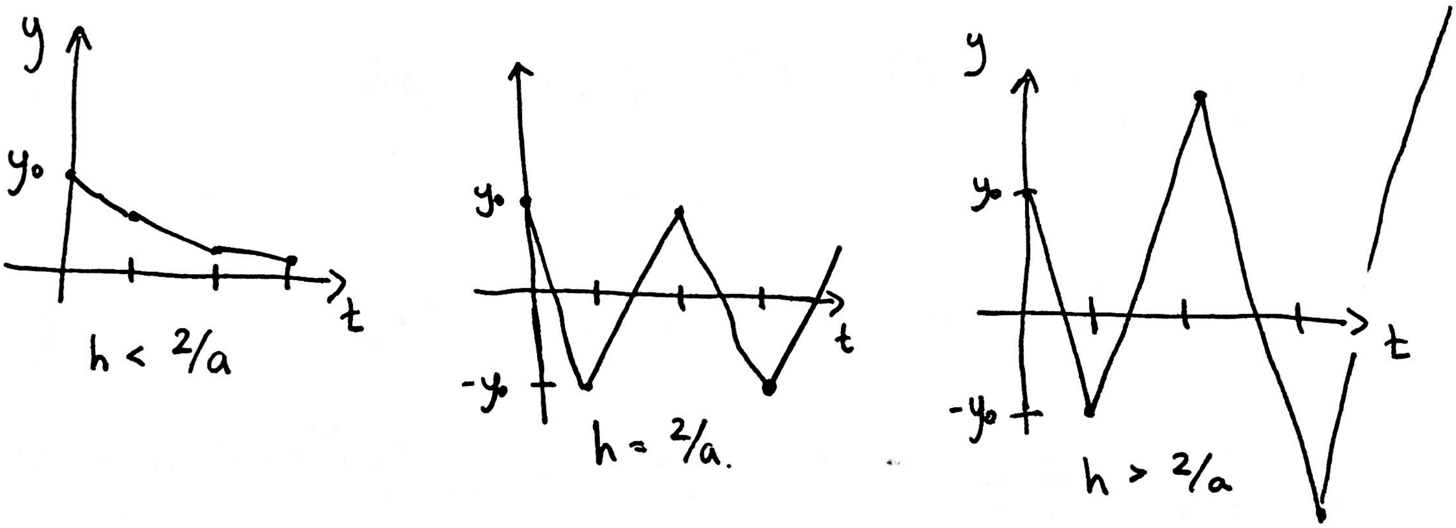
* Notice that $1-ah<1$ always true.

$\rightarrow 1-ah > -1 \rightarrow ah < 2 \rightarrow h < 2/a$. - stability limit or stability bound

If $h = \frac{2}{a}$, then $y_{n+1} = (-1)^n y_0$

If $h < 2/a$, then $y_n \rightarrow 0$

If $h > 2/a$, then $y_n \rightarrow \infty$



All explicit schemes (what we have discussed so far) are subject to numerical instability.

<u>Explicit Schemes</u>:

- Forward Euler
- Improved Euler
- RK4
- ODE45

Implicit Schemes

Before: $y_{n+1} = y_n + hf(t_n, y_n)$

Now: $y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$

This is Backward Euler. The equation is implicit. So, if we want to step, we have to solve for y_{n+1}.

Let's look at the stability. Take $f(t,y) = -ay$ again.

So, $y' = -ay$; $y(0) = y_0$.

$$y_1 = y_0 + hf(y_1) = y_0 - ahy_1 \rightarrow y_1 = \frac{y_0}{1+ah}$$

$$y_2 = y_1 + hf(y_2) = y_0 - ahy_2 \rightarrow y_2 = \frac{y_1}{1+ah} = \left(\frac{1}{1+ah}\right)^2 y_0$$

$$\vdots$$

$$y_n = \left(\frac{1}{1+ah}\right)^n y_0 \quad \text{, so} \quad y_n = Z^n y_0 \quad \text{w/} \quad Z = \frac{1}{1+ah}$$

* Notice $|Z| < 1$ for any choice of $h \rightarrow$ unconditionally stable.

ODE15s is a first order implicit scheme with a fifth order corrector.