RECAP AND FEEDBACK

IMPLICIT VS HEUN'S (EXPLICIT TRAPEZOID) METHOD

Note that the implicit trapezoid method, discussed last session, is not the same as Heun's method (also known as *explicit* trapezoid method)

The implicit trapezoid method is

$$\frac{y(t+\Delta t)-y(t)}{\Delta t} = \frac{1}{2}(g(t,y(t)) + g(t+\Delta t, y(t+\Delta t)))$$

while Heun's method is

$$\frac{y(t+\Delta t)-y(t)}{\Delta t} = \frac{1}{2}(g(t,y(t)) + g(t+\Delta t, \tilde{y}(t+\Delta t)))$$

where $\tilde{y}(t+\Delta t) = y(t) + \Delta t\, g(t,y(t))$ instead of $y(t+\Delta t)$.

So in Heun's method the derivative at the next time step is evaluated at $\tilde{y}$ which in turn is evaluated using forward Euler's method. Heun's method is not a symmetric method. In contrast, the implicit trapezoid method $y(t+\Delta t)$ is used and the equation is re-organized (where possible) to get a explicit relation for $y(t+\Delta t)$.

EXAMPLE DERIVATION FOR IMPLICIT EULER

Assume the DE is

$$\frac{dy(t)}{dt} = g(t, y(t)) = bt - ay(t)$$

Then

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \approx g(t+\Delta t, y(t+\Delta t))$$

with $g(t, y(t)) = bt - ay(t)$, and hence $g(t+\Delta t, y(t+\Delta t)) = b \cdot (t+\Delta t) - a \cdot y(t+\Delta t)$. Therefore

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \approx b \cdot (t+\Delta t) - a \cdot y(t+\Delta t)$$

This equation can be rewritten by isolating $y(t+\Delta t)$ to the LHS. The following steps show how to do this

$$\begin{aligned} y(t+\Delta t) &\approx y(t) + \Delta t \cdot (b \cdot (t+\Delta t) - a \cdot y(t+\Delta t)) \\ y(t+\Delta t)(1 + a \cdot \Delta t) &\approx y(t) + \Delta t \cdot b \cdot (t+\Delta t) \\ y(t+\Delta t) &\approx \frac{y(t) + \Delta t \cdot b \cdot (t+\Delta t)}{(1 + a \cdot \Delta t)} \end{aligned}$$

IMPLEMENTATION IMPLICIT EULER

In a programming language we could implement implicit Euler for this DE as

```
ynext = (ynow + dt*b*(t+dt)) / (1.0 + a*dt)
```

or

```
y[n+1] = (y[n] + dt*b*(t+dt)) / (1.0 + a*dt)
```

TODAY

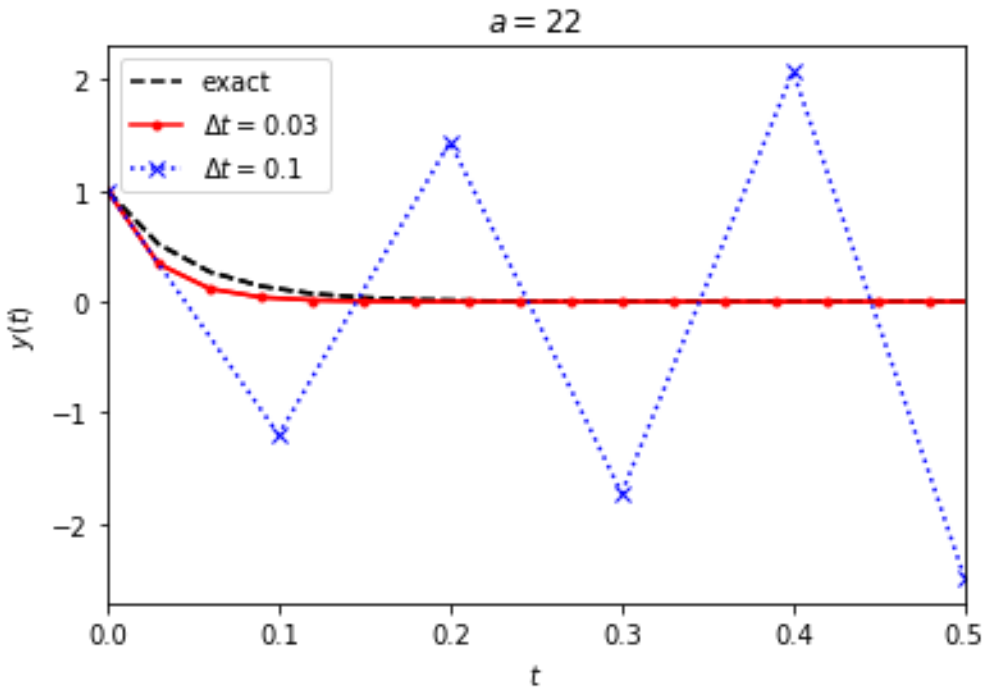
- Stability of i) ODEs and of ii) methods for solving ODEs
- System of ODEs

STABILITY

We saw before that Euler forward becomes unstable for large integration time step.

