Numerical methods for differential equations

1. Ordinary Differential Equations (ODE)

Sebastien CHARNOZ & Andrea CIARDI

Paris Physics Master
Université Paris Cité / Sorbonne Université

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:

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

- 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** must 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 *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) : position and velocity as a function of time

For heat: T (x, t) : Temperature as a function of space and time

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

A problem is well-posed if we have

- **A list of quantities that evolve according a list of parameters**

- **A differiential equations linking all quantities to all parameters. As many as. diff equations as quantities.**

- a set of initial, or boundary, conditions

EXAMPLES

Wave propagation

Parameters: X (position) and T (time)

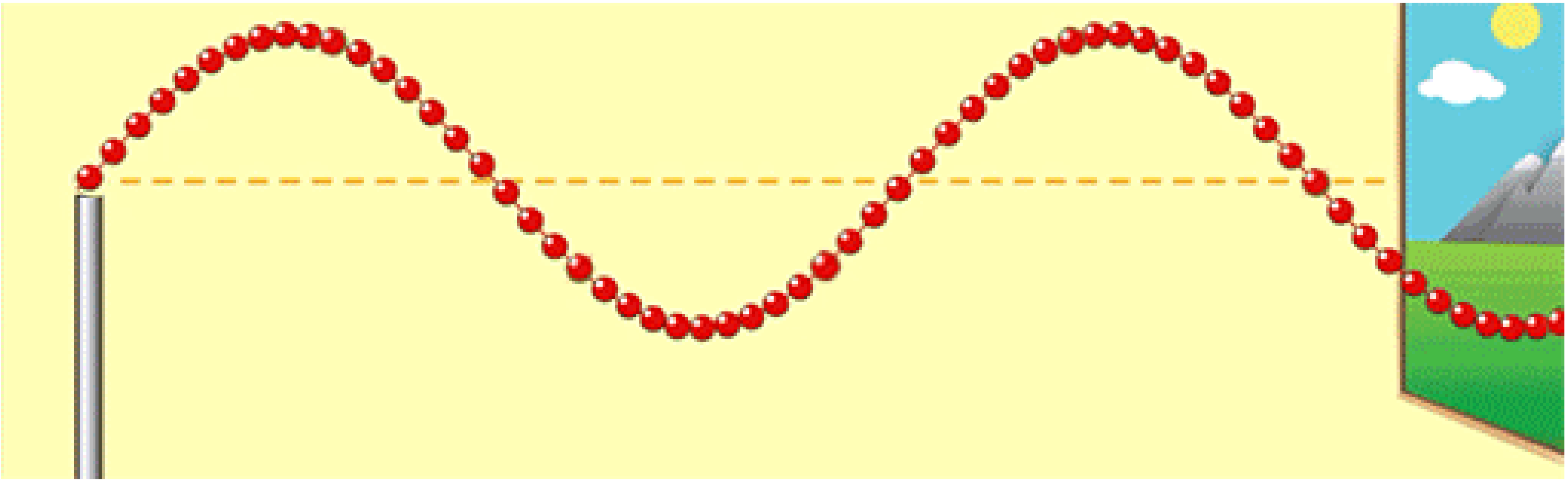
Quantity: U (x, t): Amplitude at position X and at time t

