

Numerical methods for differential equations

1. The case of Ordinary Differential Equations (ODE)

Sebastien CHARNOZ & Andrea CIARDI

Paris Physics Master

1. Classification of differential equations

Dynamical systems

The evolution of dynamical systems are governed by *Differential equations*

- Fall of a body:

$$a_z = \frac{d^2 z}{dt^2} = -g$$

- Planetary motion:

$$\vec{a}_i = \sum_{j \neq i} \frac{Gm_j}{r_{ij}^3} (\vec{r}_i - \vec{r}_j)$$

- Heat transfer :

$$\frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} = f(x, t)$$

- Wave Equation etc:

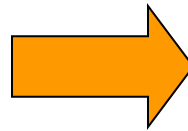
$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f(x, t)$$

- Etc ...

Depending on the system, there are always

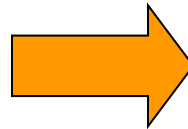
One or more **quantities** to be determined

$$a_z = \frac{d^2 z}{dt^2} = -g$$



X, V, V

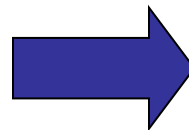
$$\frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} = f(x, t)$$



temperature T

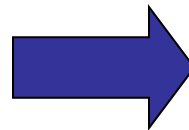
That depend on on one or more **parameters**

$$a_z = \frac{d^2 z}{dt^2} = -g$$



The time t

$$\frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} = f(x, t)$$



Time t, x space

There are always has *boundary conditions (or initial conditions)*

Falling bodies, planetary motion:
initial positions and velocities

Heat Transfer:
boundary condition initial temperature (spatial distribution)

wave equation:

boundary condition (a rope, for example)
Initial state of the rope

« Solving the problem » consists in

CALCULATING THE EVOLUTION OF QUANTITIES AS A FUNCTION OF PARAMETERS

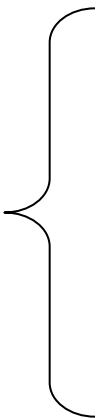
For example :

For planets: $X(t)$ and $V(t)$

For heat: $T(x, t)$

In other words: *solve the differential equation ("integrate")*

A problem is well-posed
if we have

- 
- **A list of quantities to that evolve according a list of parameters**
 - **A differential equations linking all quantities to all parameters. As many equa. Diff as quantities.**
 - **a set of initial, or boundary conditions**

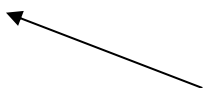
EXAMPLES

Wave propagation

Parameters: X (position) and T (time)

Quantity: U (x, t): Amplitude at position X and at time t

equation:
$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f(x, t)$$



forcing

Initial condition problem Depends of course,
profile u (x) at t = 0

http://micromachine.stanford.edu/~hopcroft/Research/resonator_images/sin_mov1.gif

Planetary motion

Quantities: position and velocity: (X, Y, Z, Vx, Vy, Vz)

Parameter: Time

Differential equations

$$\frac{dx}{dt} = V_x$$

$$\frac{dV_x}{dt} = \sum_{j \neq i} \frac{Gm_j}{r_{ij}^3} (\vec{r}_i - \vec{r}_j)$$

Same for Y and Z

6 in total

Initial conditions: Initial Position and speed

<https://thumbs.gfycat.com/EnchantingPositiveGermanshepherd-mobile.mp4>

Propagation of heat

Quantity: temperature T

Parameter: X (space) and t (time)

equation:
$$\frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} = f(x, t)$$
 ← forcage

Initial condition: Profile T (x) at t = 0

https://upload.wikimedia.org/wikipedia/commons/f/f7/Heat_Transfer.gif

RESOLUTION: THE METHODS

All solution methods are based on the same idea coming from the limits imposed by the computer :

Discretization of the problem

The paramaters are discretized :

example:

time is written $t = t_n = N * dt$ where dt is the time step

The space (1D) will be written: $x_n = n * dx$ or dx will be no gap

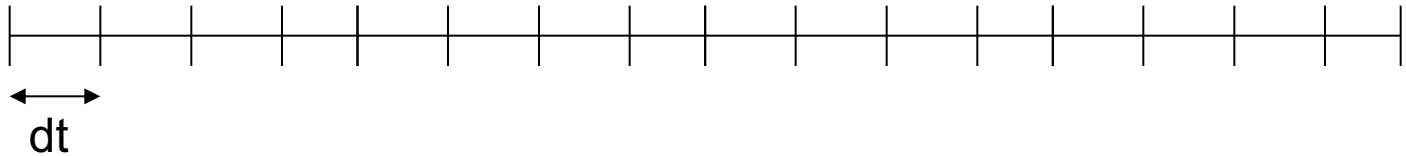
So we solve the problem on a **grid** (for the **PARAMETERS**, not the quantities)

A **time grid**, a **space grid** etc ...

Over the grid is fine (dt or dx) the higher the resolution is close to the exact solution

1D (time, space etc ... 1D) $t_{\text{not}}Dt = nx$

0 1 2 3 4 5 6



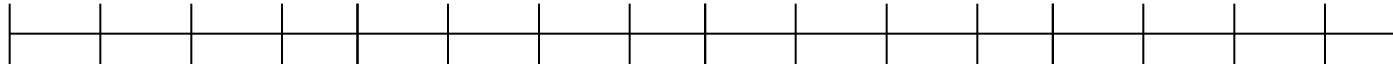
If 1D (1 single parameter, eg time)

The resolution is to calculate a Serie:

$$U_{n+1} = F(U_n, t_n)$$

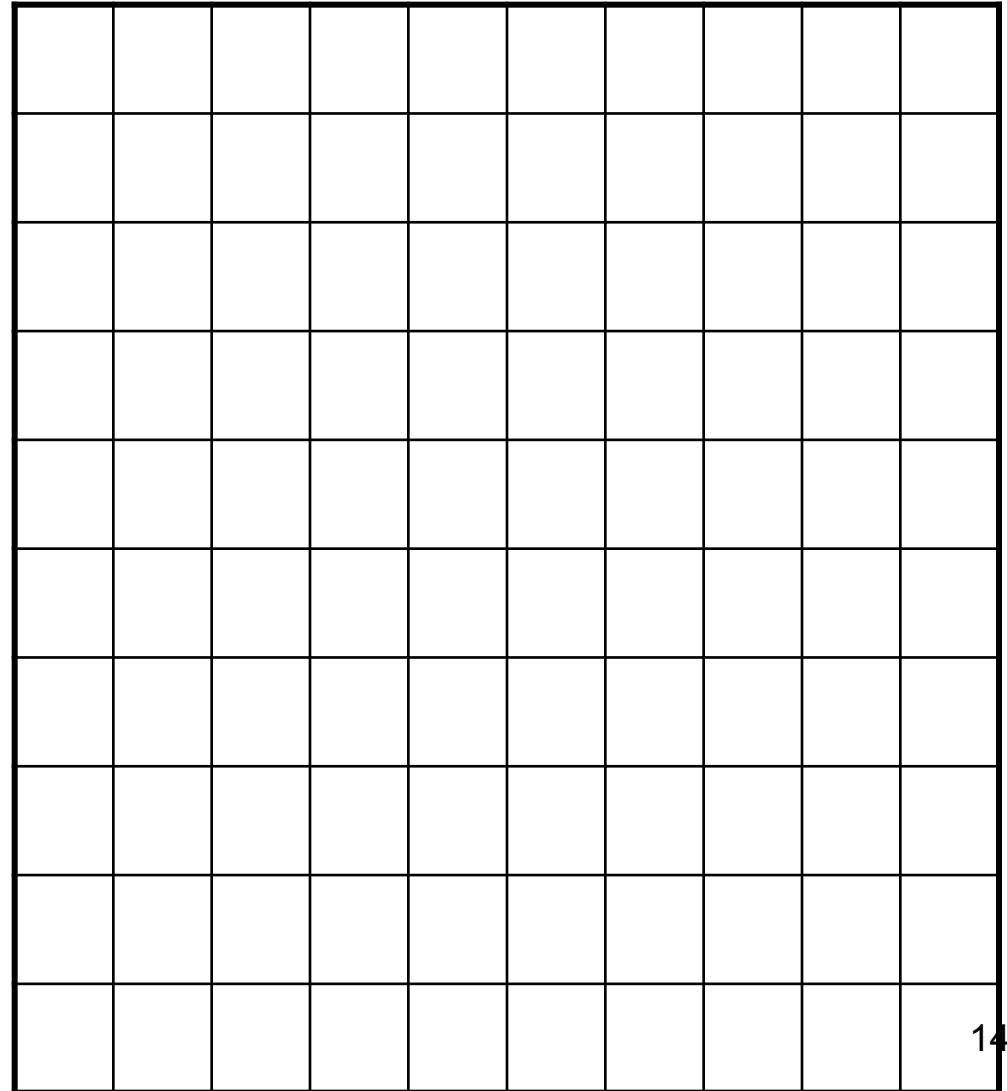
0 1 2 3 4 5 6

Time Grid



dt

Space Grid



In 2D or more

ex:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f(x, t)$$

In fact we made a grid
3D (2D space + time 1D)

$U_{i,j,k} = F$ (neighboring cells)

For now

we are only interested by Ordinary Differential Equations:

ODE

Differential equations that depend only on ONE parameter !

(only one parameter at the denominator of the derivative)

These are ODEs:

$$a_z = \frac{d^2 z}{dt^2} = -g$$

$$\frac{dP}{dz} = -\rho g$$

These are NOT ODEs (they are PDE in fact):

$$\frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} = f(x, t)$$

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f(x, t)$$

For ODE we discretize time like this

$$\mathbf{U}_{n+1} = \mathbf{F}(\mathbf{U}_n, t_n)$$

So the differential equation is solved step by step
from a starting point (= boundary condition) where
System status is known at $t = 0$

The function F is called «solver » or « solver ».
It is an approximation of the real derivative, f , according
to equation :

$$\frac{dU}{dt} = f(U, t)$$

The whole problem is to find a function F :

- * precise
- fast
- robust.

The accuracy of the solution depends on the size of the time
step and the number of calculation steps.

2. Example of numerical resolution

1 Movement of a spring: A simple ODE

Quantities X and V_x

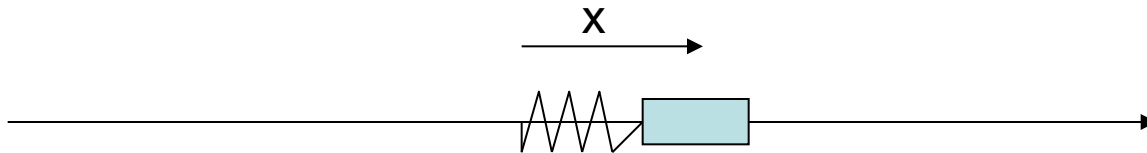
Parameter: t

Equations: $f = ma = -kx \Rightarrow$

$$\frac{dV_x}{dt} = \frac{-kx}{m}$$

k : coefficient. Spring stiffness

m : mass



Do we need something else ?

YES! because we lack the X evolution equation:

$$\begin{aligned}\frac{dx}{dt} &= V \\ \frac{dV}{dt} &= \frac{-kx}{m}\end{aligned}$$

A n^{th} order ODE ($n=2$ for newtonian mechanics) can be always be transformed into
A system of n first order equations !!

Then:

Quantities X and Vx

Initial conditions : X (t = 0) = 10 m

V (t = 0) = 0. m / s

Parameter t

Equations :

k: coefficient. stiffness

m: mass

$$\frac{dx}{dt} = V$$

$$\frac{dVx}{dt} = \frac{-kx}{m}$$

ANALYTICAL SOLUTION:

* demonstrate

$$x(t) = A \cos(\omega t + \varphi)$$

$$v(t) = -A \omega \sin(\omega t + \varphi)$$

$$\omega = \text{sqrt}(k / m), A = X0$$

**We now solve the SAMED problem
But but numerically integrating the equation.**

**We need an algorithm to calculate the evolution
the solution in small time increments dt .**

We will discover the "problems" of numerical integration

Take a grid for the parameter t with $dt = 0.01$ seconds

$$X_{n+1} = F(X_n, V_n, T_n)$$

$$V_{n+1} = F(V_n, V_n, T_n), \text{ where } t_n = N * dt$$

F is the « solver ». We will apply here the method of Euler (we'll see)

ALGORITHM

1. Initialize X_0 and V_0

2. Initialize dt

→ 3. Calculate $X_{n+1} = F(X_n, V_n, T_n)$
and $V_{n+1} = F(V_n, t_n)$

4. Increment time $T = T + dt$

5. Go 3.

2. Construction of a solver: basic methods

An ordinary differential equation can always be written as a set of differential equations of the first order

The diagram illustrates the components of an ordinary differential equation (ODE) and their roles in a solver. The central equation is $\frac{d(x, y, z, u, w....)}{dt} = f(t, x, y, z, u, w....)$. Arrows indicate the following mappings:

- The word "quantities" has two arrows: one pointing down to the dependent variables $x, y, z, u, w....$ in the numerator, and another pointing down and to the right to the state variables $x, y, z, u, w....$ in the function f .
- The label "Parameter (single)" has two arrows: one pointing up and to the right to the time variable t in the function f , and another pointing up and to the left to the differential dt in the denominator.

$$\frac{d(x, y, z, u, w....)}{dt} = f(t, x, y, z, u, w....)$$

Vector Writing

$$\frac{d}{dt} \begin{pmatrix} x \\ y \\ z \\ u \\ \dots \end{pmatrix} = \begin{pmatrix} f_x(t, x, y, z, u, \dots) \\ f_y(t, x, y, z, u, \dots) \\ f_z(t, x, y, z, u, \dots) \\ f_u(t, x, y, z, u, \dots) \\ \dots \end{pmatrix}$$

Example: the spring (calculate *)

$$\frac{d}{dt} \begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} v \\ -kx / m \end{pmatrix}$$

Note: Here the derivative does not explicitly depend on time t , because the force does not depend explicitly time

$$F_x = V \text{ and } F_v = -Kx / m$$

Real equation:

$$\frac{d}{dt} \begin{pmatrix} x \\ y \\ z \\ u \\ \dots \end{pmatrix} = \underbrace{\begin{pmatrix} f_x(t, x, y, z, u, \dots) \\ f_y(t, x, y, z, u, \dots) \\ f_z(t, x, y, z, u, \dots) \\ f_u(t, x, y, z, u, \dots) \\ \dots \end{pmatrix}}_{\text{function } f}$$

Numerical approximation

$X_{n+1} = X$ at time t_{n+1}

where $t_{n+1} = (N + 1) \times dt$

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \\ u_{n+1} \\ \dots \end{pmatrix} = \underbrace{\begin{pmatrix} F_x(t_n, x_n, y_n, z_n, u_n, \dots) \\ F_y(t_n, x_n, y_n, z_n, u_n, \dots) \\ F_z(t_n, x_n, y_n, z_n, u_n, \dots) \\ F_u(t_n, x_n, y_n, z_n, u_n, \dots) \\ \dots \end{pmatrix}}_{\text{function } F}$$

f is the derivative, F is the solver

The Euler Method

The basic tool for building **F** is the Taylor expansion:

$$X(t + dt) = X(t) + dt \cdot f(x, t) + \frac{dt^2}{2!} f'(x, t) + \frac{dt^3}{3!} f''(x, t) + \dots$$

In practice we only know **f**. The goal of any solver is to estimate the best possible développement of **X** knowing only **f**...

It's possible !

Using the development of Taylor :

$$X_{n+1} = X(t + dt)$$

$$X(t + dt) \approx X(t) + dt \frac{dX}{dt} + \frac{dt^2}{2} \frac{d^2 X}{dt^2} + \dots$$

Ignored terms

Function, Equa. diff system

Inspired by this development, the function F will be:

Where F is a numerical approximation the derivative !!

$$X_{n+1} = X_n + dt * F(t, X_n)$$

How to build F:

The simplest case:

Euler Method :

$$\frac{dX}{dt} = f(x, t)$$

We set $F(x, t) = f(x, t)$

The function F (x, t) is then: **$X_{n+1} = X_n + F(x, t) dt$**

Example: Spring Case with Euler

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} x_n + dt v \\ v_n + dt \frac{-kx}{m} \end{pmatrix}$$

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} x_n + dt \, v \\ v_n + dt \, \frac{-kx}{m} \end{pmatrix}$$

The Euler scheme is the simplest possible.