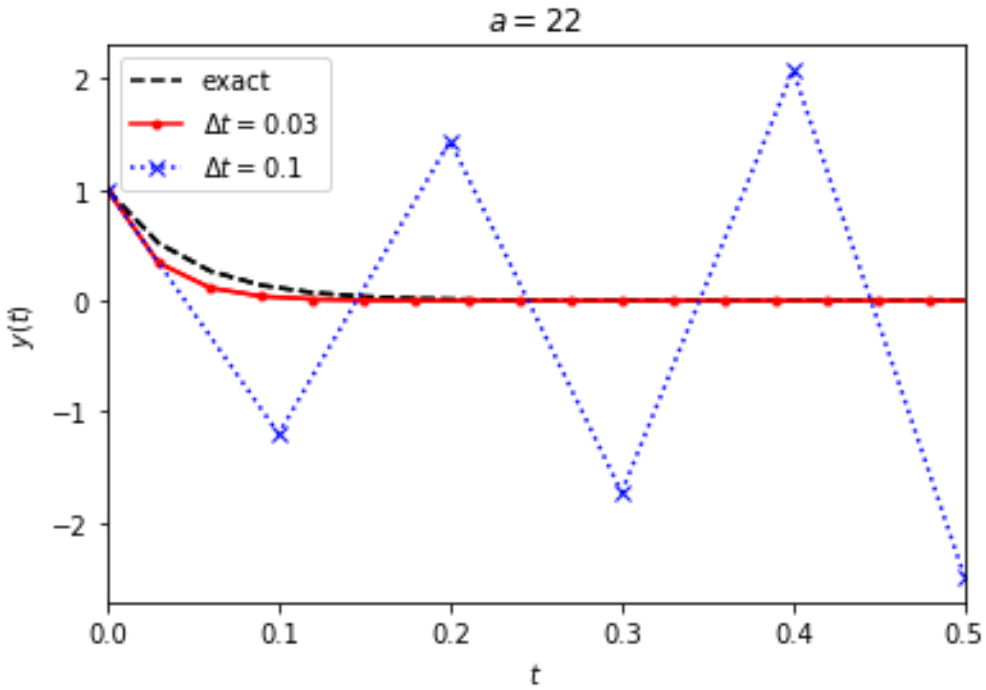
STABILITY

We saw before that Euler forward becomes unstable for large integration time step.



STABILITY

We saw before that Euler forward becomes unstable for large integration time step.



For $\Delta t = 0.03$ all seems fine. For $\Delta t = 0.1$ huge errors. What is the reason? What is stability?

STABILITY OF ODE

Before discussing the stability of the *method*, we need to discuss the stability of an *ODE*.

Assume that $y_A(t)$ and $y_B(t)$ are both solutions of an ODE, but with slightly different initial conditions.

An ODE in itself is stable when given any ϵ, there is a $\delta(\epsilon) > 0$ such that when

$$|y_A(t_0) - y_B(t_0)| \leq \delta$$

then

$$|y_A(t) - y_B(t)| \leq \epsilon \text{ for all } t > t_0.$$

So an ODE is *stable* when *small errors in the initial condition are not amplified*.

EXAMPLE UNSTABLE ODE

The ODE $\dot{y} = y$ has the solution is $y = y_0 \exp(t)$. Consider now two solutions $y_A(t) = y_{0,A} \exp(t)$ and $y_B(t) = y_{0,B} \exp(t)$. Then the difference equals

$$\begin{aligned} |y_A(t) - y_B(t)| &= |y_{0,A} - y_{0,B}||\exp(t)| \\ &= |y_{0,A} - y_{0,B}| \exp(t) \end{aligned}$$

and the latter keeps increasing with time for $y_{0,A} \neq y_{0,B}$, so there is no upper bound. So no δ exists that satisified the aforementioned condition.

Hence this ODE is *unstable*.

EXAMPLE STABLE ODE

The ODE $\dot{y} = -y$ has the solution $y = y_0 \exp(-t)$. Consider now two solutions $y_A(t) = y_{0,A} \exp(-t)$ and $y_B(t) = y_{0,B} \exp(-t)$. Then the difference equals

$$|y_A(t) - y_B(t)| = |y_{0,A} - y_{0,B}| \exp(-t)$$

and the latter goes to zero with time $t \to \infty$ and is finite for $t \geq 0$, so one could simply take $\delta = \epsilon$.

For example, if difference in solution has to deviate at much by 0.01 for all times $t > 0$, then $\epsilon = 0.01$. Hence if one takes $\delta(\epsilon) = \epsilon = 0.01$ this is satisfied.

A difference in the initial condition will be attenuated (damped) and hence this ODE is *stable*.

STABILITY OF METHOD

Assume we have a numerical method, say T.

The numerical method T, for some ODE with initial/boundary conditions, produces a solution. For example, for $\dot{y} = -y$ with $y(0) = 1$, it produces $y_T(t)$ (where T is the method). This is, generally different from the exact solution $y(t)$.

STABILITY OF METHOD

Assume we have a numerical method, say T.

The numerical method T, for some ODE with initial/boundary conditions, produces a solution. For example, for $\dot{y} = -y$ with $y(0) = 1$, it produces $y_T(t)$ (where T is the method). This is, generally different from the exact solution $y(t)$.

DEFINITION

Given two initial conditions that vary slightly, $y_A(0)$ and $y_B(0) = y_A(0) + \epsilon$, the numerical method for a *stable ODE* said be be **stable** if the difference between the two resulting solutions $y_{T,A}(t)$ and $y_{T,B}(t)$ stays bounded, i.e., there is an $M(\epsilon)$ such that $|y_{T,A}(t) - y_{T,B}(t)| < M$ for all $t > 0$.

Hence a stable method means that *errors in the computation of the solution of a stable ODE are not amplified*.

The method is **unstable** if it produces an unbounded solution for a stable ordinary differential equation.

STABILITY OF EULER METHODS

What about our Euler methods for the differential equation

$$\frac{dy(t)}{dt} = -ay(t)$$

with $a > 0$?

We know that this ODE is stable.

FORWARD

The explicit (forward) Euler (FE) method is **conditionally stable**. It becomes **unstable** for $a\Delta t > 2$; the numerical solution then runs away from the actual solution.

The reason is that the numerical scheme is

$$y_{FE}(t_0 + \Delta t) = (1 - a\Delta t)y(t_0)$$

and applying it repeatedly for 2 steps

$$y_{FE}(t_0 + 2\Delta t) = (1 - a\Delta t)y(t_0 + \Delta t) = (1 - a\Delta t)^2 y(t_0)$$

and hence for n steps

$$y_{FE}(t_0 + n\Delta t) = (1 - a\Delta t)^n y(t_0)$$

and this diverges for $n \to \infty$ when $|1 - a\Delta t| > 1$, which implies (since $a > 0$) that $a\Delta t > 2$. Then for two different initial conditions we have $|y_{FE,A}(t) - y_{FE,B}(t)| = |y_A(t_0) - y_B(t_0)||1 - a\Delta t|^n \to \infty$ for $n \to \infty$: it is always larger than any finite value M and is therefore unbounded.