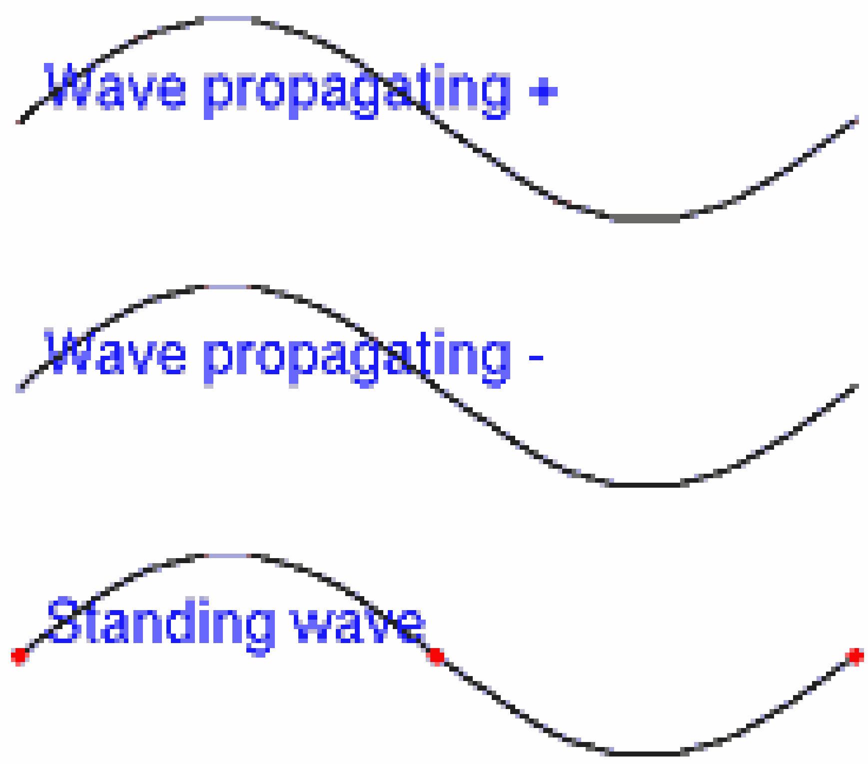
equation: $$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f(x,t)$$

forcing

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

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




Wave propagating +
Wave propagating -
Standing wave

Planetary motion

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

Parameter: Time

Differential equations

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

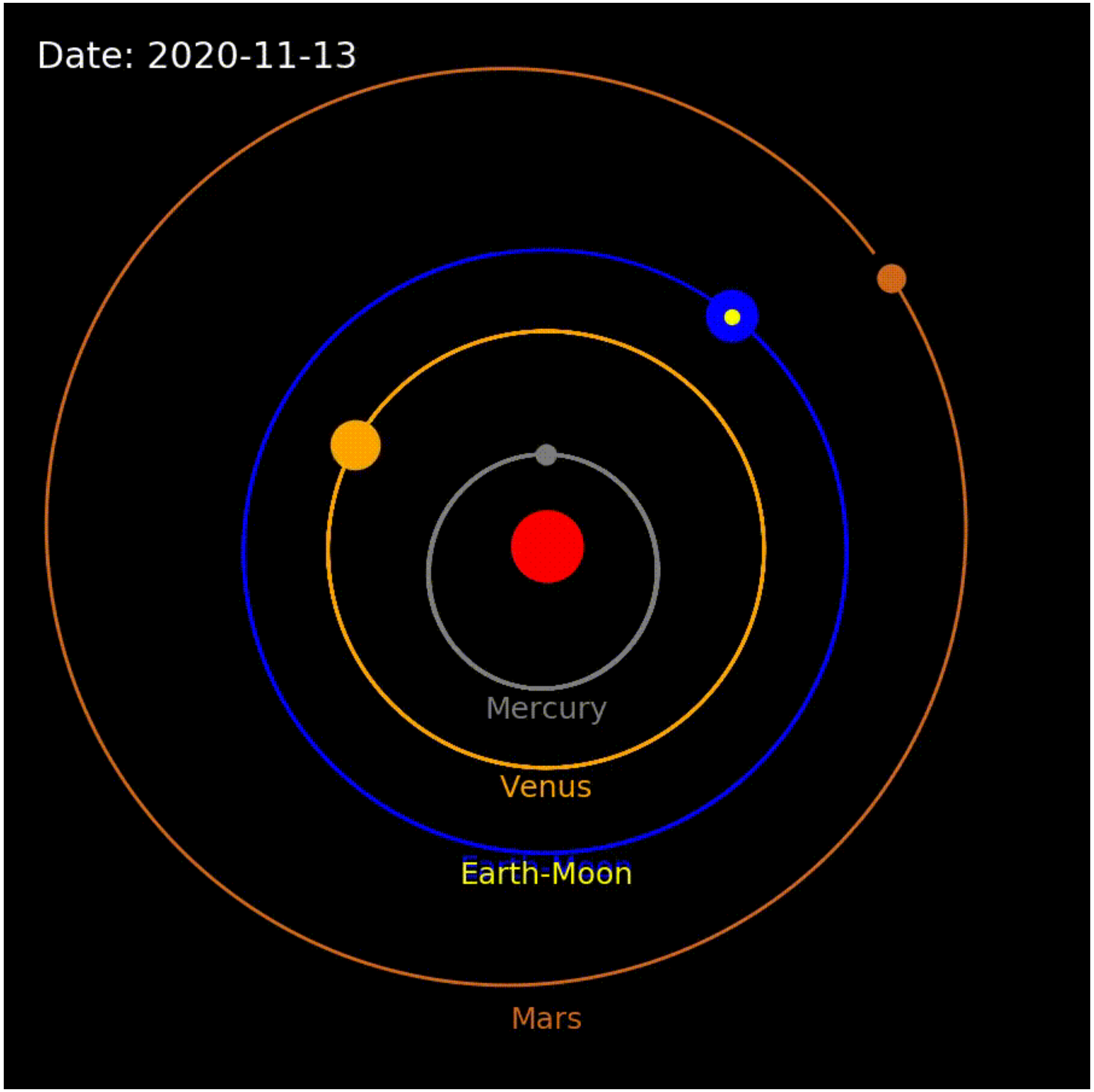
$$\frac{dV_x}{dt} = \sum_{j \neq i} \frac{Gm_j}{r_{ij}^3} (\vec{r}_i - \vec{r}_j)$$