It is a 1-order solver (as between t and $t + dt$ the ERROR is $o(dt^1)$)

It's a quick scheme because there is only ONE call to the derivative f

In practice: never used

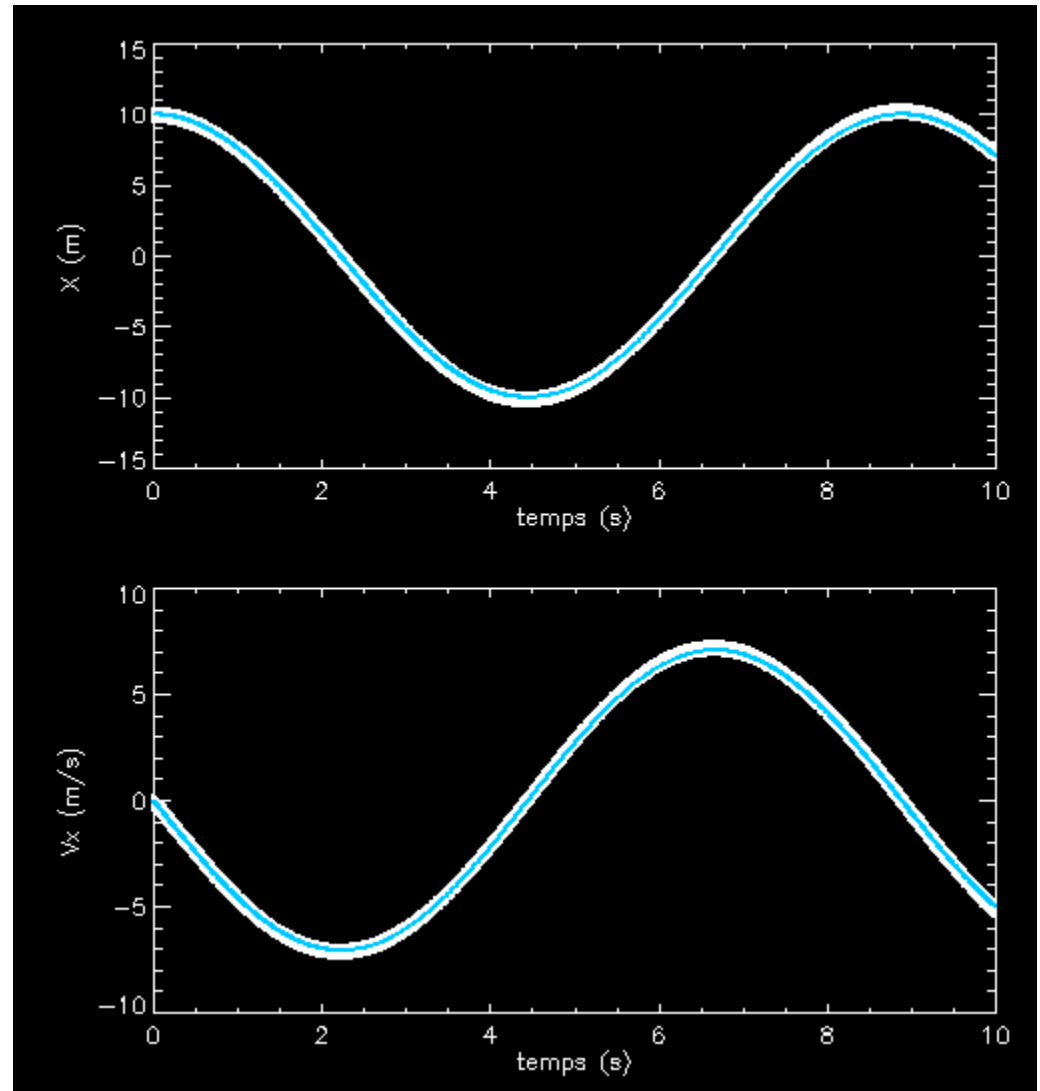
But we can do much better!

numerical solution: spring solved by Euler

$dt = 0.01$ s

N t XV

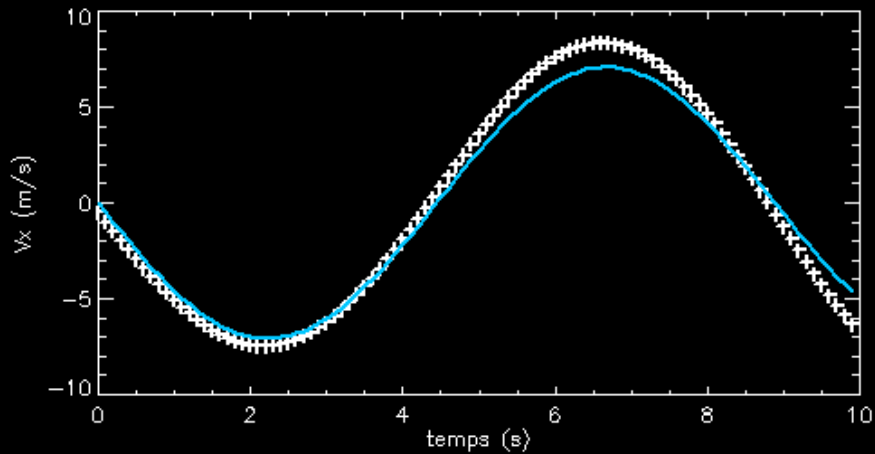
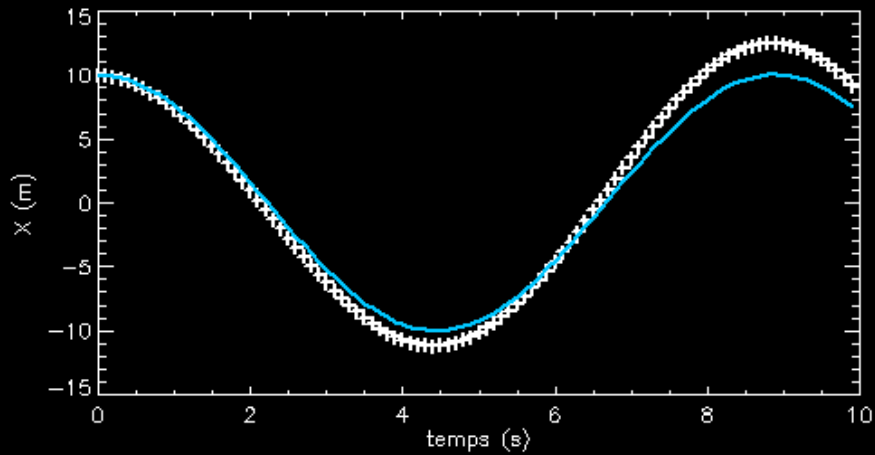
0.000000	0.000000	0.000000	10.0000
1.000000	0.010000	10.0000	-0.0500000
2.000000	0.020000	9.99950	-0.100000
3.000000	0.030000	9.99850	-0.149998
4.000000	0.040000	9.99700	-0.199990
5.000000	0.050000	9.99500	-0.249975
6.000000	0.060000	9.99250	-0.299950
7.000000	0.070000	9.98950	-0.349912
8.000000	0.080000	9.98600	-0.399860
9.000000	0.090000	9.98200	-0.449790
10.0000	0.100000	9.97751	-0.499700
11.0000	0.110000	9.97251	-0.549588
12.0000	0.120000	9.96701	-0.599450
13.0000	0.130000	9.96102	-0.649285
14.0000	0.140000	9.95452	-0.699090
15.0000	0.150000	9.94753	-0.748863
16.0000	0.160000	9.94005	-0.798601
17.0000	0.170000	9.93206	-0.848301
18.0000	0.180000	9.92358	-0.897961
19.0000	0.190000	9.91460	-0.947579
20.0000	0.200000	9.90512	-0.997152
21.0000	0.210000	9.89515	-1.04668
22.0000	0.220000	9.88468	-1.09615
23.0000	0.230000	9.87372	-1.14558
24.0000	0.240000	9.86227	-1.19495
25.0000	0.250000	9.85032	-1.24426
26.0000	0.260000	9.83787	-1.29351
27.0000	0.270000	9.82494	-1.34270
28.0000	0.280000	9.81151	-1.39182
29.0000	0.290000	9.79760	-1.44088
etc ..			



White: Digita : numerical solution
blue: Analytical true solution

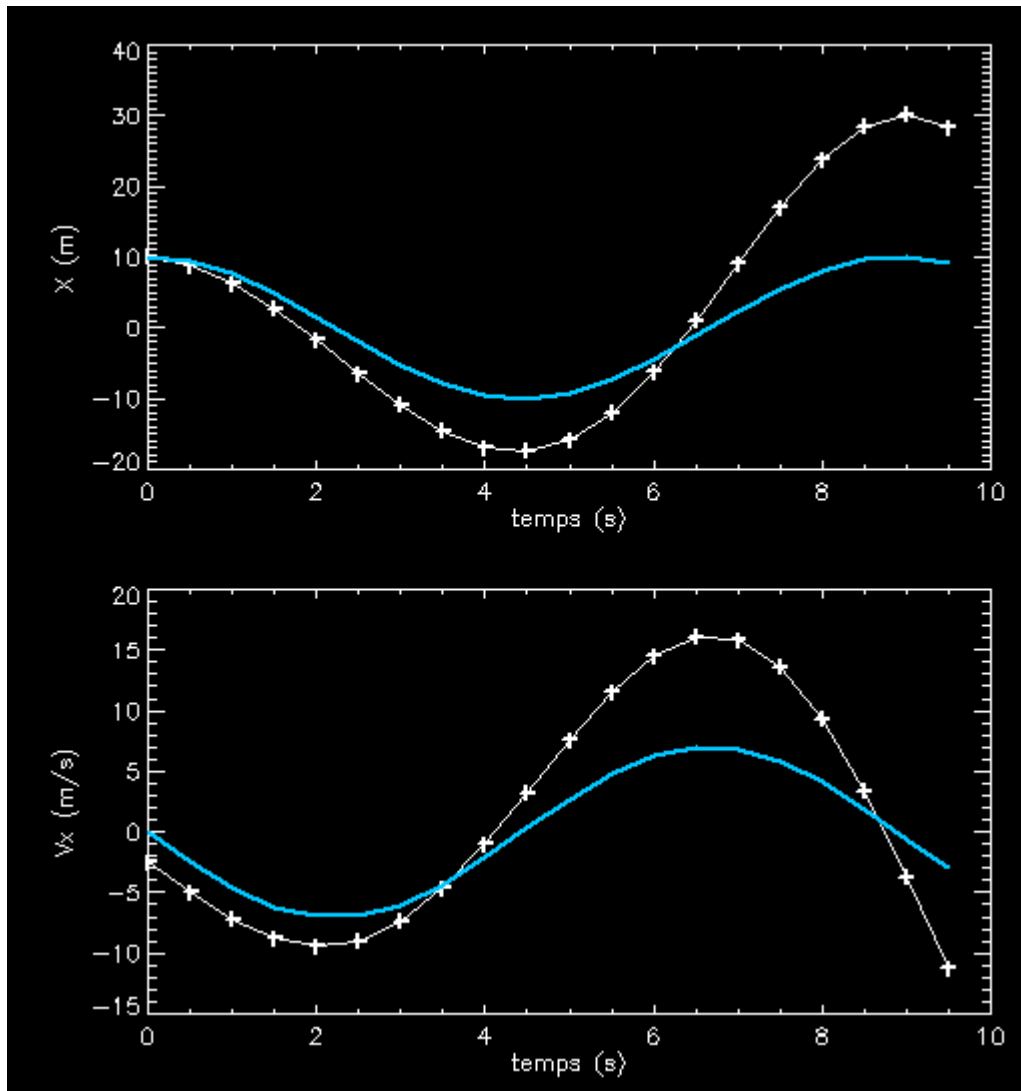
But the numerical solution depends of the time-step

$dt = 0.1$ s



The bigger dt , the larger the error

$dt = 0.5$ s



FOR ANY SOLVER:

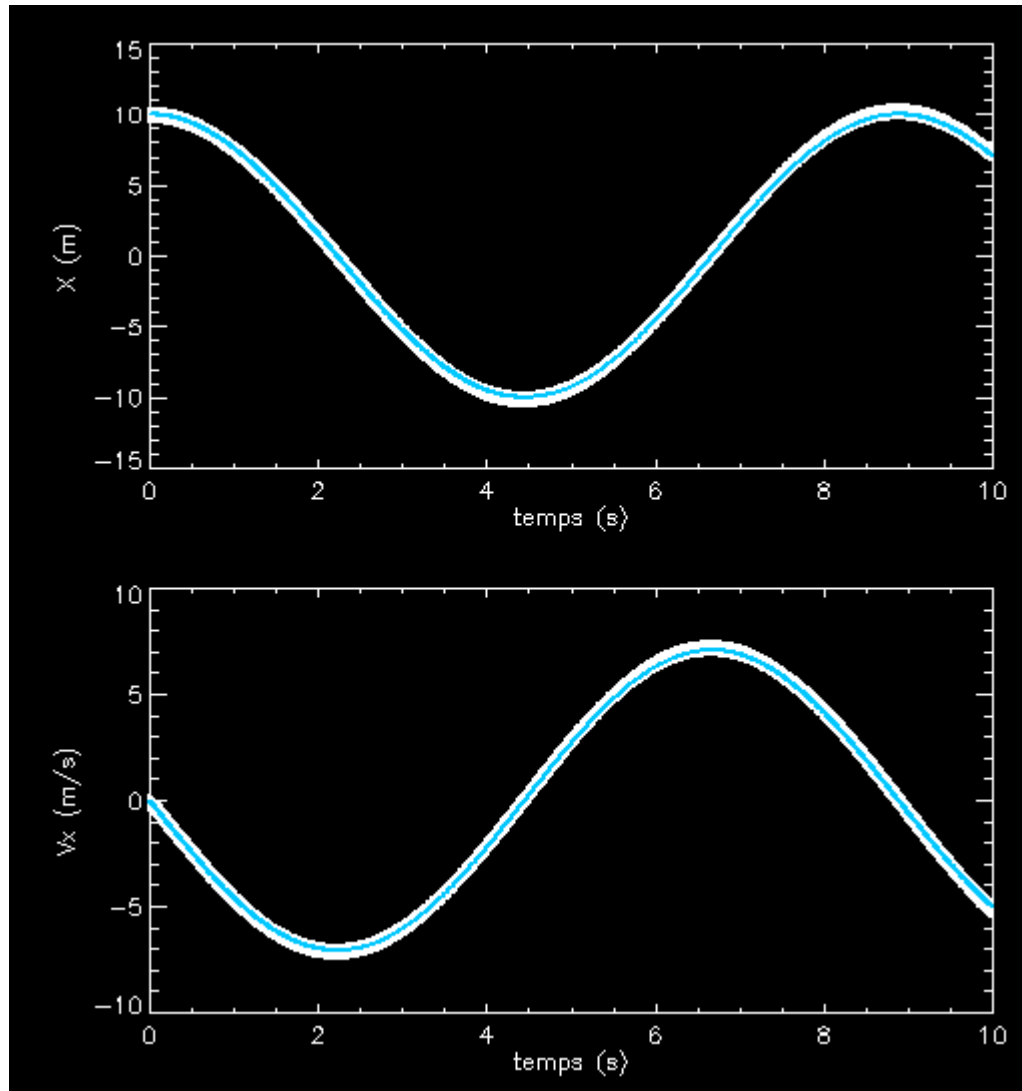
Any numerical solution
is only approximate

The accuracy depends on the
time-step of integration

For larger dt : computing is faster
BUT is less accurate

And vice versa ...