BACKWARD

The implicit (backward) Euler (BE) method is **unconditionally stable** for this stable ODE.

The reason is that the scheme is

$$y_{BE}(t+\Delta t) = y_{BE}(t)/(1+a\Delta t)$$

and applying it repeatedly for 2 steps

$$y_{BE}(t+2\Delta t) = y_{BE}(t+\Delta t)/(1+a\Delta t) = y_{BE}(t)/(1+a\Delta t)^2$$

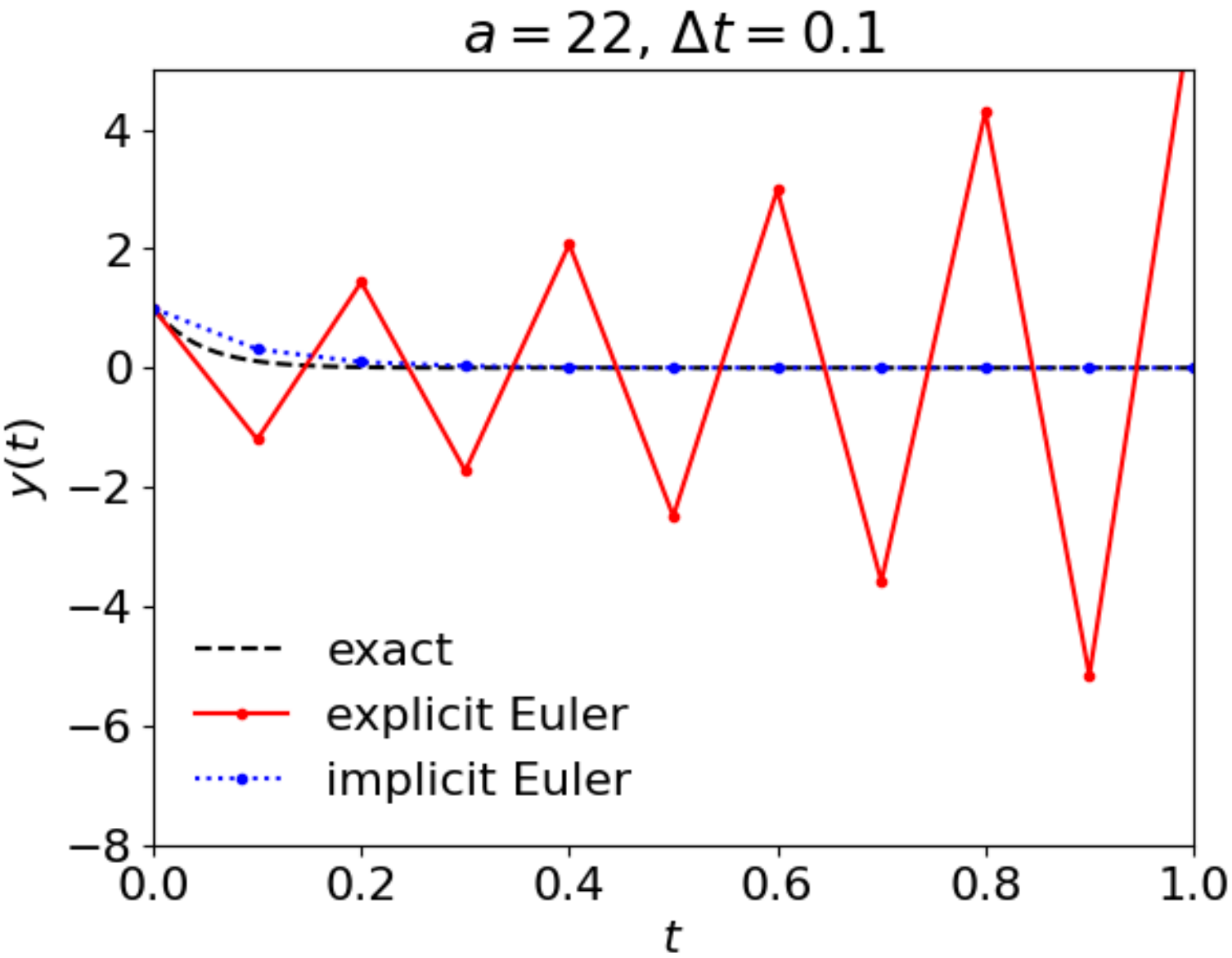
and hence for n steps

$$y_{BE}(t+n\Delta t) = y_{BE}(t)/(1+a\Delta t)^n$$

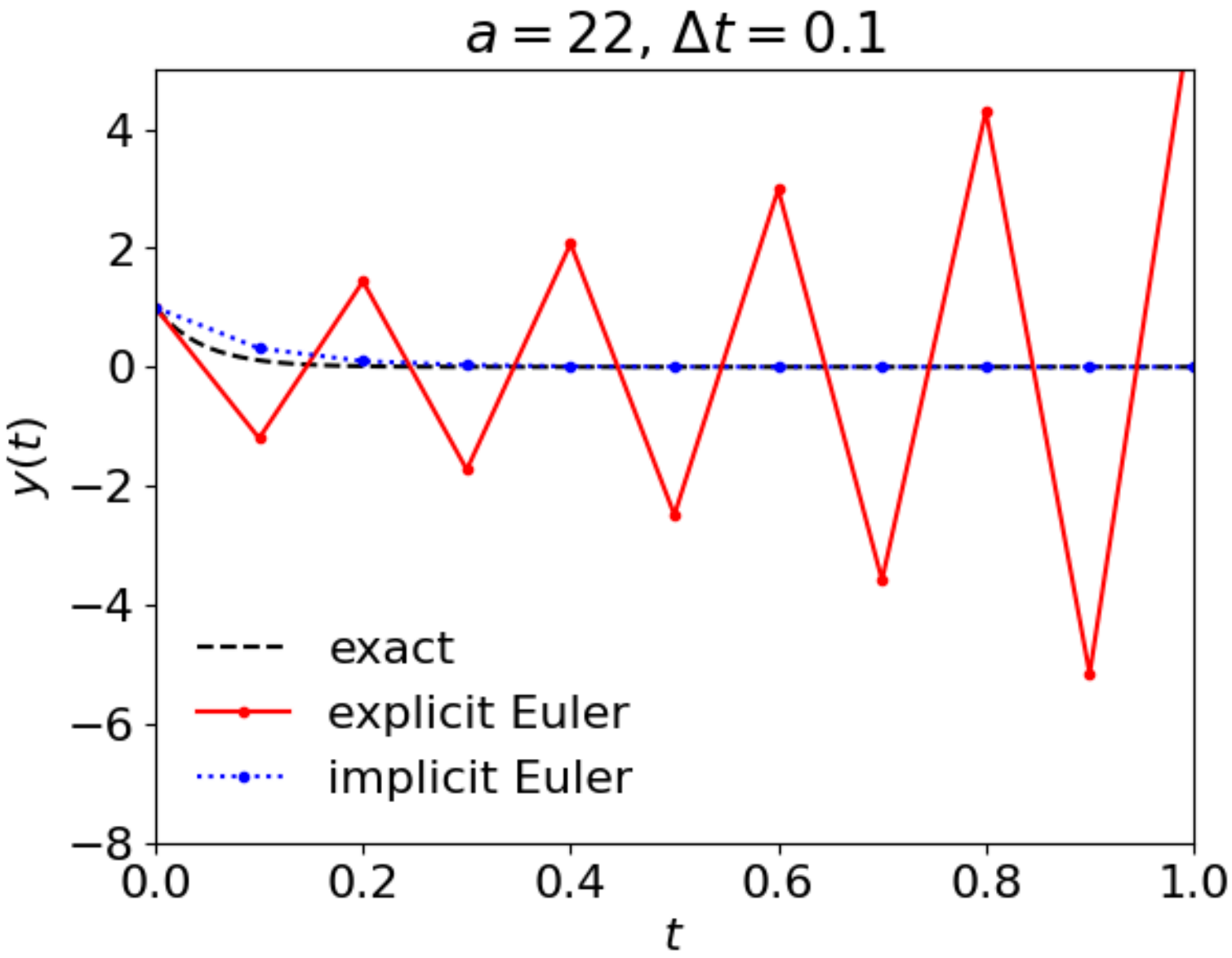
and since $a > 0$, $(1+a\Delta t) > 1$, $1/(1+a\Delta t) < 1$ and hence it always goes to zero for $n \to \infty$, irrespective of Δt (hence *unconditionally*). Because this goes to zero, the difference between two different initial conditions $|y_{BE,A}(t) - y_{BE,B}(t)| = |y_A(t_0) - y_B(t_0)|/(1+a\Delta t)^n$ goes to zero for any $\Delta t > 0$ and is therefore bounded and hence unconditionally stable.

EXAMPLE STABILITY

EXAMPLE STABILITY



EXAMPLE STABILITY



This shows that for $a \cdot \Delta t = 2.2$ the explicit Euler method is unstable (errors grow), but the implicit Euler method is stable (errors don't grow).

CONVERGENCE

DEFINITION

A numerical algorithm is said to **converge**, when continually refined (more terms or smaller steps) yields a sequence of approximate solutions converging to the exact solution.

So when $\Delta t \to 0$ we would approach the true solution for a converging algorithm.

In convergence proofs only the truncation error is considered, round-off errors are discarded.

CONVERGENCE

DEFINITION

A numerical algorithm is said to **converge**, when continually refined (more terms or smaller steps) yields a sequence of approximate solutions converging to the exact solution.

So when $\Delta t \to 0$ we would approach the true solution for a converging algorithm.

In convergence proofs only the truncation error is considered, round-off errors are discarded.

ORDER OF CONVERGENCE