Same pout Y and Z

6 in total

Initial conditions: Initial Position and speed

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


Date: 2020-11-13
Mercury
Venus
Earth-Moon
Mars

Propagation of heat

Quantity: temperature T

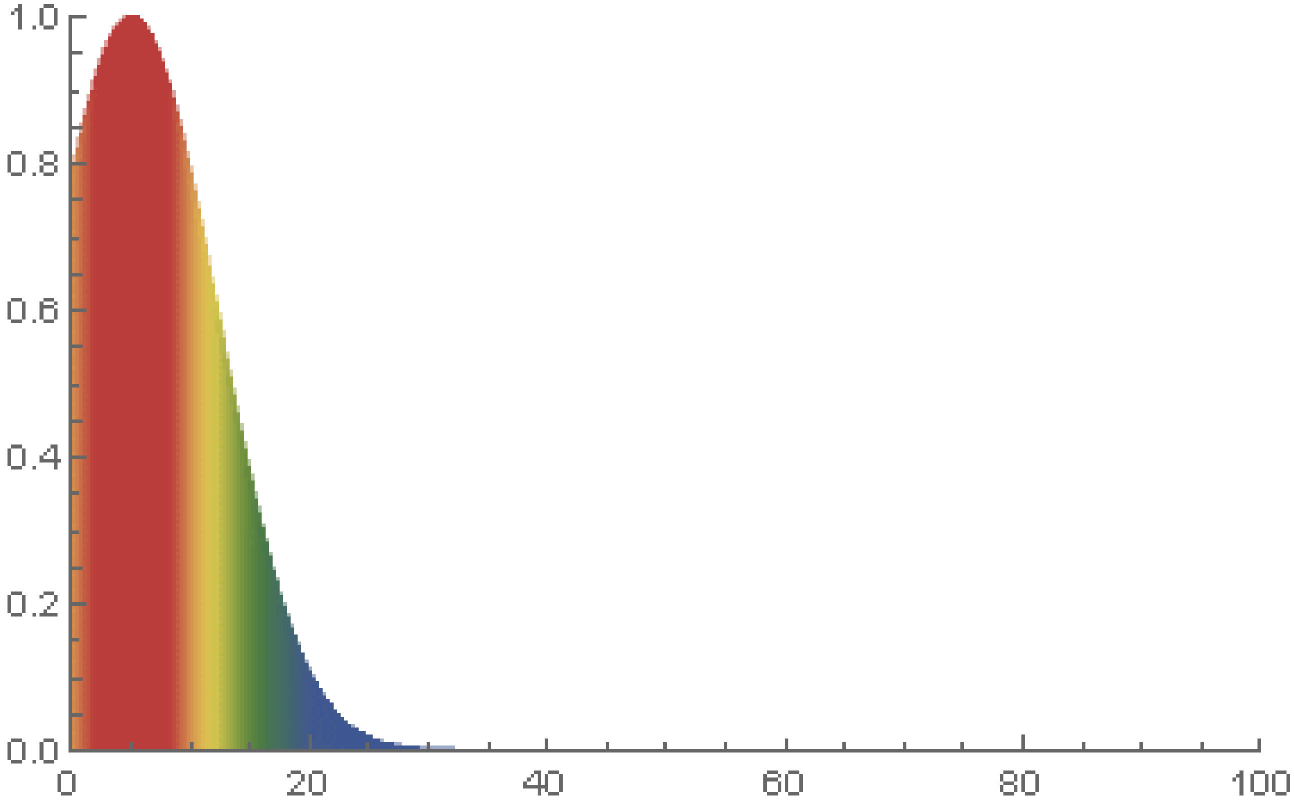
Parameter: X (space) and t (time)