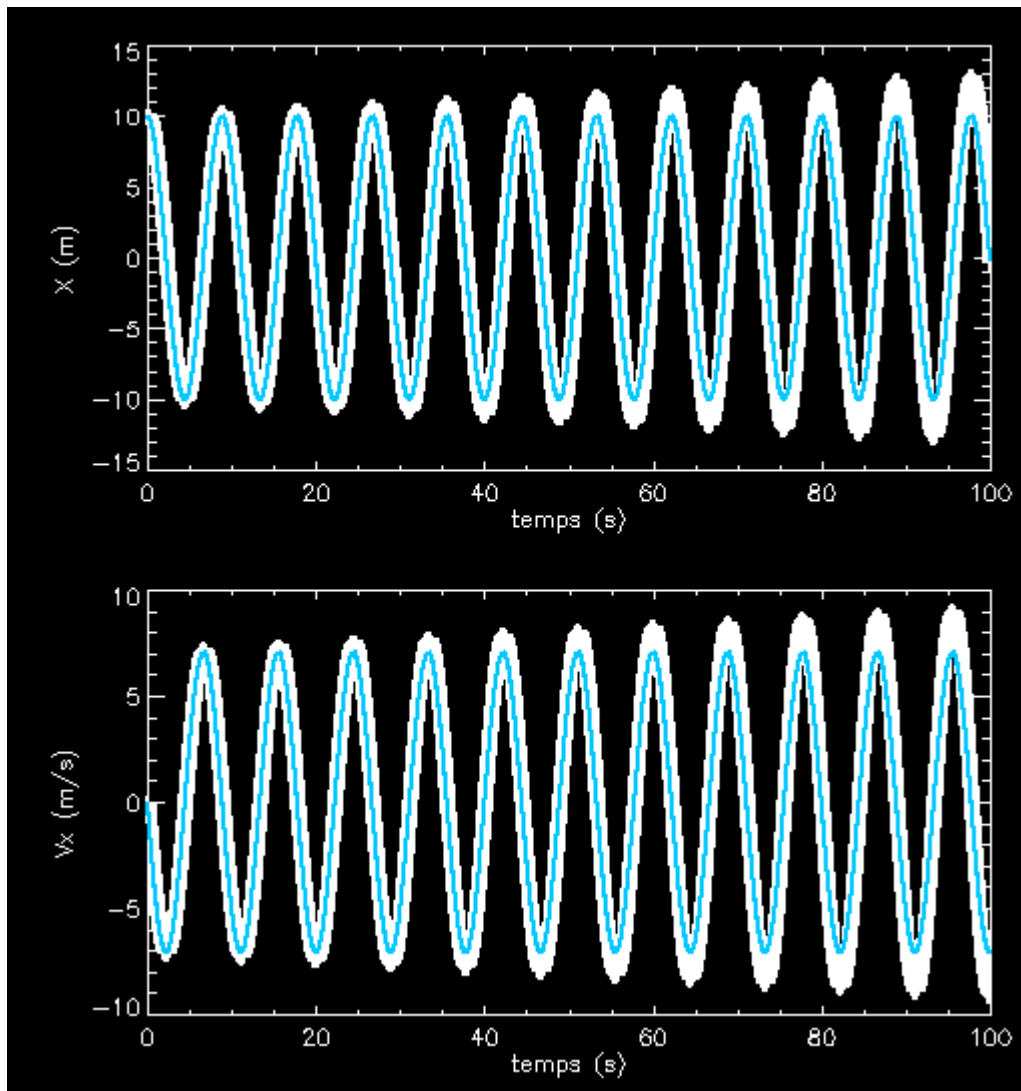
The accuracy decreases as the number of stages of calculations:



$\Delta t = 0.01$ s

1000 calculation steps

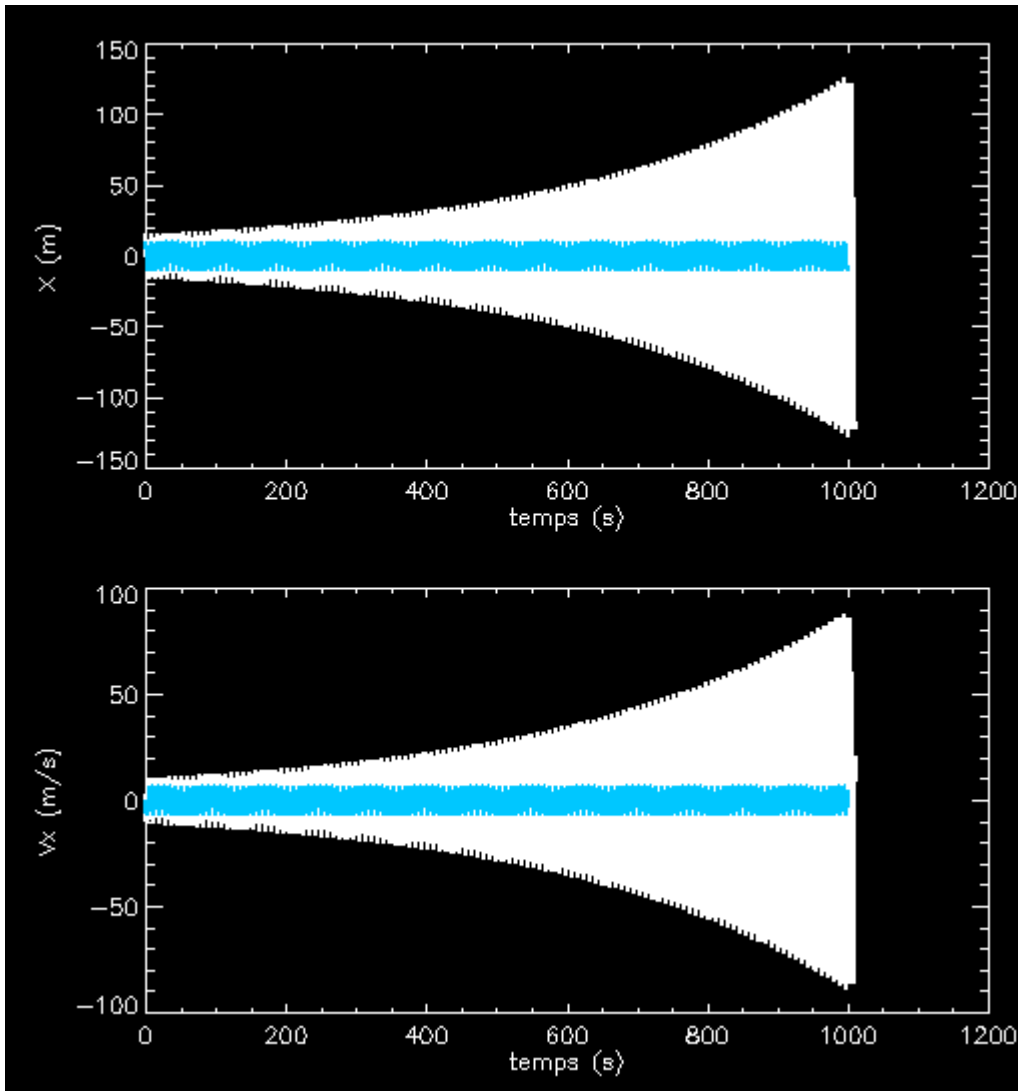
Looks good !



$\Delta t = 0.01$ s

10000 steps

still good !



$\Delta t = 0.01\text{s}$

100000 steps

not good !

All solutions ends
by shifting away from the
solution
when the number of steps
calculation increases

=> Accumulation of errors

How to build a solver?

- Accurate
- Stable
- fast

Build a better approximation, by considering integration, rather than Taylor Expansion

It is known that *correct* X_{n+1} is :

$$X_{n+1} = X_n + \int_t^{t+dt} f(t, X(t), \dots) dt = X_n + dt F(t, X_n)$$

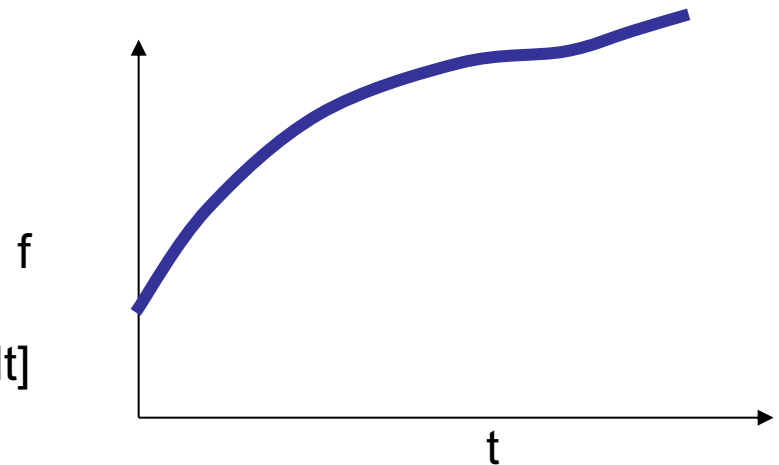
therefore

$$F(t, X_n) = \frac{\int_t^{t+dt} f(t, X(t), \dots) dt}{dt}$$

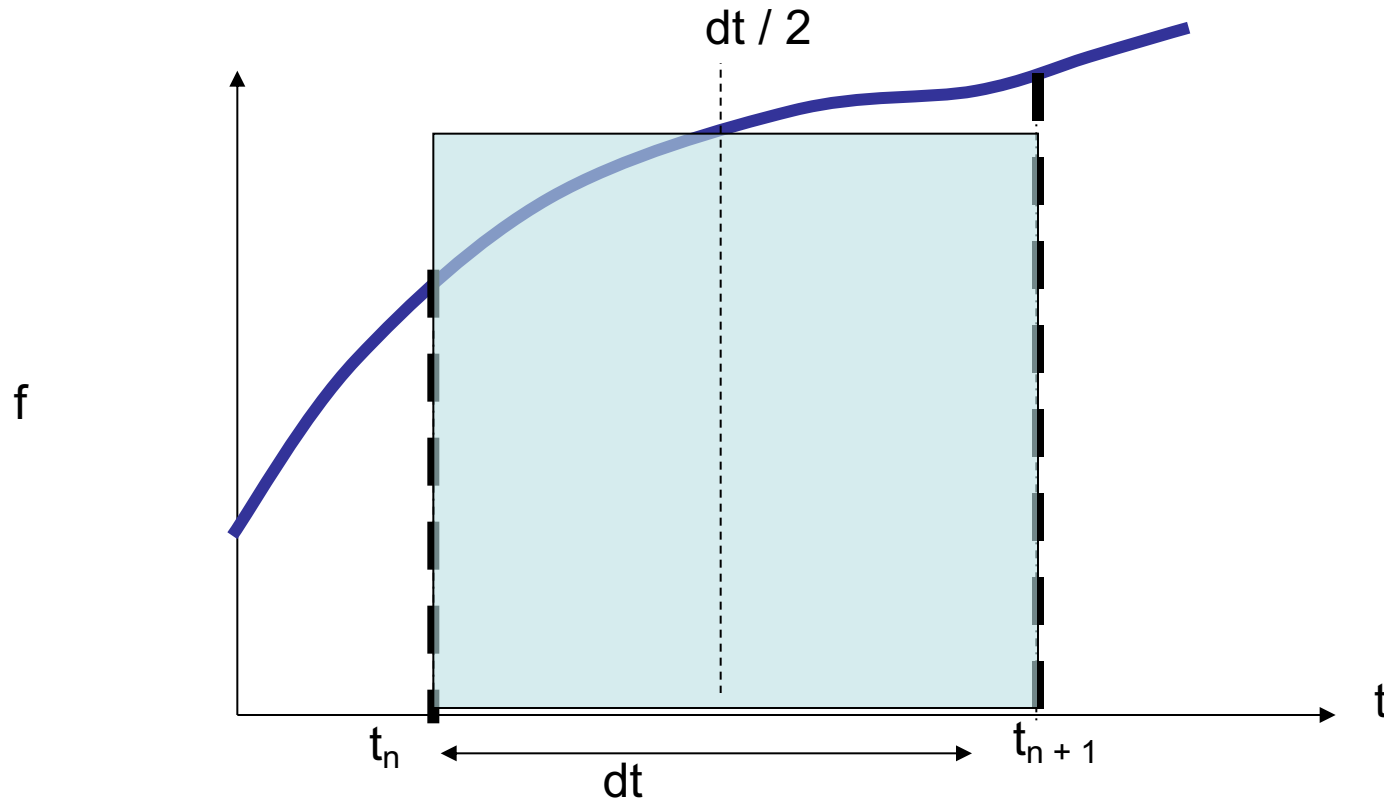
F is the area under the curve divided by dt

Or

F is the mean value of $f(x, t)$ within interval $[t, t+dt]$



Idea: F approximated by the trapezoidal method, midpoint method



$F \sim (\text{blue area}) / dt$

$$\sim \frac{dt * f\left(t + \frac{dt}{2}, X\left(t + \frac{dt}{2}\right)\right)}{dt}$$

Implementation

$$F(X, t) = f\left(t + \frac{dt}{2}, X\left(t + \frac{dt}{2}\right)\right)$$

How do we know $X(t + dt/2)$?

Use Taylor expansion

$$X(t + dt/2) \sim X(t) + dt/2 * f(t, x)$$

then obtained a new integration scheme: **Modified Euler modified**

$$\begin{aligned} X_{n+1} &= X_n + dt D(X_n, t) \\ &= X_n + dt f\left(t + \frac{dt}{2}, X_n + \frac{dt}{2} f(t, X_n)\right) \end{aligned}$$

Algorithm

$$k_1 = X_n + \frac{dt}{2} * f(t, X_n)$$

$$X_{n+1} = X_n + dt * f\left(t + \frac{dt}{2}, k_1\right)$$

Modified Euler is a more accurate solver than simple Euler!!

Show that a modified Euler solver is order 2.

Below we only consider time t dependence

The scheme is:

$$X_{n+1} = X_n + dt \, f\left(t_n + \frac{dt}{2}\right)$$
$$f\left(t_n + \frac{dt}{2}\right) \approx f(t_n) + \frac{dt}{2} f'(t_n) + \frac{dt^2}{8} f''(t_n) + \dots$$

$$X_{n+1} = X_n + dt \, f(t) + \frac{dt^2}{2} f'(t) + \frac{dt^3}{8} f''(t_n) + \dots$$

We therefore get

$$X_{n+1} = X_n + dt f(t) + \frac{dt^2}{2} f'(t) + \frac{dt^3}{8} f''(t_n) + \dots$$

But the real development of X Taylor_{not} is (knowing that $dX / dt = f$)

$$X_{n+1} = X_n + dt f(t) + \frac{dt^2}{2} f'(t) + \frac{dt^3}{6} f''(t_n) + \dots$$

The method of modified Euler is accurate up to order 2 (it fails at the 3rd order)

Difference between 2 steps of "Euler" and 1 step of "Euler Modified"

integrate: $\frac{du}{dt} = f(u)$

Euler

2 steps of length $dt / 2$

$$U_1 = U_0 + \frac{dt}{2} f(U_0)$$

from 0 to $dt / 2$

$$U_2 = U_1 + \frac{dt}{2} f(U_1)$$

$dt / 2$ to dt

\Leftrightarrow

$$U_2 = U_0 + \frac{dt}{2} f(U_0) + \frac{dt}{2} f\left(U_0 + \frac{dt}{2} f(U_0)\right)$$

Step 2 is time dt

Euler modified

1 step of length dt

$$U_1 = U_0 + dt \cdot f\left(U_0 + \frac{dt}{2} f(U_0)\right)$$

Step 1 is time dt

example: $\frac{du}{dt} = -3 \cdot u$; $U(t = 0) = 1$

Integrate with $dt = 0.1$

Euler

2 steps $dt / 2$

$$U_1 = U_0 + \frac{dt}{2} f(U_0) = 0.85$$

$$U_2 = U_1 + \frac{dt}{2} f(U_1) = 0.7225$$

Euler: $U(t = 0.1) = 0.7225$

Modified Euler

1 step dt

$$U_1 = U_0 + dt \cdot f\left(U_0 + \frac{dt}{2} f(U_0)\right)$$

$$U_1 = 0.740818$$

Euler modified :

$U(t = 0.1) = 0.740818$

In this simple case, we have an analytical solution
To which we can compare our result

$$\begin{cases} \frac{du}{dt} = -3u \\ U(t=0) = 1 \end{cases} \Rightarrow U(t) = e^{-3t}$$

$$U(t = 0.1) = 0.740818 \quad \text{Real result}$$

$$\text{Euler: } U(t = 0.1) = 0.7225$$

$$\begin{aligned} &\text{Euler modified} \\ &U(t = 0.1) = 0.740818 \end{aligned}$$