The **order** of an algorithm is the rate at which the *global error* decreases as the grid spacing or size approaches zero.

[//]: # (From Pinchover)

LAX EQUIVALENCE THEOREM

It can be shown that a numerical method **converges** when the following two conditions are both met:

1. The method is *stable*
2. The method is **consistent**: the *exact solution* of the ODE satisifies the numerical method in the limit where the step size (say Δt) goes to zero.

This is also known as the **Lax Equivalence theorem**: a finite difference method for a well-posed initial-boundary value problem converges if it is both consistent and stable (proof of the theorem outside current scope, see here if you are interested)

REMARK

The reason to split *convergence* in these two conditions is that *stability* and *consistency* are generally easier to proof than convergence.

EXAMPLES

Both the explicit and implicit Euler methods are consistent. Hence they will converge for sufficiently small time step (for avoiding the stability issues).

STABILITY EXPLICIT RK METHODS

The explicit Runge Kutta methods are only conditionally stable.

One should be aware of this and always take a sufficiently small integration step.

STABILITY SYMMETRIC RICHARDSON METHOD

DESCRIPTION METHOD

We had for explicit Euler:

$$\frac{y_{i+1} - y_i}{\Delta t} \approx g(t_i, y_i)$$

and implicit Euler

$$\frac{y_{i+1} - y_i}{\Delta t} \approx g(t_{i+1}, y_{i+1})$$

What happens if we try to make a symmetric method? One way:

$$\frac{y_{i+1} - y_{i-1}}{2\Delta t} \approx g(t_i, y_i)$$

This is called the Richardson method.

STABILITY

One can proof that the Richardson method is **unconditionally unstable** (i.e., for any timestep it is unstable) for ODEs such as $\dot{y} = -y$. (See also Pinchover et al.) So this method is not very useful.

The implicit trapezoid rule, on the other hand, is unconditionally stable.

SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS

Up to now we only considered a single ODE, $\frac{dy}{dt} = g(t, y)$.