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


1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

RESOLUTION: THE METHODS

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

Discretization of the problem

The parameters are discretized :

example:

If time is a parameter then :
time is written $\mathbf{t(n) = t_n = N * dt}$ where **dt** is the time step

If space is a parameter then :
Then position : x**(n)**=$\mathbf{x_n = n * dx}$ where **dx** will be the space step

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

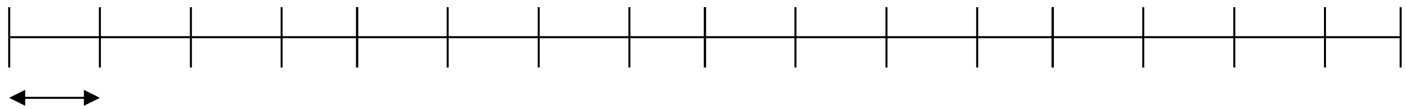
A **time grid, a space grid etc ...**

The smaller the step (dt or dx) the closer the numeric solution will be to the exact solution

---

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

0 1 2 3 4 5 6 ....



dt

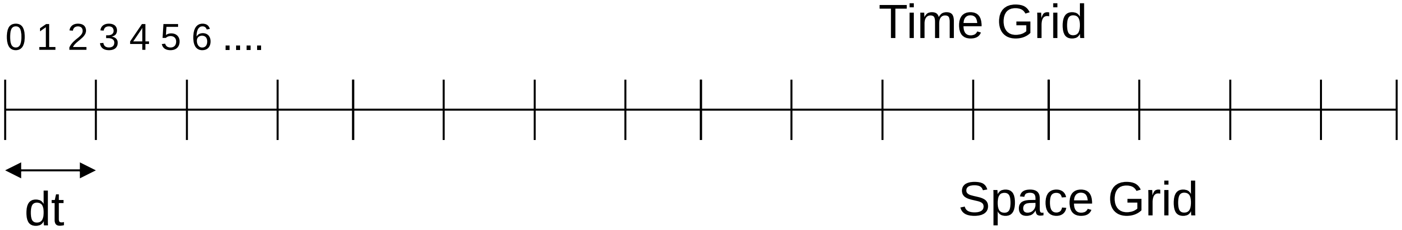
If 1D (1 single parameter, eg time)

The resolution consist in calculating a Serie :