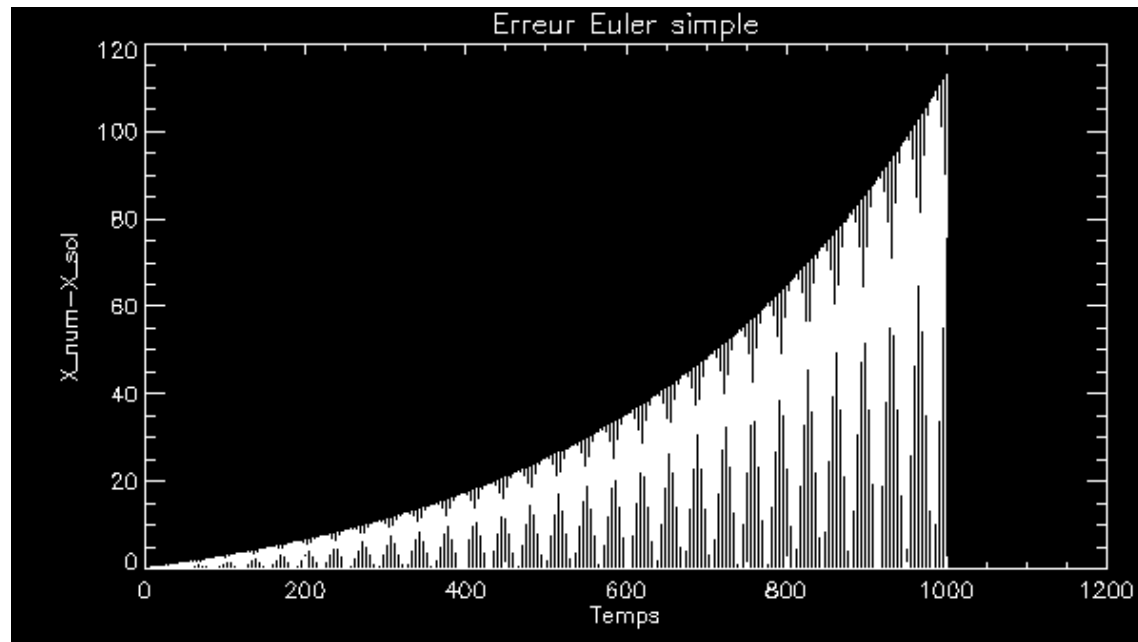
Example with same timestep

Example of the spring

$\text{Abs}(X_{\text{vrai}} - X_{\text{approx}})$

Euler :

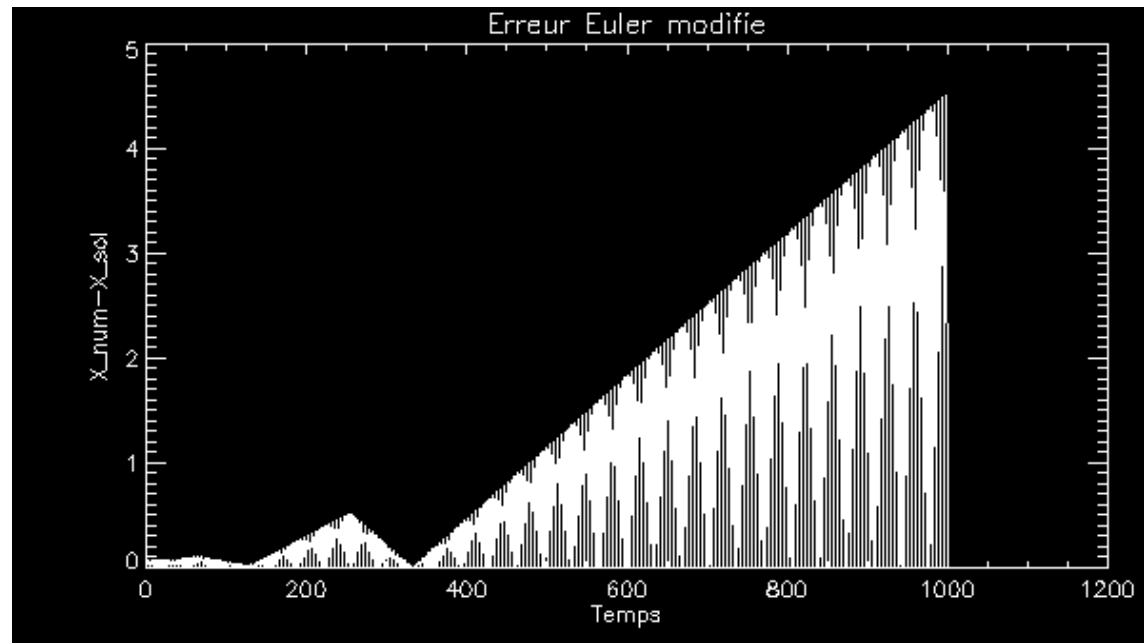
$\Delta t = 0.01$



Euler modified:

Precision Gain:

A factor 20 !!



Euler and Modified Euler are the two simplest solvers

BUT

Many other solvers exist for ODE

Catalogue of ODE most common solvers.

The order of the solver is in parenthesis

explicit Euler (1) $U_{n+1} = U_n + dt \cdot f(t_n, U_n)$

Euler implicit (1) $U_{n+1} = U_n + dt \cdot f(t_{n+1}, U_{n+1})$

Leap Frog (2) $U_{n+1} = U_{n-1} + 2dt \cdot f(t_n, U_n)$

Euler modified (2) $U_{n+1} = U_n + dt \cdot f\left(t_n + \frac{dt}{2}, U_n + \frac{dt}{2} f(t_n, U_n)\right)$

Cranck Nicholson (2) $U_{n+1} = U_n + \frac{dt}{2} \cdot (f(t_n, U_n) + f(t_{n+1}, U_{n+1}))$
implicit

Adam Bashfort (2) $U_{n+1} = U_n + dt \cdot \left(\frac{3}{2} f(t_n, U_n) - \frac{1}{2} f(t_{n-1}, U_{n-1})\right)$

Adam Bashfort (3) $U_{n+1} = U_n + dt \cdot \left(\frac{23}{12} f(t_n, U_n) - \frac{16}{12} f(t_{n-1}, U_{n-1}) + \frac{5}{12} f(t_{n-2}, U_{n-2})\right)$

Adam Moulton (3) $U_{n+1} = U_n + dt \cdot \left(\frac{5}{12} f(t_{n+1}, U_{n+1}) + \frac{8}{12} f(t_n, U_n) - \frac{1}{12} f(t_{n-1}, U_{n-1})\right)$

Runge Kutta (2)

$$\begin{cases} k_1 = dt \cdot f(t_n, U_n) \\ k_2 = dt \cdot f(t_n + dt, U_n + k_1) \\ U_{n+1} = U_n + \frac{1}{2}(k_1 + k_2) \end{cases}$$

Runge Kutta (4)

$$\begin{cases} k_1 = dt \cdot f(t_n, U_n) \\ k_2 = dt \cdot f(t_n + \frac{dt}{2}, U_n + \frac{k_1}{2}) \\ k_3 = dt \cdot f(t_n + \frac{dt}{2}, U_n + \frac{k_2}{2}) \\ k_4 = dt \cdot f(t_n + dt, U_n + k_3) \\ U_{n+1} = U_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases}$$

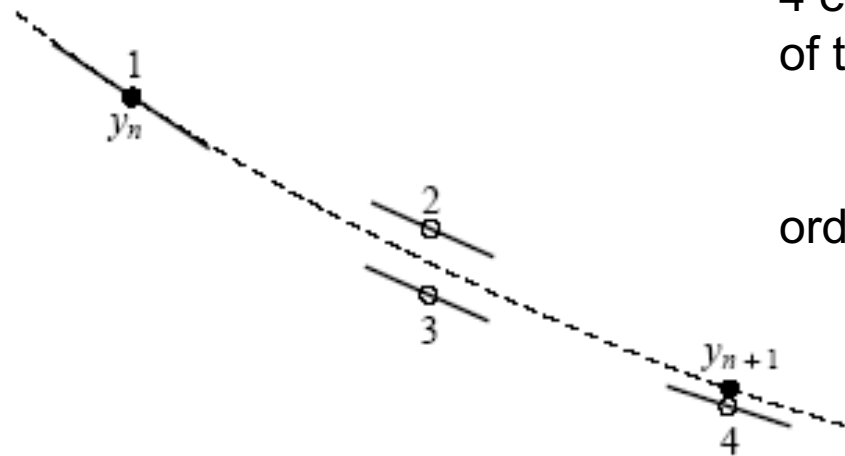
A "popular" explicit solver: the Runge Kutta 4 (RK4)

$$\left\{ \begin{array}{l} k_1 = dt \cdot f(t_n, U_n) \\ k_2 = dt \cdot f(t_n + \frac{dt}{2}, U_n + \frac{k_1}{2}) \\ k_3 = dt \cdot f(t_n + \frac{dt}{2}, U_n + \frac{k_2}{2}) \\ k_4 = dt \cdot f(t_n + dt, U_n + k_3) \\ U_{n+1} = U_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{array} \right.$$

Easy to implement, "Relatively" stable ..

Quite "slow" because 4 calls to the derivative.

Principle



RK4:
4 evaluations in
of the time

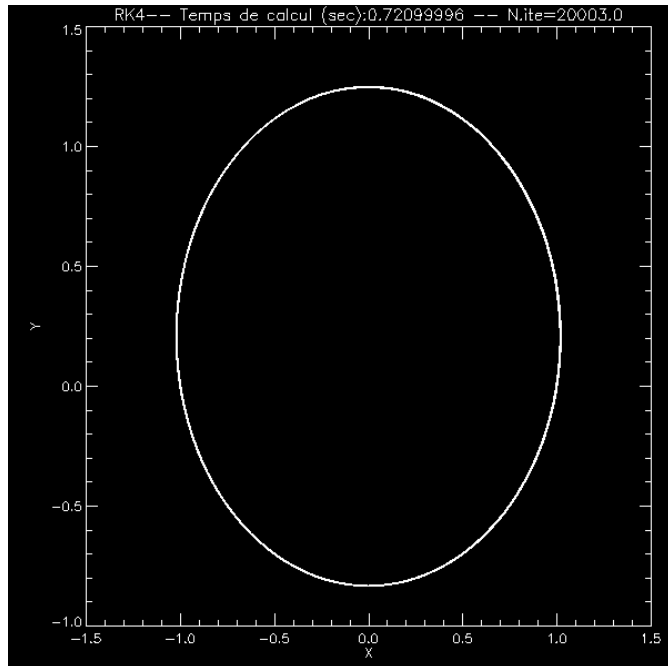
order 4

Figure 16.1.3. Fourth-order Runge-Kutta method. In each step the derivative is evaluated four times: once at the initial point, twice at trial midpoints, and once at a trial endpoint. From these derivatives the final function value (shown as a filled dot) is calculated. (See text for details.)

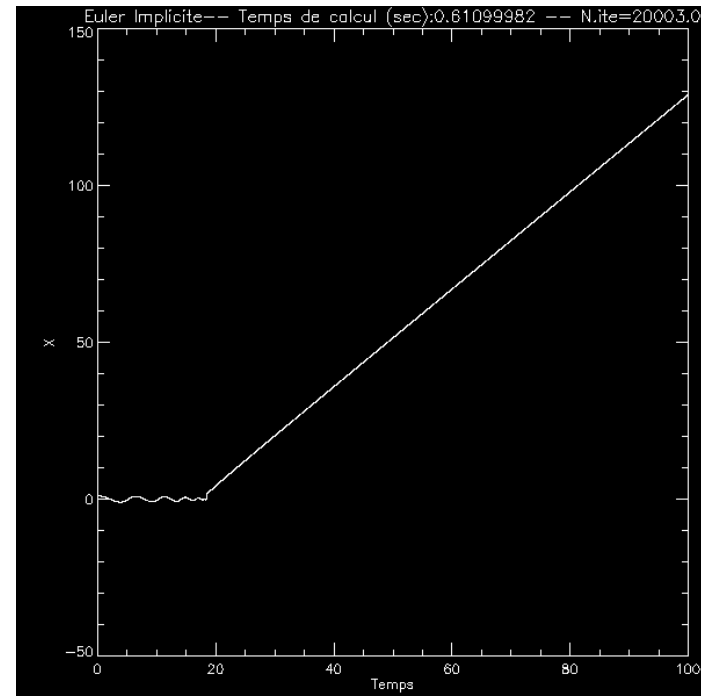
Consider a complicated system:
The motion of a planet around the Sun:

$$\mathbf{A} = -GM \mathbf{u} / r^2$$

$\Delta t = 0.1$, Dynamic Time = 2π



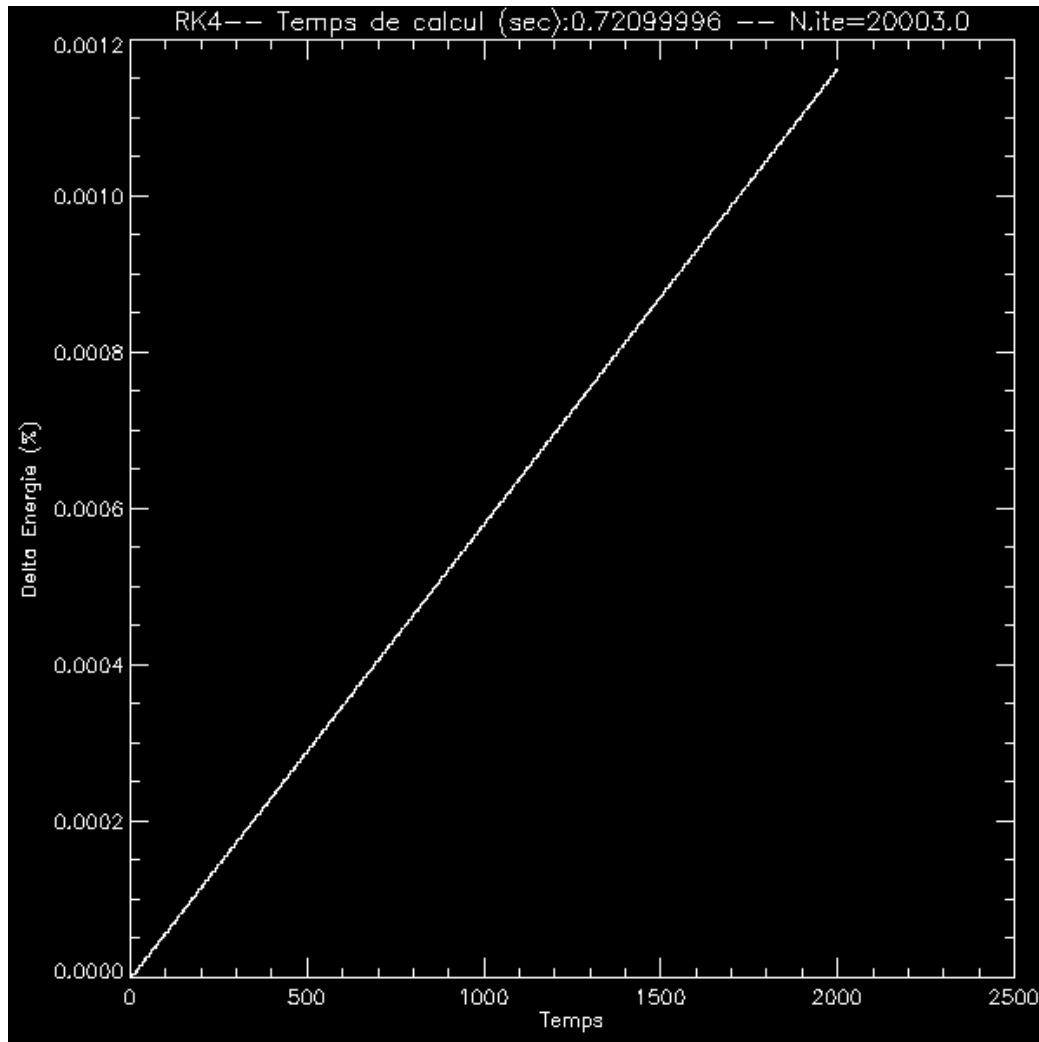
RK4
Everything goes well priori



Implicit Euler

A simple control parameter: Energy

The total energy should be kept: $E = 1/2 mV^2 - Gm / r$



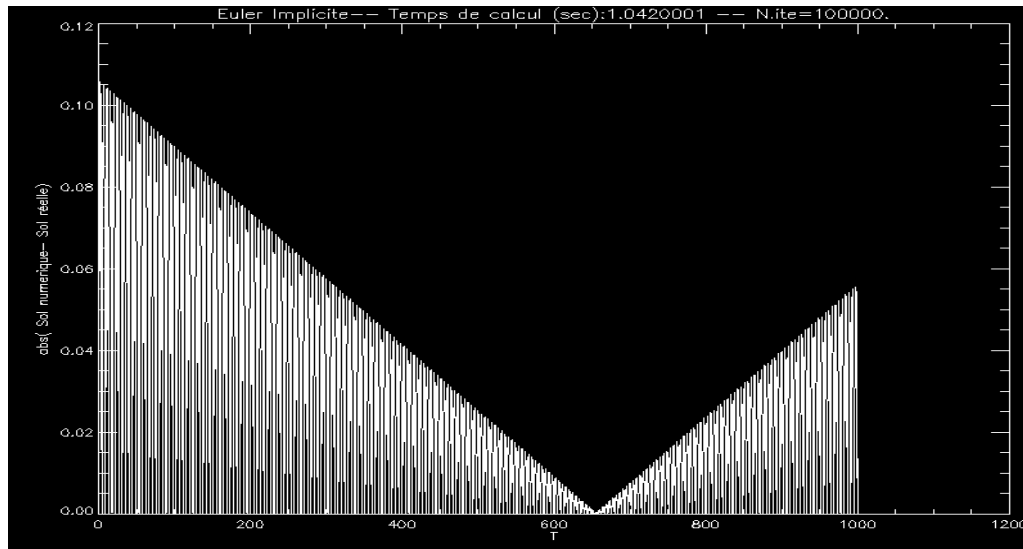
$dt = 0.1$

1 orbit = 2π

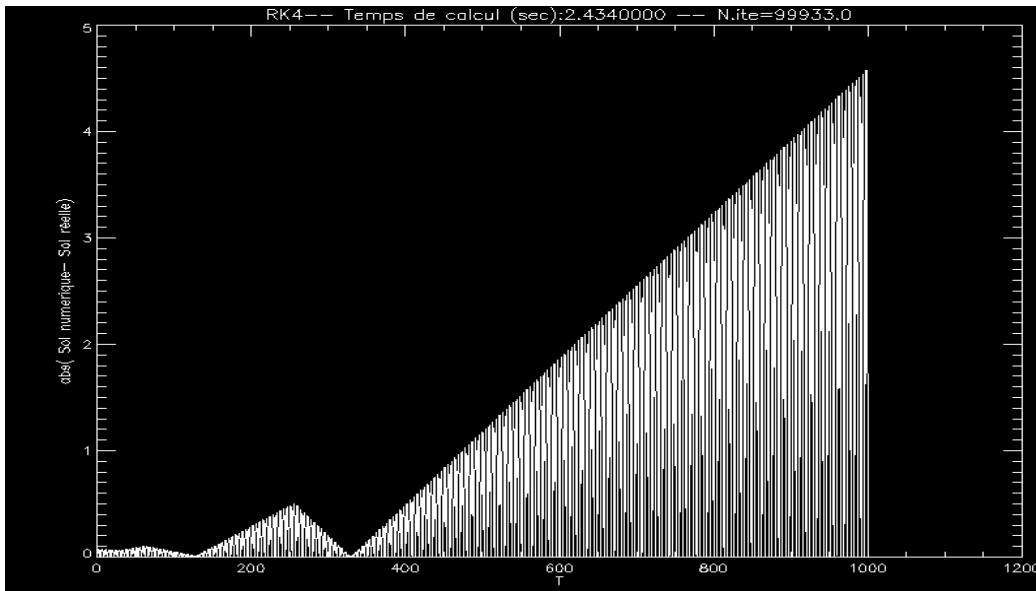
Energy artificially
increased from 0.12%
in 250 orbits
(250 dynamic time)

⇒ We believe the result will
Beyond 250000 orbits
(1000 times) because
 $\Delta E \sim E$ after this time

**Is The Runge Kutta more accurate than Euler implicit?
In principle yes, but not always !!! example: spring**



Implicit Euler



RK4

The RK4 is worse

⇒ In fact the implicit equation
is better suited to for the spring

⇒ Fewer calls to the derivative
⇒ less rounding errors

IMPLICIT SOLVERS

The schemes we have seen are called **explicit solvers** :

$$\text{Euler: } X_{n+1} = X_n + dt f(t_n, X_n)$$

$$\text{Euler modified: } X_{n+1} = X_n + dt f(t + dt / 2, f(X_n + dt / 2))$$

because X_{n+1} **only DEPENDS** on the previous indices (n, n-1, etc ..).
It is calculated directly.

ANOTHER family of solvers are the

« **implicit solvers** » :

Example: Cranck Nicolson

$$X_{n+1} = X_n + \frac{dt}{2} [f(t_n, X_n) + f(t_{n+1}, X_{n+1})]$$

X_{n+1} depends on X_{n+1} and X_n

It is necessary to solve a nonlinear equation to determine X_{n+1} .

Not straightforward ! \Rightarrow implicit scheme.

Implicit solvers are often less accurate

" " " " " " " " " " "are more complicated to use and modify

BUT

They are more stable than the explicit solvers: they diverge less quickly.

In other words: less accurate in the short term, more accurate on then long-term⁵¹

Parctical example EULER implicit

Schemes: $U_{n+1} = U_n + dt \cdot f(t_{n+1}, U_{n+1})$

equation to
resolve:

$$\frac{d}{dt} \begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} v \\ -kx / m \end{pmatrix}$$

Writing the implicit algorithm:

$$\begin{cases} x_{n+1} = x_n + dt v_{n+1} \\ v_{n+1} = v_n + dt \frac{-kx_{n+1}}{m} \end{cases}$$

This is solved by substitution

$$\begin{cases} x_{n+1} = x_n + dt v_{n+1} \\ v_{n+1} = v_n + dt \frac{-kx_{n+1}}{m} \end{cases} \Rightarrow$$

$$\begin{cases} x_{n+1} = x_n + dt \left(v_n + dt \frac{-kx_{n+1}}{m} \right) \\ v_{n+1} = v_n + dt \frac{-k}{m} (x_n + dt v_{n+1}) \end{cases} \Rightarrow$$

$$\begin{cases} x_{n+1} = (x_n + dt v_n) \cdot \frac{1}{\alpha} \\ v_{n+1} = \left(v_n + dt \frac{-kx_n}{m} \right) \cdot \frac{1}{\alpha} \end{cases} \Rightarrow \boxed{} \alpha = 1 + \frac{kdt^2}{m}$$

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \frac{1}{\alpha} \begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_n \\ v_n \end{pmatrix}$$

Stability of a solver

For a given solver, it is important to **quantify its stability**, in other words its conditions of divergence

What is the "stability"?

- MOST solvers go away from the real solution.

**⇒ The question is: what is the "sensitivity" of the computational schemes?
How fast does it diverge ?**

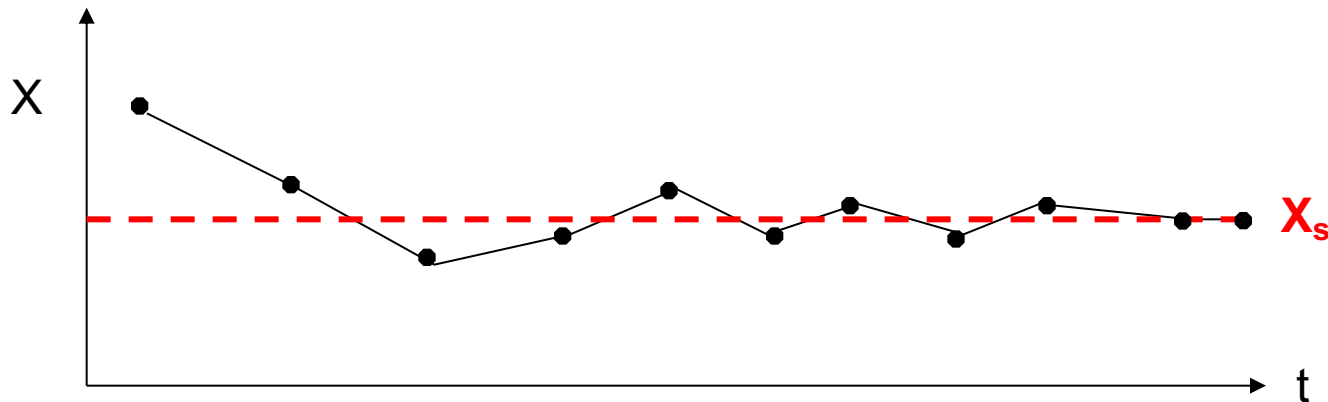
or
How fast are small errors amplified by the scheme ?

Let's consider this scheme : $X_{n+1} = X_n + dt F(t_n, X_n)$

To quantify the **stability**, we \Rightarrow implicit scheme.

assume that X is **adjacent to a stationary point** X_s

We quantify the the rate at which the scheme *goes away from* the solution.



EXAMPLE spring using Euler explicit method (Intrinsically linear method) :

$$\begin{cases} x_{n+1} = x_n + dt \, v_n \\ v_{n+1} = v_n + dt \, \frac{-kx_n}{m} \end{cases} \Rightarrow$$

**We use matrix notation
for systems of ODEs**

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_n \\ v_n \end{pmatrix} \Rightarrow$$

$$A = \begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix}$$

$$\mathbf{X}_{n+1} = A \mathbf{X}_n$$

Let $\mathbf{e}_{n+1} = \mathbf{X}_{n+1} - \mathbf{X}_n$ the error in the solution. We assume $e_n \ll X_n$

Will the numerical scheme will amplify or damp the error ?

$$X_{n+1} = F(X_n) = (1 + dt \cdot f)X_n = AX_n$$

$$e_{n+1} = X_{n+1} - X_n \Rightarrow$$

$$e_{n+1} = AX_n - AX_{n-1} \Rightarrow$$

$$e_{n+1} = A(X_n - X_{n-1}) \Rightarrow$$

$$e_{n+1} = Ae_n$$

$$1 + dt \cdot f$$

A is the amplification matrix

Stability conditions? Looking at the eigenvalues of A

Matrix notation. Example with a system with three variables: X, Y and Z. Let ex , ey and ez errors in X, Y and Z

$$\begin{pmatrix} ex_{n+1} \\ ey_{n+1} \\ ez_{n+1} \end{pmatrix} = A \begin{pmatrix} ex_n \\ ey_n \\ ez_n \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} ex_{n+1} \\ ey_{n+1} \\ ez_{n+1} \end{pmatrix} = M^{-1} D M \begin{pmatrix} ex_n \\ ey_n \\ ez_n \end{pmatrix} \Rightarrow$$

$$M \underbrace{\begin{pmatrix} ex_{n+1} \\ ey_{n+1} \\ ez_{n+1} \end{pmatrix}} = D M \underbrace{\begin{pmatrix} ex_n \\ ey_n \\ ez_n \end{pmatrix}}$$

If A is diagonalizable
 $A = M^{-1} D M$,
 where M is the transition matrix
 and the diagonal matrix D containing
 eigen values

Vectors in the eigen base A

In the eigen base

$$\begin{pmatrix} ex'_{n+1} \\ ey'_{n+1} \\ ez'_{n+1} \end{pmatrix} = \begin{pmatrix} v_1 & & 0 \\ & v_2 & \\ 0 & & v_3 \end{pmatrix} \cdot \begin{pmatrix} ex'_n \\ ey'_n \\ ez'_n \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} ex'_{n+1} \\ ey'_{n+1} \\ ez'_{n+1} \end{pmatrix} = \begin{pmatrix} v_1 \cdot ex'_n \\ v_2 \cdot ey'_n \\ v_3 \cdot ez'_n \end{pmatrix} \Rightarrow \mathbf{e}'_n = \mathbf{V}^n \mathbf{e}'_1$$

So for STABILITY (\Leftrightarrow no amplification of error)

all the eigenvalues of the amplification matrix A must be <1 in absolute value.

Let $V = \text{MAX} [\text{abs}(v_1), \text{abs}(v_2), \text{abs}(v_3)]$. V depends of the time dt

- If $V < 1$ $dt < dt_{\text{max}}$ then the scheme is *conditionally stable*
- if $V_i < 1$ for all dt , then the scheme is *unconditionally stable*
- if $V_i > 1$ for all dt , then the schema is *unconditionally unstable*

Example with the spring + Euler method

$$\begin{cases} x_{n+1} = x_n + dt \, v_n \\ v_{n+1} = v_n + dt \, \frac{-kx_n}{m} \end{cases} \Rightarrow$$

Form: $X_{n+1} = AX_n$

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_n \\ v_n \end{pmatrix} \Rightarrow$$

$$A = \begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix}$$

Amplification matrix A

λ Eigen values : solution of characteristic polynomial
 $\lambda^2 - \text{trace}(A) \times \lambda + \text{determinant} = 0 \Rightarrow$

$$\lambda^2 - 2\lambda + \left(1 + \frac{kdt^2}{m}\right) = 0 \Rightarrow$$

$$\Delta = 4 - 4\left(1 + \frac{kdt^2}{m}\right) = \frac{-4kdt}{m} \Rightarrow$$

$$\lambda = \frac{2 \pm i\sqrt{\frac{4kdt^2}{m}}}{2} = 1 \pm idt\sqrt{\frac{k}{m}}$$

The two eigenvalues are always smaller than (in norm): $\sqrt{1 + \left(dt\sqrt{\frac{k}{m}}\right)^2} > 1$

So Euler scheme, applied to spring equation is *unconditionally unstable* ($dt > 0$)

Example with the spring + implicit Euler method

$$\begin{cases} x_{n+1} = x_n + dt v_{n+1} \\ v_{n+1} = v_n + dt \frac{-kx_{n+1}}{m} \end{cases} \Rightarrow$$

Substitution

$$\begin{cases} x_{n+1} = x_n + dt \left(v_n + dt \frac{-kx_{n+1}}{m} \right) \\ v_{n+1} = v_n + dt \frac{-k}{m} (x_n + dt v_{n+1}) \end{cases} \Rightarrow$$

$$\frac{1}{\alpha} \begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix}$$

Matrix amplification

$$\begin{cases} x_{n+1} = (x_n + dt v_n) \cdot \frac{1}{\alpha} \\ v_{n+1} = \left(v_n + dt \frac{-kx_n}{m} \right) \cdot \frac{1}{\alpha} \end{cases} \Rightarrow \quad \text{où } \alpha = 1 + \frac{kdt^2}{m}$$

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \frac{1}{\alpha} \begin{pmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_n \\ v_n \end{pmatrix}$$

Form: $X_{n+1} = AX_n$

$$\begin{cases} x_{n+1} = x_n + dt v_{n+1} \\ v_{n+1} = v_n + dt \frac{-kx_{n+1}}{m} \end{cases} \Rightarrow$$

$$\begin{cases} x_{n+1} = x_n + dt \left(v_n + dt \frac{-kx_{n+1}}{m} \right) \\ v_{n+1} = v_n + dt \frac{-k}{m} (x_n + dt v_{n+1}) \end{cases} \Rightarrow$$

$$\begin{cases} x_{n+1} = (x_n + dt v_n) \cdot \frac{1}{\alpha} \\ v_{n+1} = \left(v_n + dt \frac{-kx_n}{m} \right) \cdot \frac{1}{\alpha} \end{cases} \Rightarrow \quad \text{où } \alpha = 1 + \frac{kdt^2}{m}$$

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \frac{1}{\alpha} \begin{pmatrix} 1 & dt \\ -kdt & 1 \end{pmatrix} \cdot \begin{pmatrix} x_n \\ v_n \end{pmatrix}$$

Calculating the amplification A matrix
(A such that $X_{n+1} = AX_n$)