Many practical applications are actually coupled ODEs, such as two coupled ODEs:

$$\frac{dy_1}{dt} = g_1(t, y_1, y_2)$$
$$\frac{dy_2}{dt} = g_2(t, y_1, y_2)$$

or in general for n coupled ODEs

$$\frac{dy_1}{dt} = g_1(t, y_1, y_2, \ldots, y_n)$$
$$\vdots$$
$$\frac{dy_n}{dt} = g_n(t, y_1, y_2, \ldots, y_n)$$

SOLVING A SYSTEM OF ODES NUMERICALLY

EXAMPLE

Assume we have the following system of ODEs:

$$z_1(t)' = z_2(t)$$
$$z_2(t)' = (1+4t)\sqrt{z_1(t)} - z_2(t)$$

Consider explicit Euler: replace derivative by forward differences

$$\frac{z_1(t+\Delta t) - z_1(t)}{\Delta t} \approx z_2(t)$$
$$\frac{z_2(t+\Delta t) - z_2(t)}{\Delta t} \approx (1+4t)\sqrt{z_1(t)} - z_2(t)$$

Moving all terms depending on $t + \Delta t$ to the LHS, and the rest to the RHS gives

$$
\begin{aligned}
z_1(t + \Delta t) &\approx z_1(t) + \Delta t\ z_2(t) \\
z_2(t + \Delta t) &\approx z_2(t) + \Delta t \left((1 + 4t) \sqrt{z_1(t)} - z_2(t) \right)
\end{aligned}
$$

or in terms of index notation $z_{m,i} = z_m(t_i)$

$$
\begin{aligned}
z_{1,i+1} &\approx z_{1,i} + \Delta t\ z_{2,i} \\
z_{2,i+1} &\approx z_{2,i} + \Delta t \left((1 + 4t_i) \sqrt{z_{1,i}} - z_{2,i} \right)
\end{aligned}
$$

This can also be written as

$$
\begin{pmatrix} z_{1,i+1} \\ z_{2,i+1} \end{pmatrix} \approx \begin{pmatrix} z_{1,i} \\ z_{2,i} \end{pmatrix} + \Delta t \begin{pmatrix} z_{2,i} \\ (1 + 4t_i) \sqrt{z_{1,i}} - z_{2,i} \end{pmatrix}
$$

GENERAL METHOD 2D

For a general system of ODEs this becomes (using y instead of z)

$$\frac{d}{dt}\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} g_1(t, y_1, y_2) \\ g_2(t, y_1, y_2) \end{pmatrix}$$

or in vector notation

$$\frac{d}{dt}\mathbf{y}(t) = \mathbf{g}(t, \mathbf{y}(t))$$

where $\mathbf{y}(t)$ and $\mathbf{g}(t)$ are vectors given by

$$\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$$

and

$$\mathbf{g}(t, \mathbf{y}) = \begin{pmatrix} g_1(t, y_1, y_2) \\ g_2(t, y_1, y_2) \end{pmatrix}$$

For the system of ODEs

$$\frac{d}{dt}\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} g_1(t, y_1, y_2) \\ g_2(t, y_1, y_2) \end{pmatrix}$$

the explicit Euler algorithm becomes

$$\begin{pmatrix} y_{1,i+1} \\ y_{2,i+1} \end{pmatrix} \approx \begin{pmatrix} y_{1,i} \\ y_{2,i} \end{pmatrix} + \Delta t \begin{pmatrix} g_{1,i} \\ g_{2,i} \end{pmatrix}$$

where $g_{m,i} = g_m(t_i, y_{1,i}, y_{2,i})$.

Again, in vector notation this becomes

$$\mathbf{y}_{i+1} \approx \mathbf{y}_i + \Delta t\, \mathbf{g}(t_i, \mathbf{y}_i)$$

GENERAL SCHEME SYSTEM OF ODES

Compare the explicit Euler method for an ODE for a single variable $y(t)$

$$y_{i+1} \approx y_i + \Delta t \; g(t_i, y_i)$$

with the explicit Euler method for a system of variables $y_n(t)$

$$\mathbf{y}_{i+1} \approx \mathbf{y}_i + \Delta t \; \mathbf{g}(t_i, \mathbf{y}_i)$$

Note that they are exactly the same! Also for the other explicit methods one can just *use the same algorithm but then with vector functions instead of scalar functions*.

Hence it is relatively straightforward to generalize the aforementioned explicit methods (such as the RK family) to a system of differential equations.

EXAMPLE: PREDATOR-PREY

Consider the following pair of ODEs

$$\frac{dx(t)}{dt} = a\,x(t) - b\,x(t)y(t)$$
$$\frac{dy(t)}{dt} = -c\,y(t) + d\,x(t)y(t)$$

where x and y are the number of prey and predators, respectively; a the prey growth rate, c the predator death rate, and b and d characterizing the effect of the predator-prey interaction on prey death and predator growth. A specific example would be wolves (predators) and rabbits (preys).

This can be written as

$$\frac{d}{dt}\mathbf{y}(t) = \mathbf{g}(t, \mathbf{y}(t))$$

with $y_1(t) = x(t)$, $y_2(t) = y(t)$, $g_1(t) = ax(t) - bx(t)y(t)$, and $g_2(t) = -cy(t) + dx(t)y(t)$.