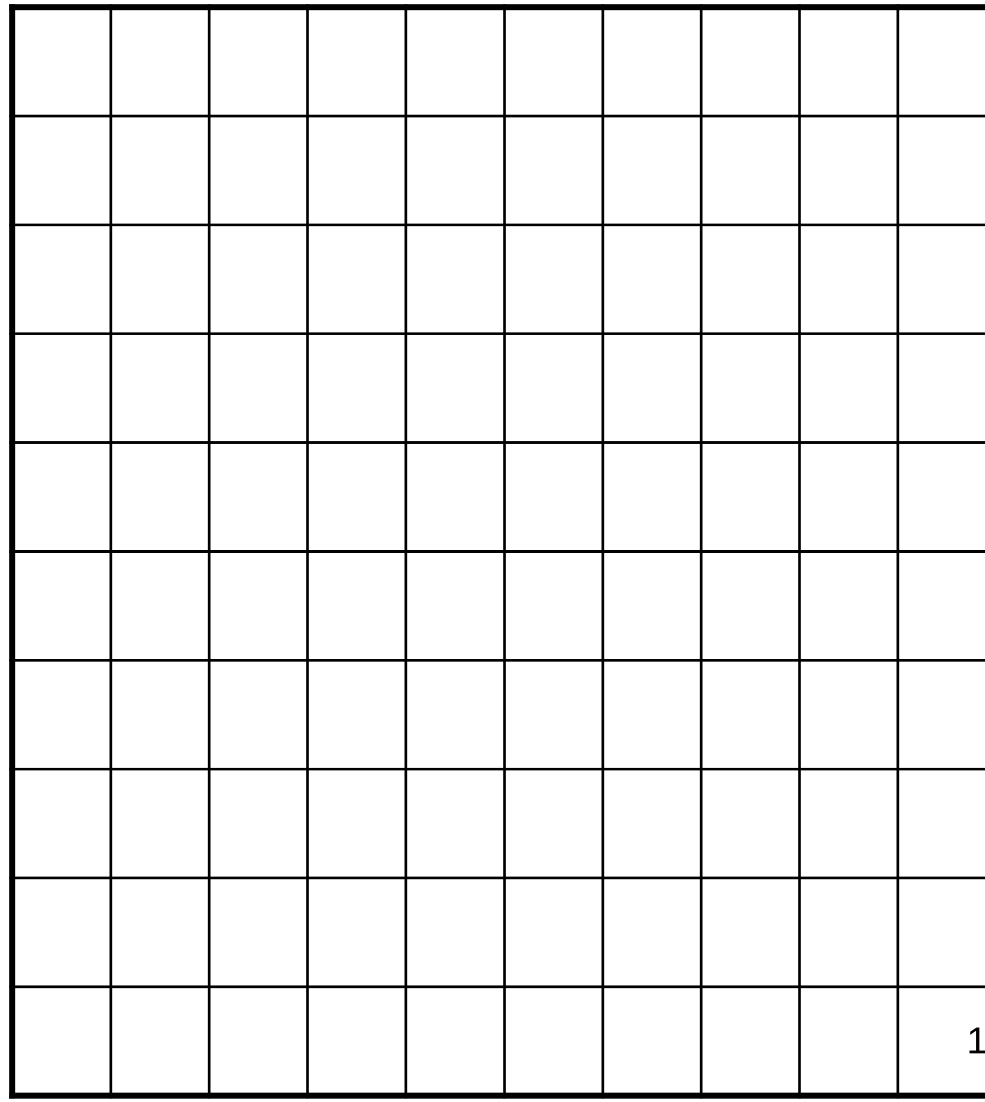
$$U_{n+1} = S(U_n, t_n)$$

That means : Solution at next step = S(solution at previous step)
The big question is: WHAT IS F ?? S is « the solver »

Time Grid



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 make a 3D grid
(2D space + time 1D)

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

For now

we are only interested by Ordinary Diferential Equations:

ODE

Differential equations that dependi only on ONE parameter !

(only one parameter at the denominator of the derivative)

Example ODEs:

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

Why ? Because only One parameter (t or z)

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

BUT

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

Why ? Because > 1 parameter (2 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 write the following

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

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

The function S is called « 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 :

* accurate
* fast
* robust.

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

2. Example of numerical resolution

1 Movement of a spring: A simple ODE

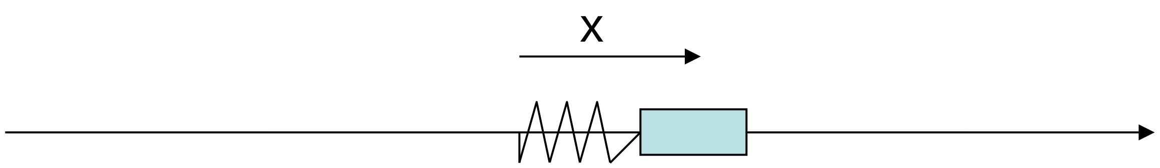
Quantities X and Vx

Parameter: t

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

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

k: coefficient. Spring stiffness
m: mass



Do we need something more ?

YES! because we lack the X evolution equation:

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

In the above equation, x appears on th RHS but its evolution is not given. So we need an additional equation for x

So the full set of Equations is :

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

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

Parameter t

Initial conditions : X (t = 0) = 10 m
V (t = 0) = 0. m / s

Equations :
k: coefficient. stiffness
m: mass

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

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

ANALYTICAL SOLUTION:
*** demonstrate**

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

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

ω = sqrt (k / m), A = X0

We now solve the SAME problem
But numerically integrating the equation.