The eigen values are:

$$\frac{1}{\alpha} \left(1 \pm dt \sqrt{\frac{k}{m}} \right) = \frac{1 \pm idt \sqrt{\frac{k}{m}}}{1 + \frac{kdt^2}{m}}$$

$$= \frac{1 \pm idt \sqrt{\frac{k}{m}}}{\left(1 + idt \sqrt{\frac{k}{m}} \right) \left(1 - idt \sqrt{\frac{k}{m}} \right)} \quad \text{(Where we use identity : } a^2 + b^2 = (a + ib)(a - ib))$$

$$= \frac{1}{\left(1 + dt^2 \frac{k}{m} \right)^{1/2}}$$

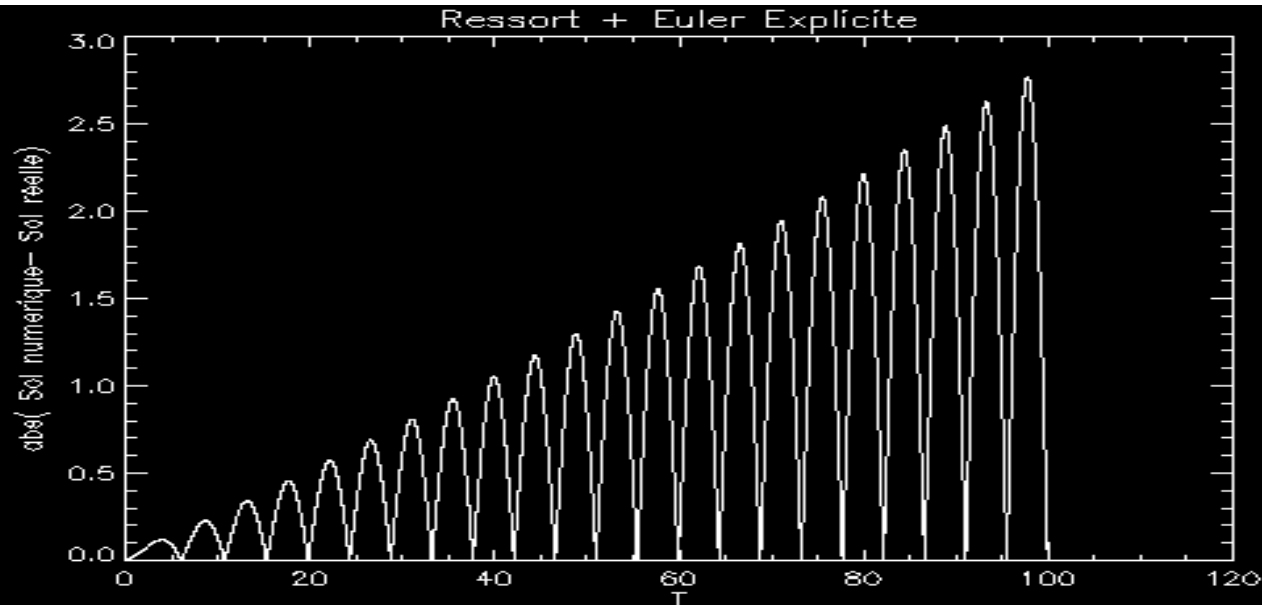
Eigen values norm are:

$$\frac{1}{\left(1 + dt^2 \frac{k}{m} \right)^{1/2}}$$

They are always < 1

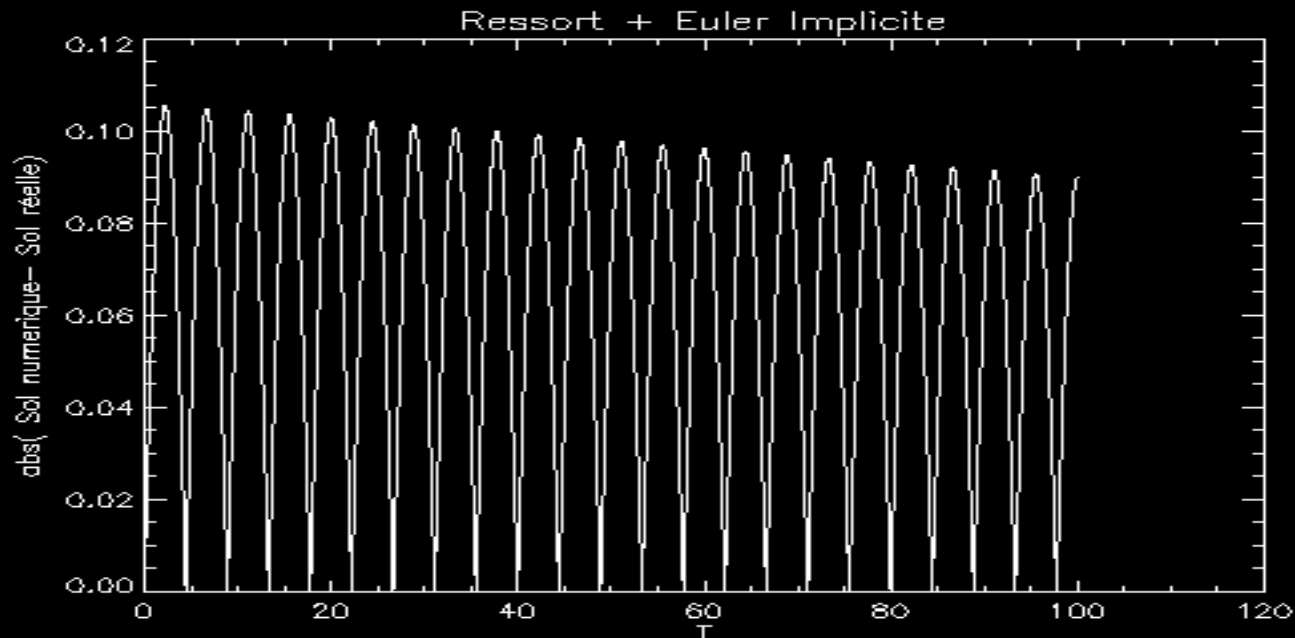
With the problem of the spring, Euler Implicit scheme : *unconditionally stable*

Compare Euler Explicit vs Implicit: Spring Case

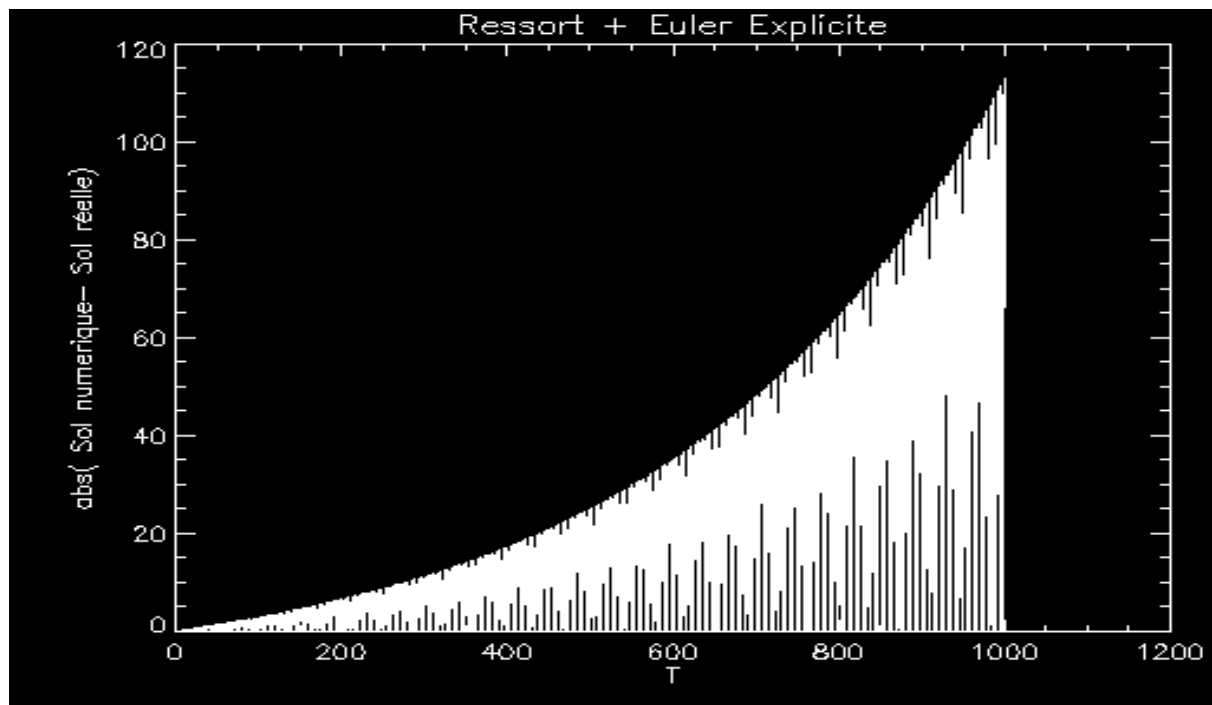


error Explicit

$\Delta t = 0.01$
10000 iterations

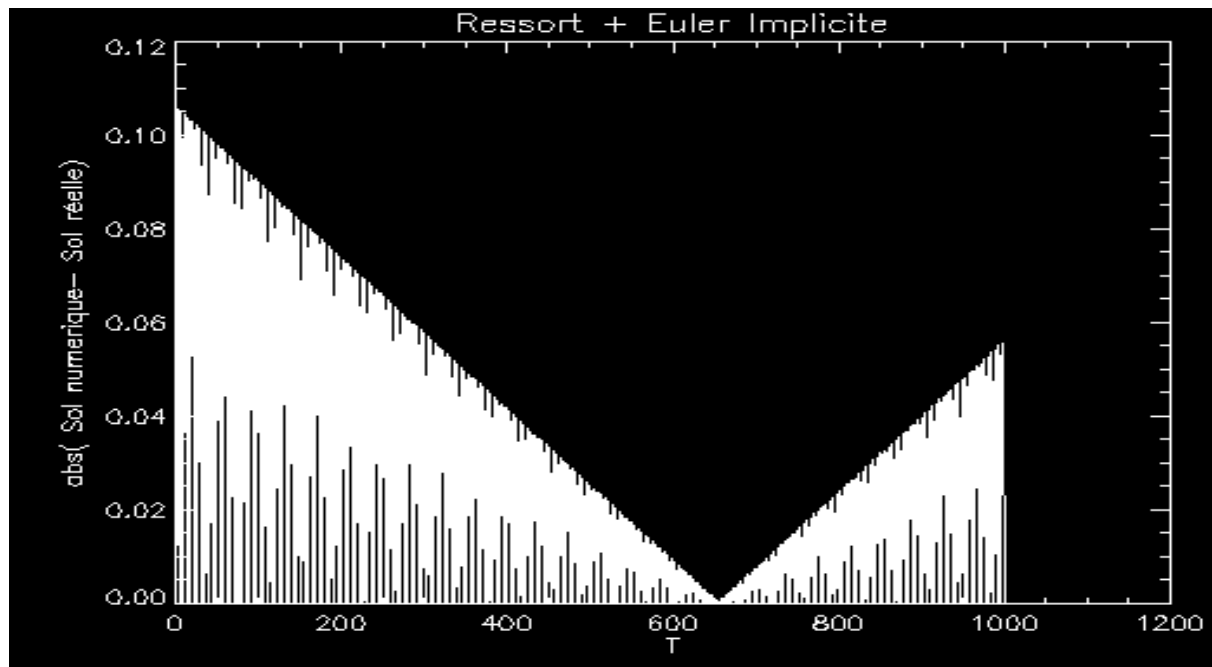


Implicit error



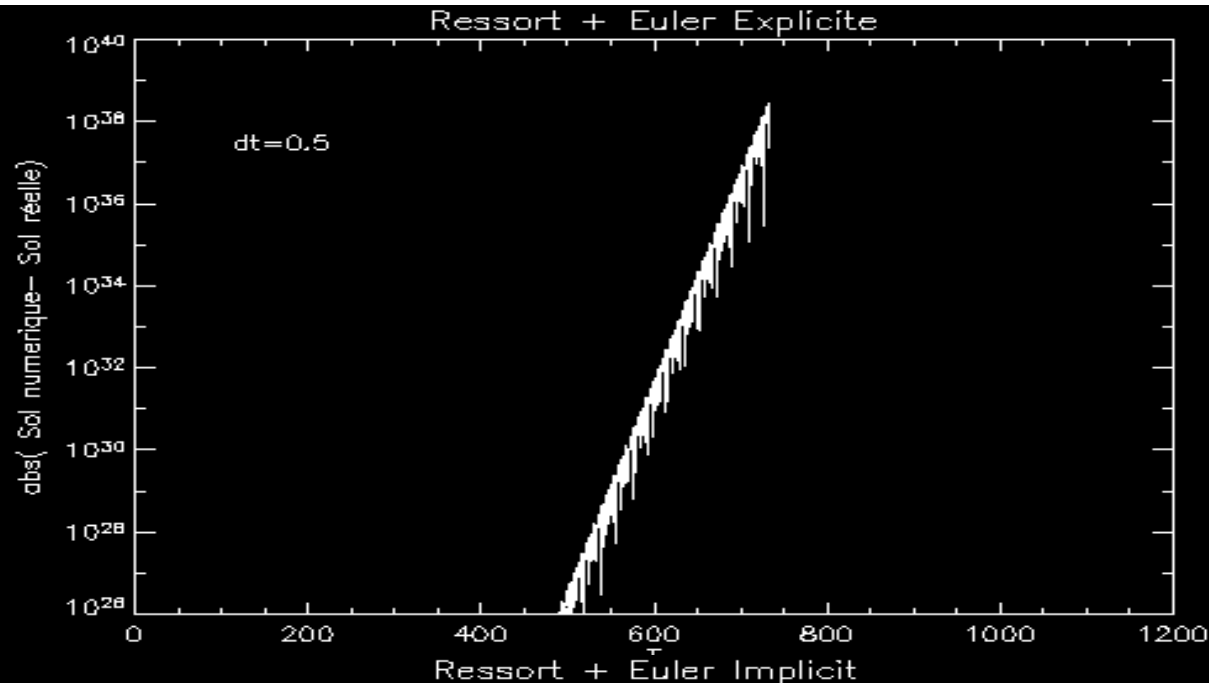
error Explicit

$\Delta t = 0.01$
100000 iterations



Implicit error

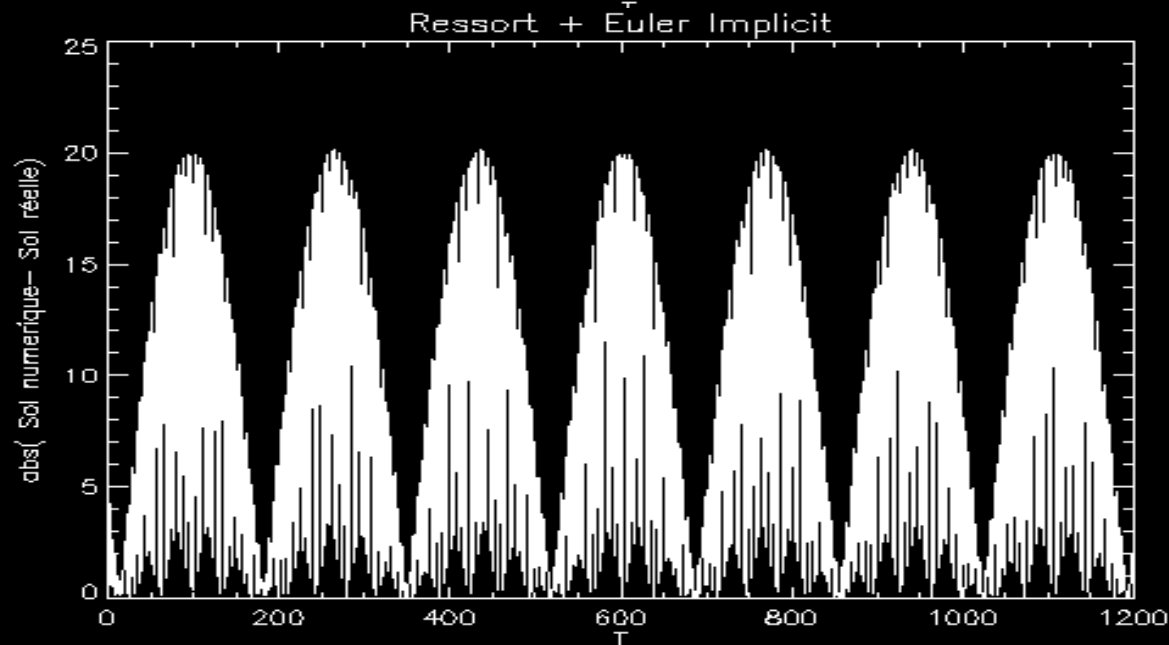
Lare time steap: $dt = 0.5$ (instead of 0.01)



Error for Explicit Euler

.... The solution diverges...