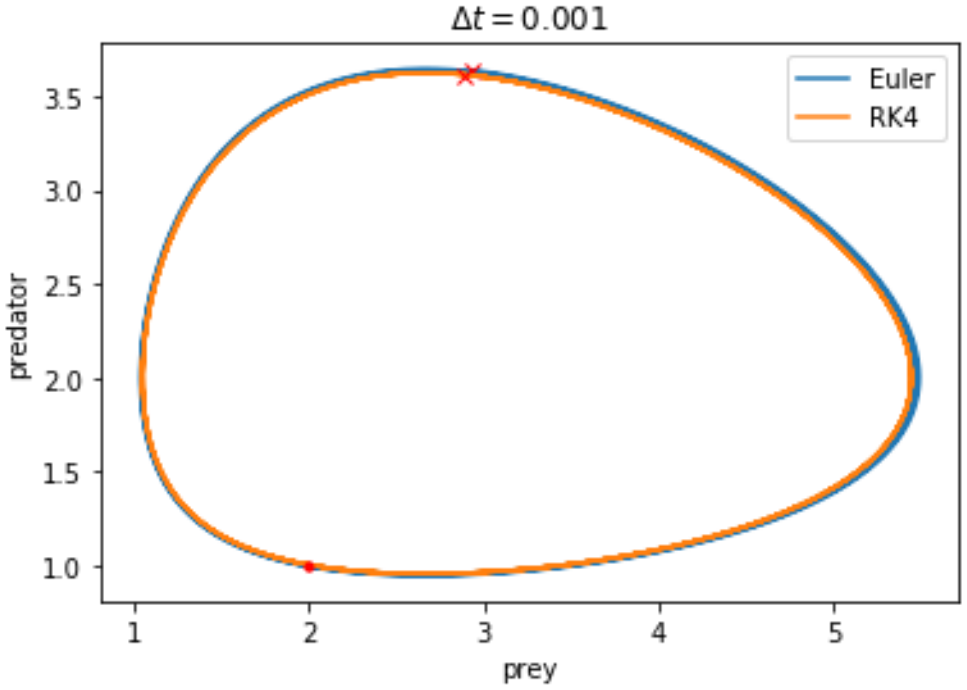
PREDATOR-PREY CODE

For $a = 1.2$, $b = 0.6$, $c = 0.8$, $d = 0.3$ the Python code using the Explicit Euler method for these two coupled differential equations is

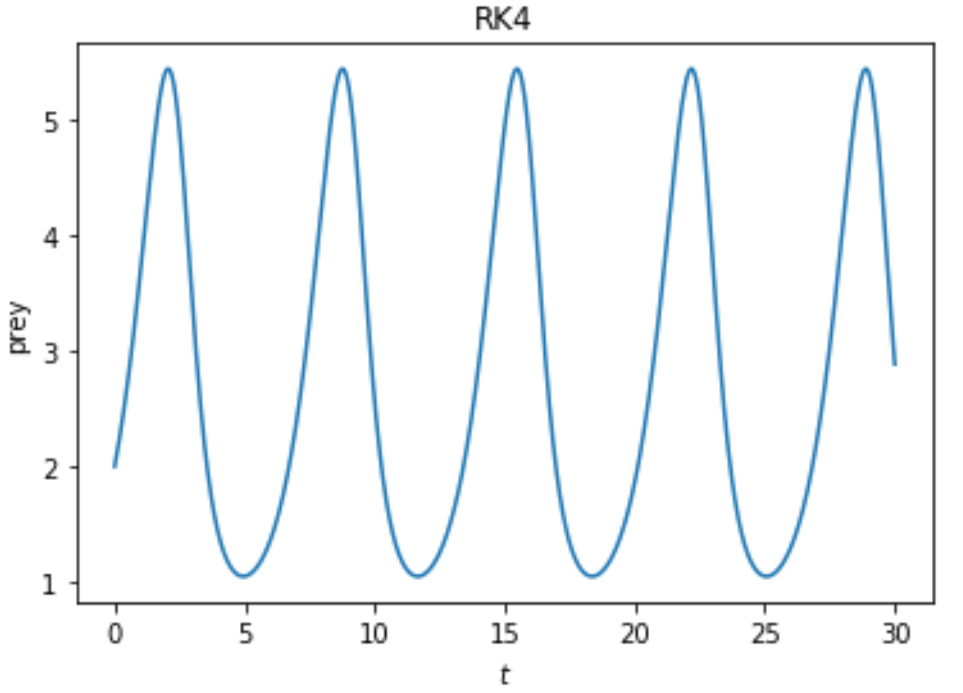
```
import numpy as np
import matplotlib.pyplot as plt

neq=2
tmax=30.0
dt=0.01

# Initial conditions
t0=0.0
x0=2.0
y0=1.0


def gx(t, x, y):
    a=1.2
    b=0.6
    return a*x-b*x*y

def gy(t, x, y):
    c=0.8
    d=0.3
    return -c*y+d*x*y

def g(t, X):
    return np.array([gx(t, X[0], X[1]), gy(t, X[0], X[1])])

nint=int(round((tmax-t0)/dt))
X=np.zeros((neq,nint+1))
t=np.zeros(nint+1)
```