We need an algorithm to calculate the evolution
Of the solution with small time increments dt.

We will discover the "problems" of numerical integration

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

$X_{n+1} = S(X_n, V_{n,} T_n)$
$V_{n+1} = S(V_n, V_{n,} T_n)$, where $t_n = N * dt$

S 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 $V_{n+1} = F(Vn, tn)$
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

quantities

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

Parameter (single)

Vector Writing

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

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 explicitely depend on time t, because the force does not depend explicitly time

F_x= V and F_v= -Kx / m

Vector Writing

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

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 explicitely depend on time t, because the force does not depend explicitly time

F_x= V and F_v= -Kx / m

Real equation:

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

Numerical approximation

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

where t_{n+1} = (N + 1) x dt

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

f is the derivative, F is the solver

The Euler Method

The basic tool for building **F(MAIN PARAMETER)** is the **Taylor expansion**:

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

$$\boldsymbol{f} = \frac{\boldsymbol{\partial X}}{\boldsymbol{\partial t}}$$

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

It's possible !

Using the development of taylor :

Ignored terms

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

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

Function, Equa. diff system

Inspired by this development, the function F will be:

Where F is a numerical approximation the derivative !!

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

How to build S:

The simplest case is the Euler Method:

Euler Method :

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

For Euler we just set F (x, t) = f (x, t)

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

Example: Spring Case with Euler

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

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

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

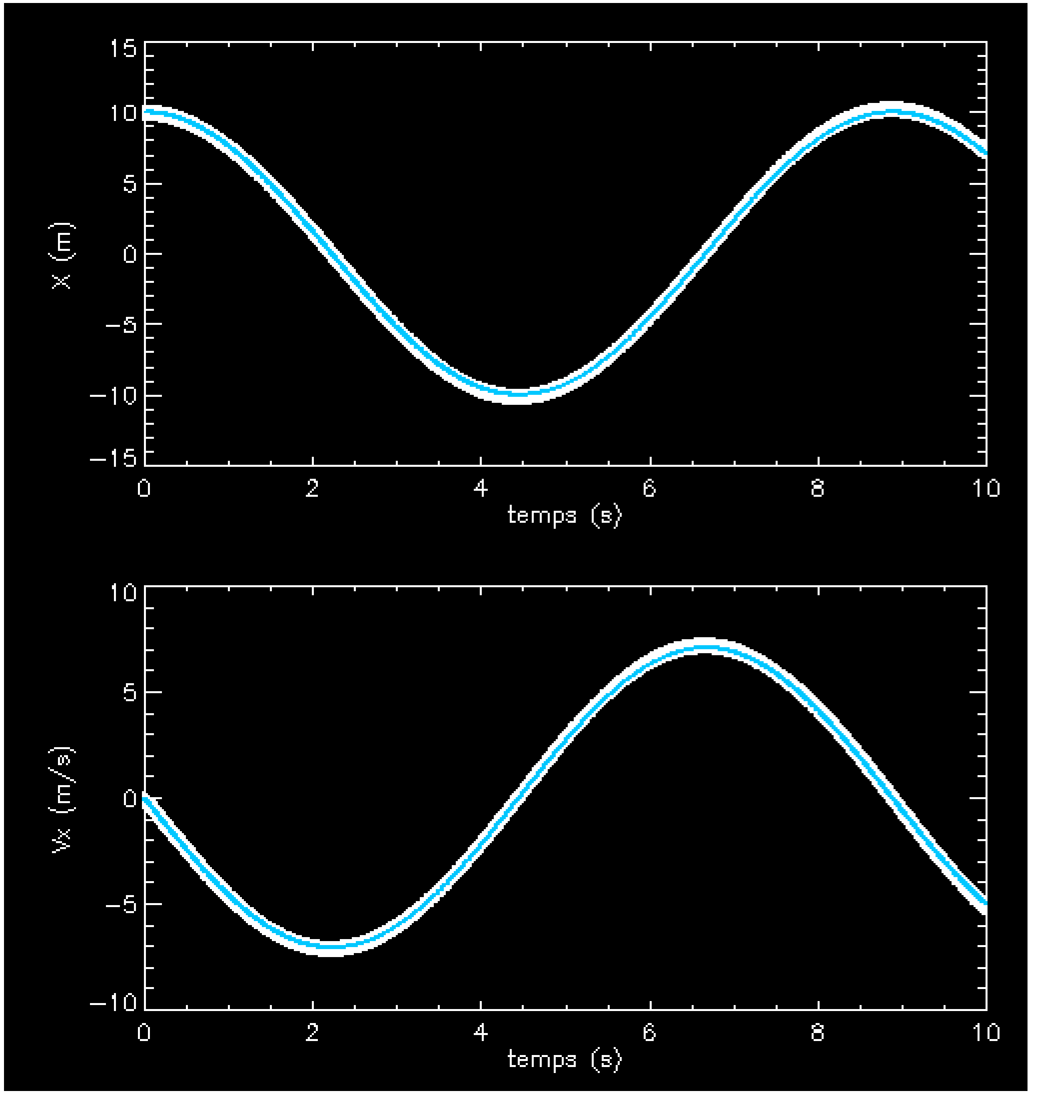
But we can do much better!