100000 iterations



Implicit Euler Error

The error does not diverge
... but the result is
still wrong

The implicit solver is somewhat less accurate at the beginning of integration

BUT

In the long term, it does not diverge like crazy, even though the solution might be wrong..

THEREFORE

When stability is important, it is interesting to use an implicit solver

What solver choose?

⇒ Trade-off between the computation time, desired accuracy and stability

low-order (1,2) explicit

Fast, very unstable, very precise on short term

low-order (1,2) implicit

Moderately fast, very stable, may be more accurate on long term, than on short term

high order (3.4 +) Explicit

Slow, steady, accurate enough for most application (but not always...)

high order (3,4, +) Implicit

Very slow, very stable, very accurate BUT ... almost never used.

What time step to choose?

General rule: $\Delta t \ll$ all dynamical timescales of the system

It is necessary to **always** thoroughly test the time step !
by controlling certain parameters.

(Typical example: use energy, angular momentum, any quantity that should be preserved, like integrals of motions)

example:

For the spring, the characteristic time is the oscillation period,

$$T = \frac{2\pi}{\omega} = 2\pi \sqrt{\frac{m}{k}}$$

So we should take $dt \ll T$

In our example: $m = 1$, $k = 1 \Rightarrow T = 6.28$ seconds.

With $dt = 0.01$ s OK

With $dt = 0.5$ s PB !! (See above figures)

What to do when there are many different timescales in a problem?

« STIFF problems »

=> We are constrained by the smallest time-scale....

Example: The Solar System:

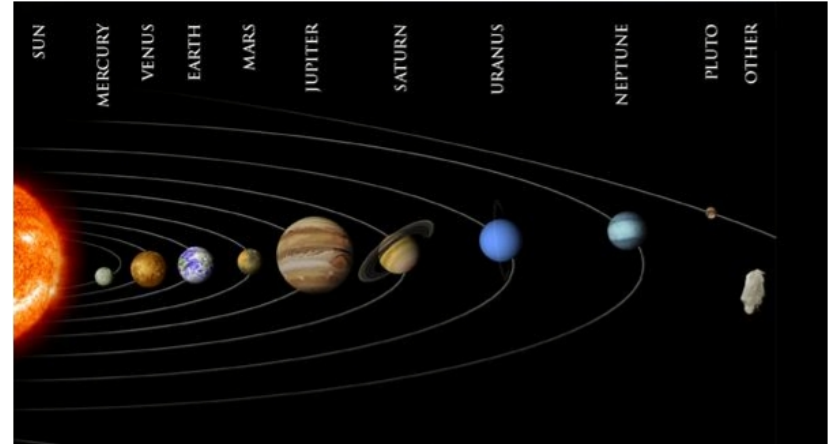
Orbital Period

Mercury: 88 days

Earth: 1 year

Jupiter: 12 years

Pluto: 248 years



ALL the planets interact (coupling)

In this system, it has a factor of 1000 between the shortest time dynamic and the longest ...

What to choose for dt ??

We have no choice: $dt \ll 88$ weeks ...

Conclusion The majority of computing time just serves to integrate Mercury

A bad idea :



Integrate planet motion with different time step for each planet

⇒ The result will invariably be false because different variables
the systems will not be SYNCHRONIZED !!

For example:

Mercury dt , $2dt$, $3dt$, $4dt$, $5dt$, $6dt$, $7dt$, $8dt$, $9dt$, $10dt$

Earth $2dt$, $4dt$, $6dt$, $8dt$, $10dt$

Jupiter $3dt$, $6dt$, $9dt$

Pluto: $5dt$, $10dt$

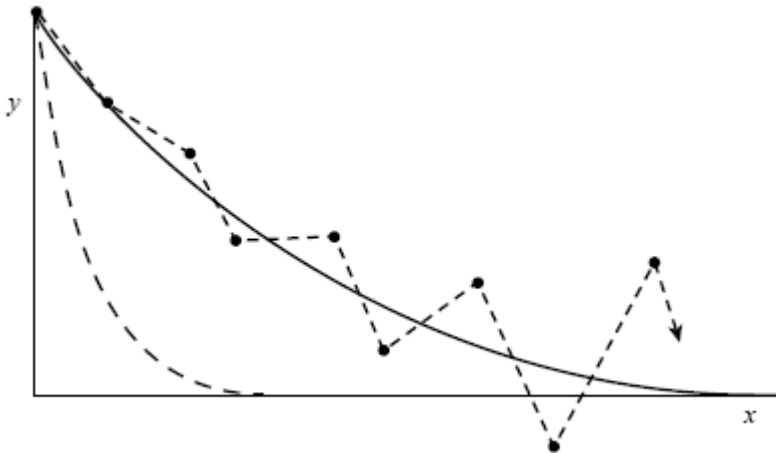
PB: For Mercury we need to know ALL the position
planets to each value of dt ...

Another example: A system with two variables

$$\begin{cases} u' = 998u + 1998v \\ v' = -999u - 1999v \end{cases} \quad \longrightarrow \quad \begin{cases} u = 2e^{-t} - e^{-1000t} \\ v = -e^{-t} + e^{-1000t} \end{cases}$$

With $U(0) = 1, V(0) = 0$

Two time scales: 1 and 1/1000



An explicit method will oscillate between both exp., even after the most Quick is ~ 0 .

Solution: Implicit Method
Whose stability range is infinite.
But no precise ...

CONCLUSION FOR stiff problem

Some methods exist, rather subtle, but do not use a independent timestep for each variable ...

⇒ Often this is wrong

In a system where ALL the variables are coupled together to others, $\Delta t \ll$ smallest dynamical timescale

To avoid excessive instabilities use an IMPLICIT SOLVER solver ... needs some efforts..

Towards adaptive methods:

The adaptive RK: managing error control

Idea: how to control dt to be on that one error is not too large

Several methods exist, A method for adaptive time is complex to implement, but often faster and more accurate. WELL requires knowing the physical system

Difficulty

As we do not know *a priori* the exact solution, it is difficult to estimate the error.

A common method is to realize that if the calculation is wrong, or very approximated, The solution found by the solver should depend *very strongly* of the dt value.

Why ? By that: $\lim (F(X_n)) = \text{Solution}$, when $dt \rightarrow 0$

So when one is far from the solution (contrapositive) F highly dependent on no 81 time.

Idea : Compare different assessments of the solver, is not based on time, or according to the Order of the solver.

We must introduce a precision parameter, Δ_0 **The desired accuracy**

1^{era} technical : Make 2 evaluations result, taking dt and $DT / 2$.
(Double computation time). If both results are equal roughly Δ_0
So the solution is acceptable, otherwise it must reduce the time step.

Simple method but very costly in time :

How many derivative evaluations

4 for time step dt

8 for 2 time step using $dt / 2$, but the first to $dt / 2$ is the same as that at dt
so 11 in total.

To compare with 8 evaluations (one advances $dt / 2$)

So an increase in the computation time of $11/8 \sim 1.4$
40% slower

2nd method: More elegant and faster cheaper better : Adaptive Runge Kutta 5
Fehlberg method for Runge Kutta

Fehlberg studied RK5. It requires 6 calls to the derivative

The RK5 is as follows

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + a_2h, y_n + b_{21}k_1)$$

...

$$k_6 = hf(x_n + a_6h, y_n + b_{61}k_1 + \dots + b_{65}k_5)$$

$$y_{n+1} = y_n + c_1k_1 + c_2k_2 + c_3k_3 + c_4k_4 + c_5k_5 + c_6k_6 + O(h^6)$$

The result is accurate to order 5

But Fehlberg shows that other combination coefficients gives a result with 4th order accuracy (with different coefficients, but same evaluations of derivative)

5th order

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 + O(h^6)$$

4th order

$$y_{n+1}^* = y_n + c_1^* k_1 + c_2^* k_2 + c_3^* k_3 + c_4^* k_4 + c_5^* k_5 + c_6^* k_6 + O(h^5)$$

SO: In calculating the same quantities k_1 to k_6 , we can have two different assessments of the result:

Y_{not} to about 5

Y_{not}^* to order 4

$\Rightarrow \text{ABS}(Y_{\text{not}}^* - Y_{\text{not}})$ Is an estimate of the error in the order of 5

Coefficients table for RK5

Cash-Karp Parameters for Embedded Runge-Kutta Method								
i	a_i	b_{ij}					c_i	c_i^*
1							$\frac{37}{378}$	$\frac{2825}{27648}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$	$\frac{18575}{48384}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$	$\frac{13525}{55296}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0	$\frac{277}{14336}$
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$	$\frac{1}{4}$
$j =$		1	2	3	4	5		

5th order 4th order

We can use an array of coefficients also for the RK4

RK4

$$f_0 = f(x_i, y_i)$$

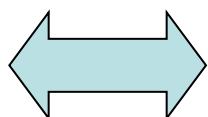
$$f_1 = f(x_i + \alpha_1 h, y_i + h\beta_{10}f_0)$$

$$\dots \quad \dots$$

$$f_k = f(x_i + \alpha_k h, y_i + h(\beta_{k0}f_0 + \beta_{k1}f_1 + \dots + \beta_{k,k-1}f_{k-1}))$$

$$y_{i+1} = y_i + h(c_0f_0 + c_1f_1 + \dots + c_kf_k)$$

i	α_i	β_{ij}	c_i
0	.		$\frac{1}{6}$
1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$
2	$\frac{1}{2}$	0 $\frac{1}{2}$	$\frac{1}{3}$
3	1	0 0 1	$\frac{1}{6}$



$$f_0 = f(x_i, y_i),$$

$$f_1 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}f_0),$$

$$f_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}f_1),$$

$$f_3 = f(x_i + h, y_i + hf_2)$$

$$y_{i+1} = y_i + \frac{h}{6}(f_0 + 2f_1 + 2f_2 + f_3).$$

Suppose we have two estimates of the result,

X_n and X_n^*

The error $\Delta \sim ||X_n^* - X_{\text{not}}|| \propto dt^5$

We seek the new timestep so that $\Delta / \Delta_0 = \text{Precision required}$

So we have $(dt' / dt)^5 = \Delta / \Delta_0$ $dt' = \text{new time step}$

$$\Rightarrow dt'/dt = (\Delta_0 / \Delta)^{1/5}$$

This serves two purposes:

- 1- If the error is too large time step decreases
- 2- If the error is too small time step increases \Rightarrow saves time!

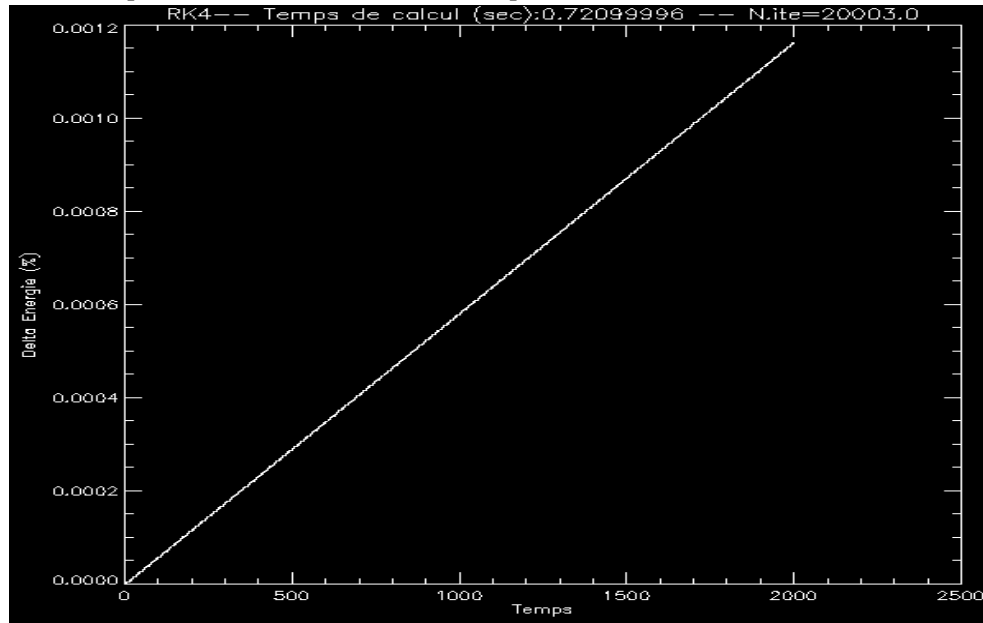
A typical adaptive RK5. scheme

1. Evaluates Y_{not} and Y^*_{not}
2. Calculate Δ
3. Calculate dt'
4. If $dt < dt'$ return 2 Verifier
5. Replace dt by dt'
6. Return 1 (the next time step)

CAUTION: It is generally safe to work with no Adaptive time because no one miscalculated time can distort the calculation. You have to control the instabilities.

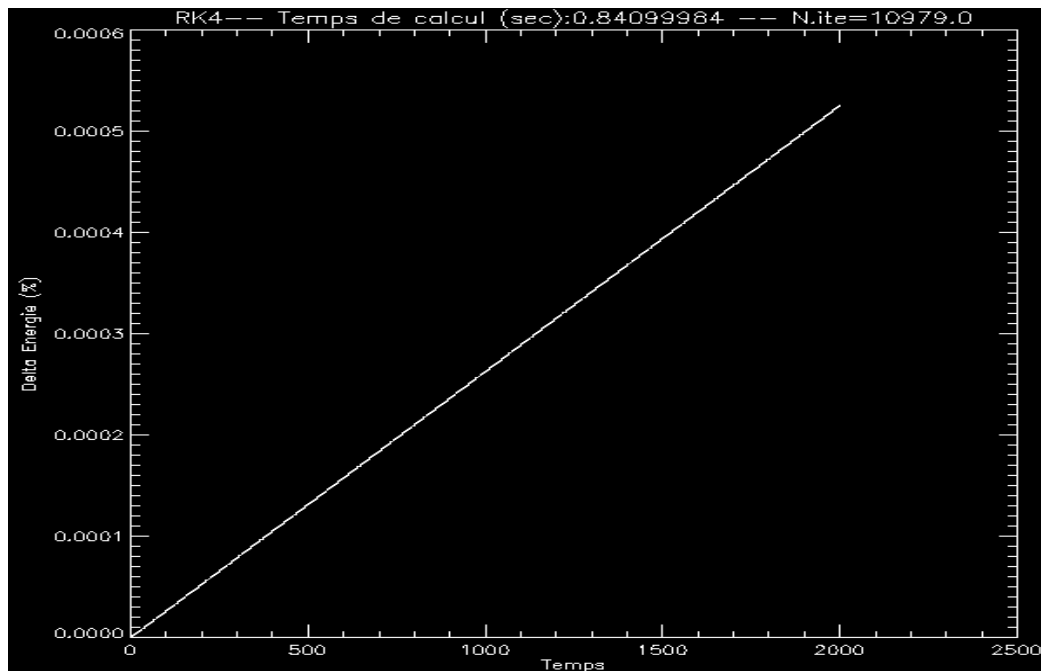
BUT if it works, it is worthwhile to use