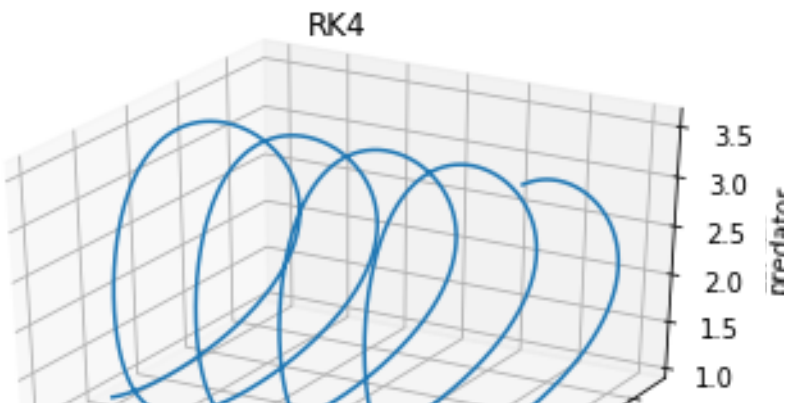
See blackboard for entire '.py' file

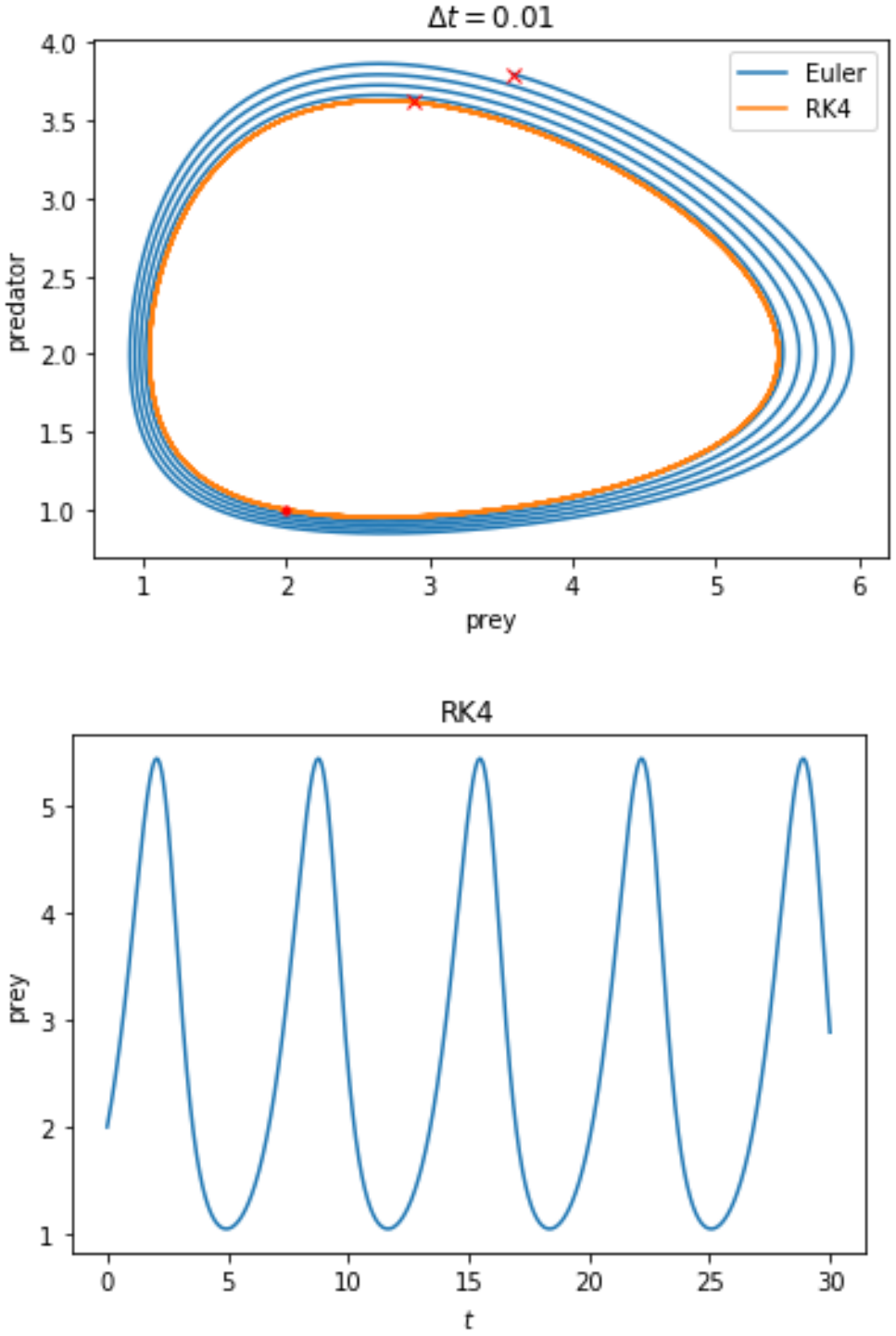
RESULT


$\Delta t = 0.001$
Euler
RK4
predator
prey
1.0
1.5
2.0
2.5
3.0
3.5
1
2
3
4
5

RK4

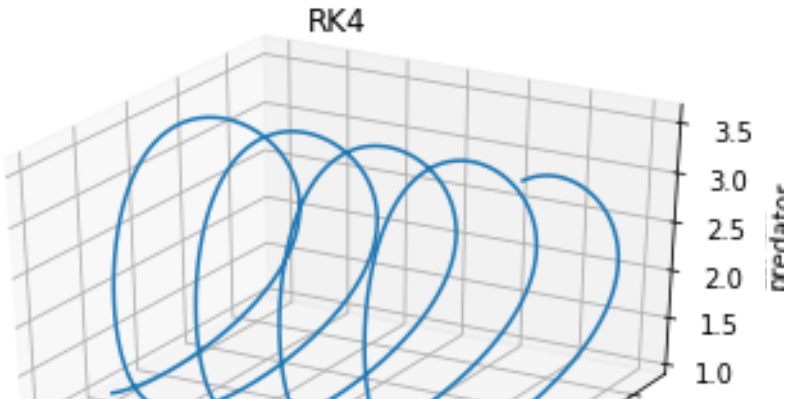

prey
t
1
2
3
4
5
0
5
10
15
20
25
30


RK4
3.5
3.0
2.5
2.0
1.5
1.0

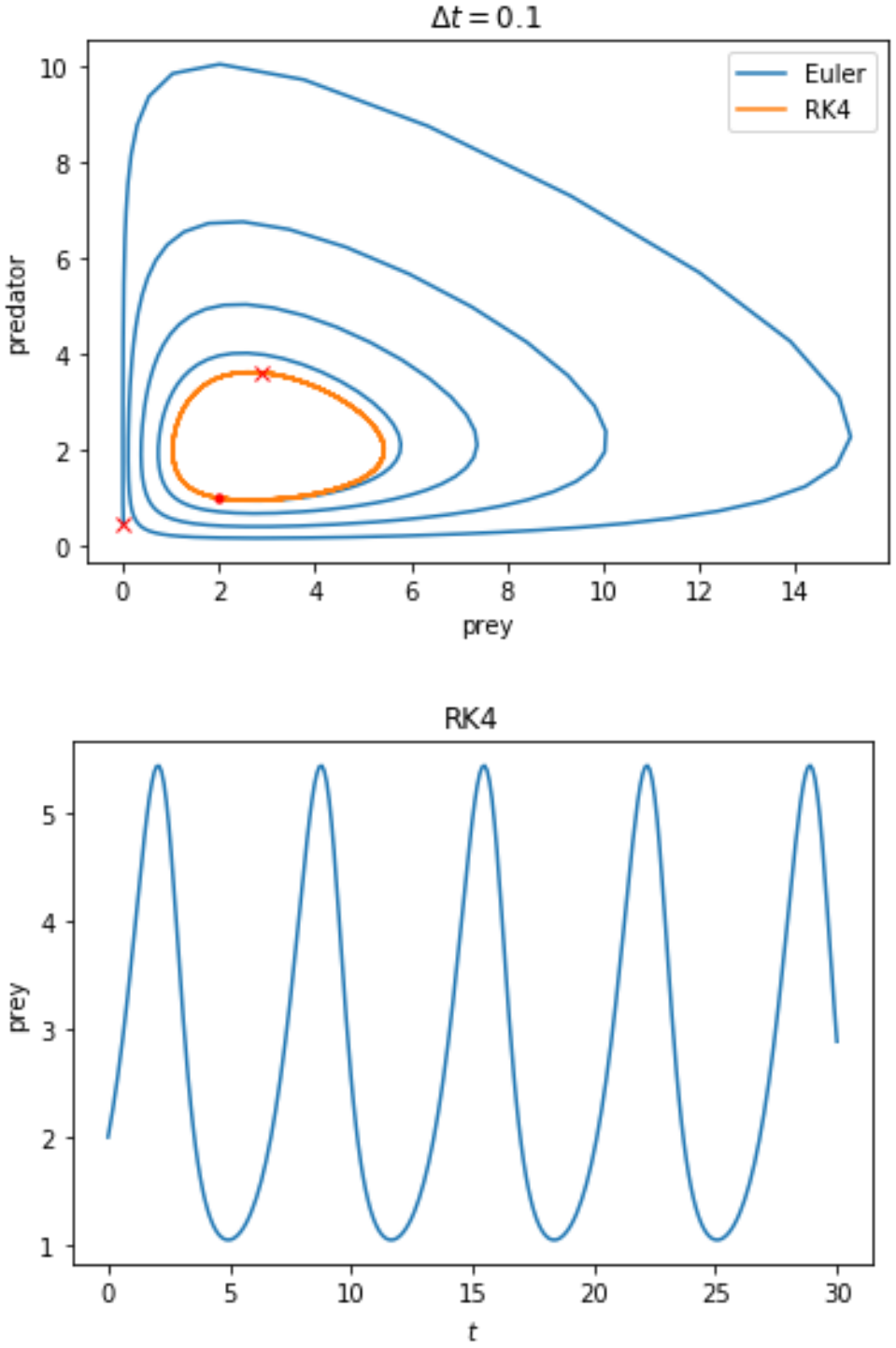

$\Delta t = 0.01$
Euler
RK4
predator
prey
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1
2
3
4
5
6

RK4

prey
t
1
2
3
4
5
0
5
10
15
20
25
30

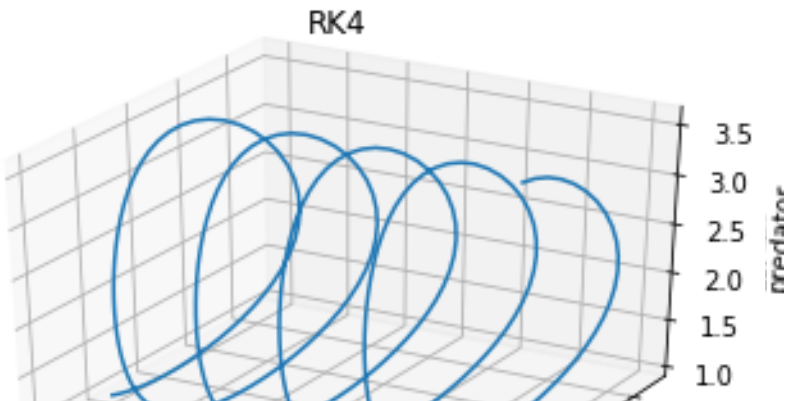

RK4
3.5
3.0
2.5
2.0
1.5
1.0

The stable orbits become unstable for too large timesteps (particularly so for the explicit Euler method).


$\Delta t = 0.1$
Euler
RK4
predator
prey
0
2
4
6
8
10
12
14

RK4

prey
t
1
2
3
4
5
0
5
10
15
20
25
30


RK4
3.5
3.0
2.5
2.0
1.5
1.0

EXERCISES SESSION 4

1 IMPLICIT TO EXPLICIT EQUATION

Rewrite the following equations, so that y is isolated to the left handed side of the equation (i.e., rewrite the equations in an explicit form):

1.1

$$x + y = 4$$

1.2

$$ax + by = c$$

1.3

$$ax + by = cy + dx + e$$

1.4

$$ax + by^2 = cy + dx + e$$

2 IMPLICIT TO EXPLICIT FOR NEXT ITERATION

Do the same, but now isolate y_{i+1} for

2.1

$$y_{i+1} + y_i = 4$$

2.2

$$\frac{y_{i+1} - y_i}{\Delta t} = -a y_{i+1}$$

3 IMPLICIT TO EXPLICIT FOR ODE

Do the same, but now isolate z_{i+1}, by using the finite difference method with implicit Euler for the ordinary differential equation

$$\frac{dz}{dt} = 4c_1 z - 8c_2 t^2$$

where $z_{i+1} = z(t + \Delta t)$.

4 REPRODUCE SOLVING A SYTEM OF ODES

Implement the aforementioned predator-prey equations and see if you can reproduce the results.

5 CONCEPTS

Explain in your own words all the concepts introduced in this lecture, such as:

- stability ODE
- stability method solving ODE
- convergence numerical method
- consistency numerical method
- order of convergence

6 MAIN EXERCISES SESSION 4

See Blackboard, section Assessments