numerical solution: spring solved by Euler

dt = 0.01 s

N t XV

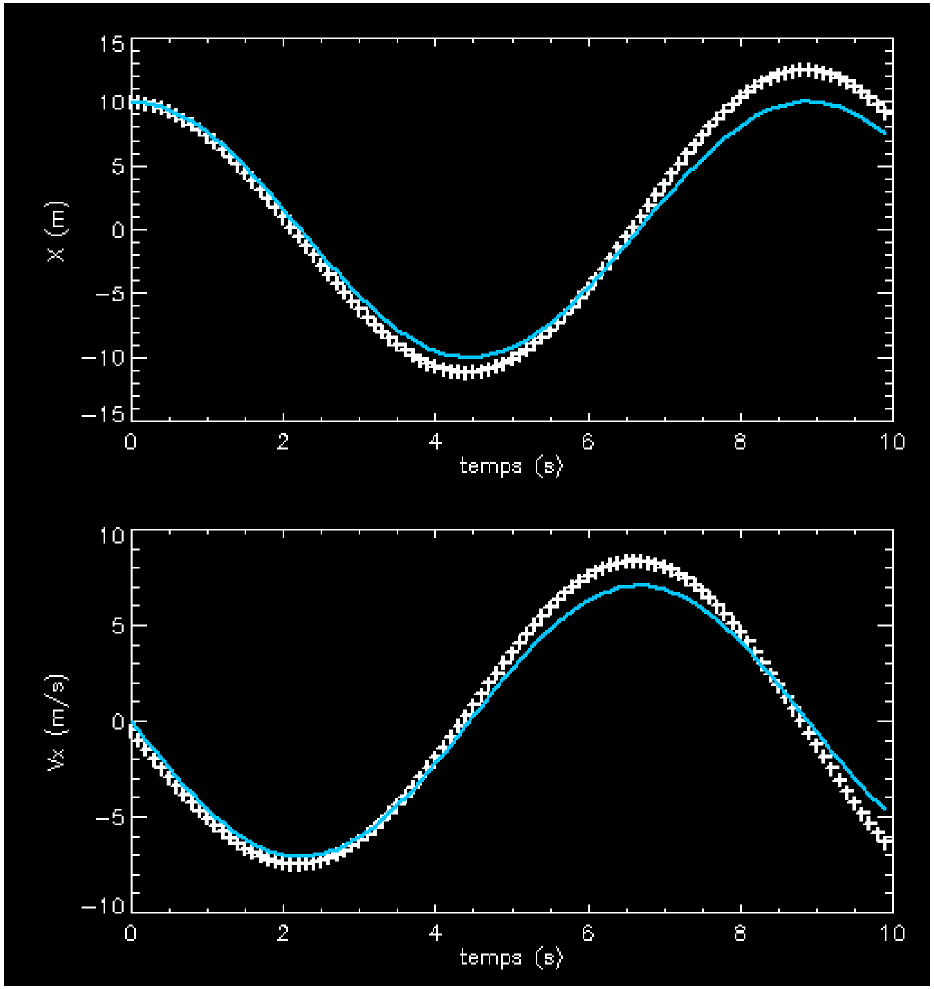
0.000000 0.000000 0.000000 10.0000
1.00000 0.0100000 10.0000 -0.0500000
2.00000 0.0200000 9.99950 -0.100000
3.00000 0.0300000 9.99850 -0.149998
4.00000 0.0400000 9.99700 -0.199990
5.00000 0.0500000 9.99500 -0.249975
6.00000 0.0600000 9.99250 -0.299950
7.00000 0.0700000 9.98950 -0.349912
8.00000 0.0800000 9.98600 -0.399860
9.00000 0.0900000 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