Example: RK4 Vs. Adaptive RK5. Problem of the planet



RK4 Energy

Dt = 0.2

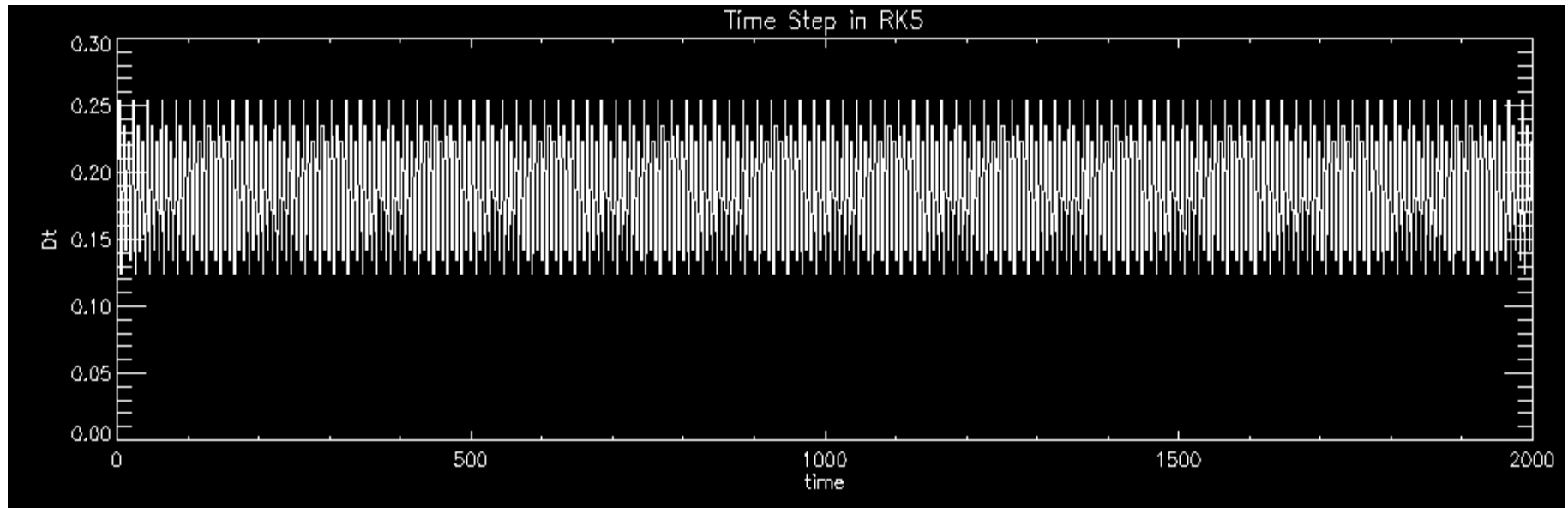


Adaptive Energy RK5

**The error on E increases
2 times slower**

With comparable computing time

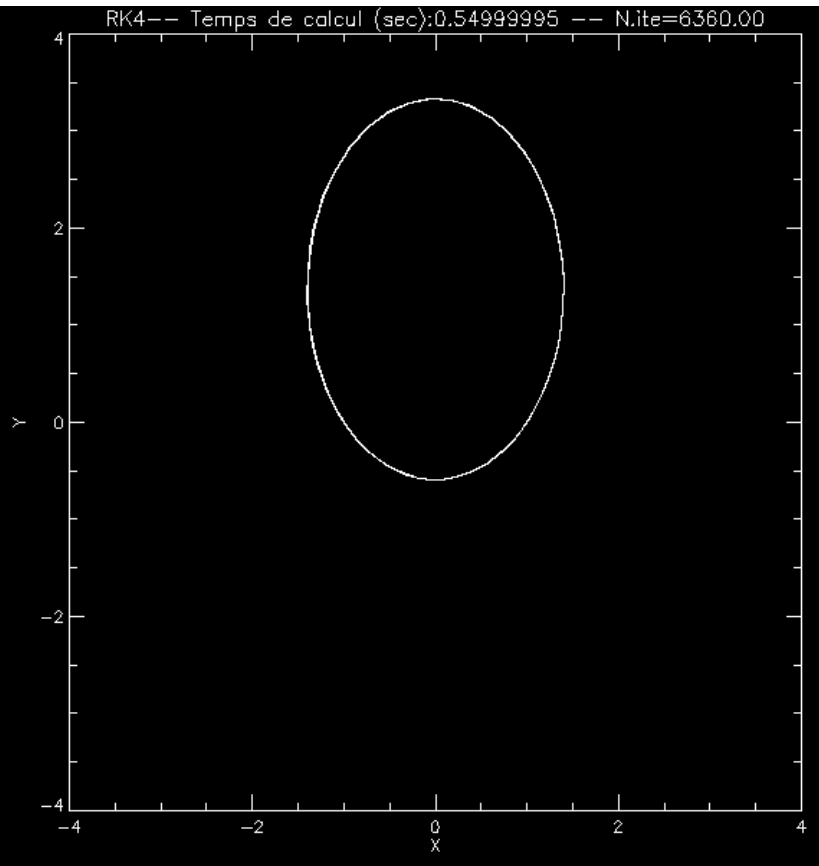
Here is the time step of the Adaptive RK5



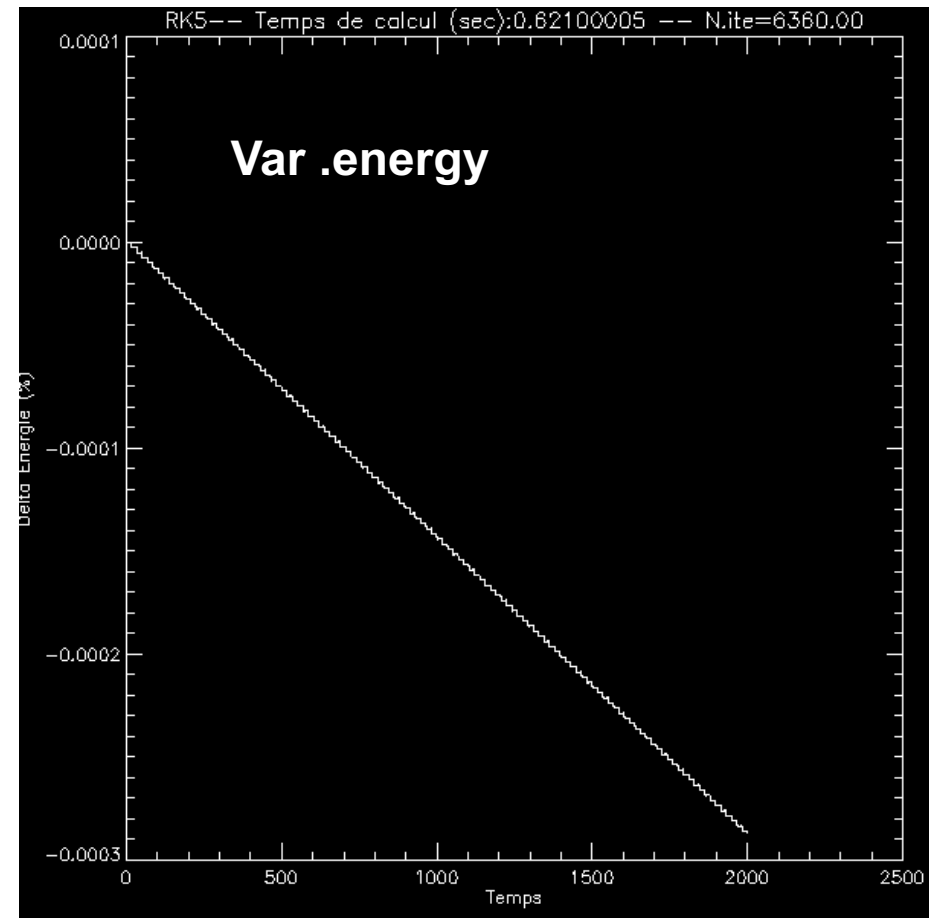
dt decreases as the planet accelerates (perihelion)

dt increases when the planet decelerates (Aphelion)

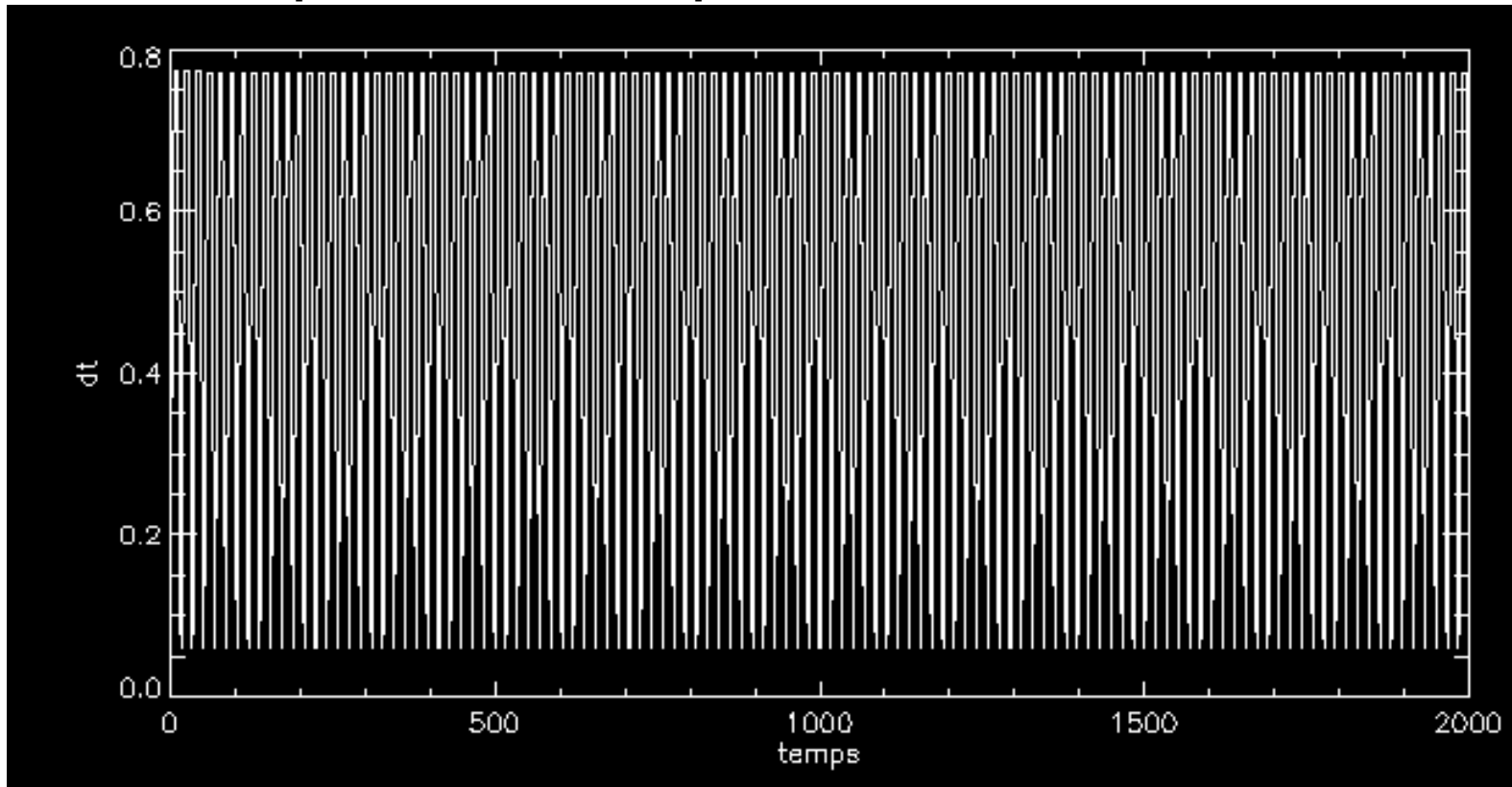
Consider a **VERY** elongated orbit (difficult to integrate)



adaptive RK5

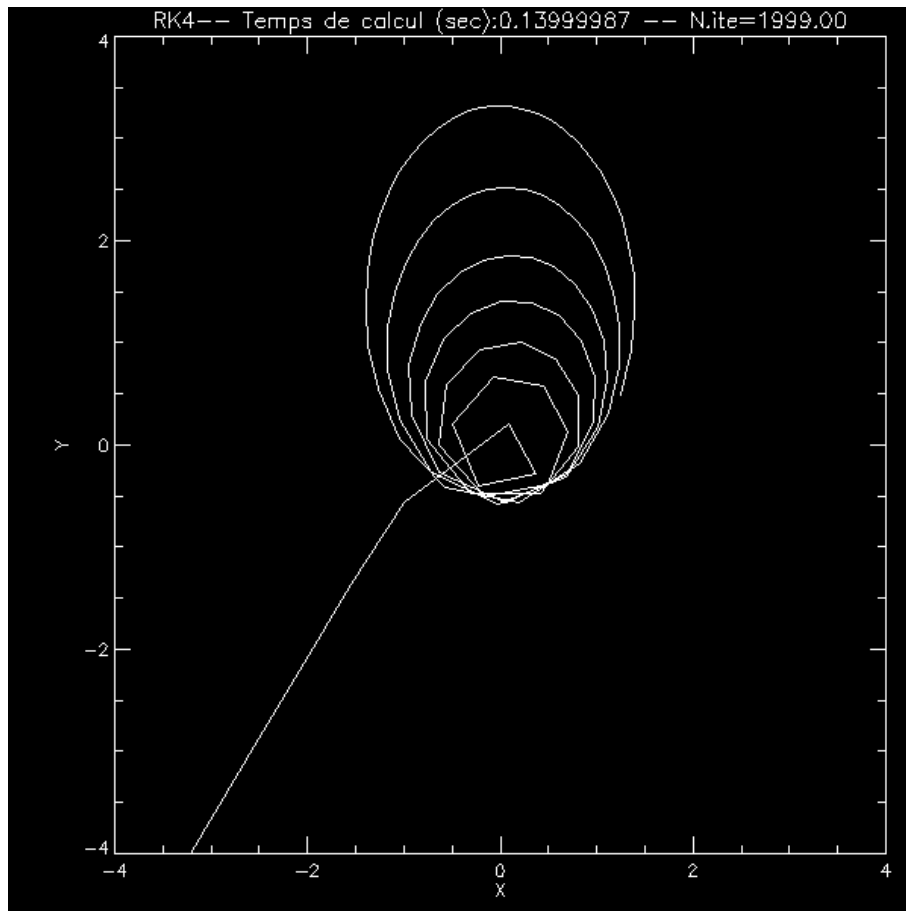


Evolution of adaptive RK5 time step:



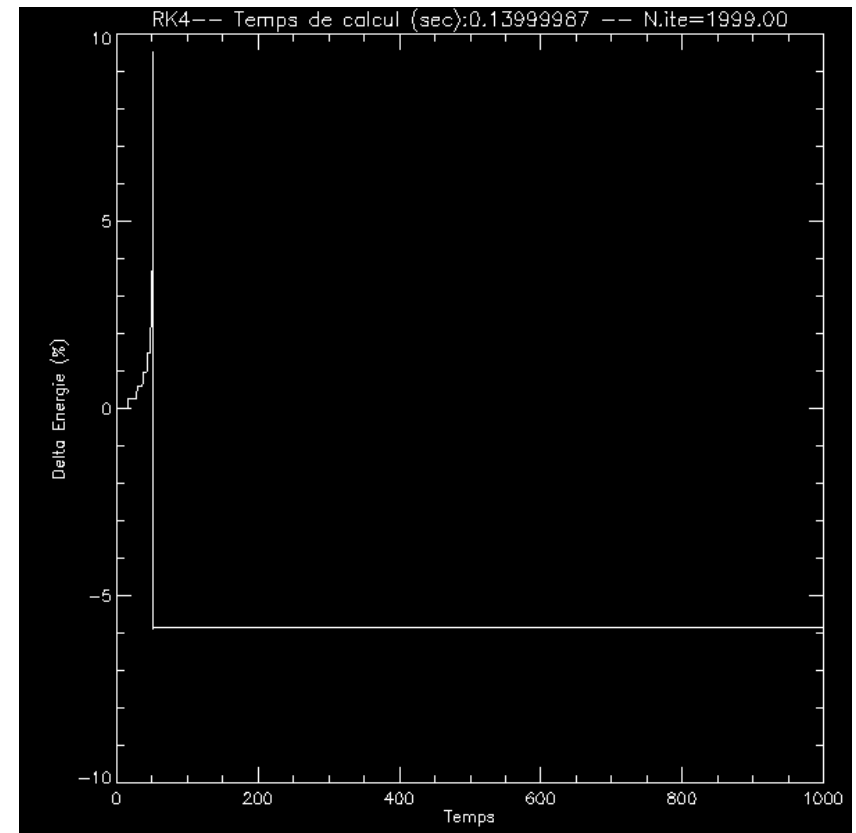
**The time step adapts to the orbit.
Initial time step: 0.5**

RK4 DT = 0.5, same initial conditions



Hmmm

Energy



CONCLUSION

1. Choosing the solver depends on the problem
(Single Issue? Stiff Problem? Etc ...)
2. A high order solver does not mean ALWAYS higher accuracy
3. Sometimes a Implicit solver can simplify your life and increase accuracy

- Do not believe the result of a solver too rapidly !!!
- You must always keep a critical viewpoint in front of a numerical integration
- You must define control parameters

4. Always check what is done
Compare analytical solutions, control energy if possible

5. Use adaptive time-steps with * lots * of precautions