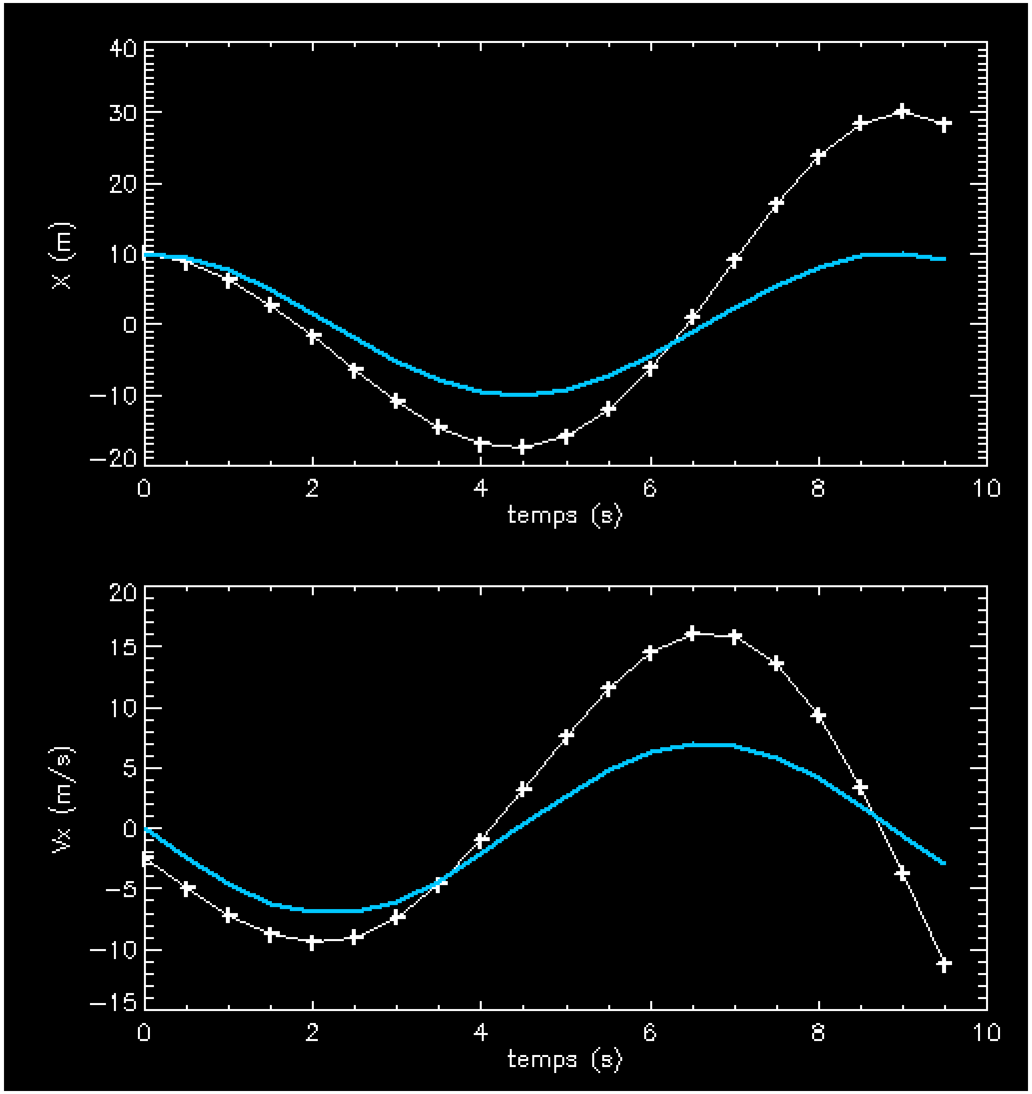
But the numerical solution depends of the time-step

dt = 0.1 s



The bigger dt, the larger the error

dt = 0.5 s



FOR ANY SOLVER:

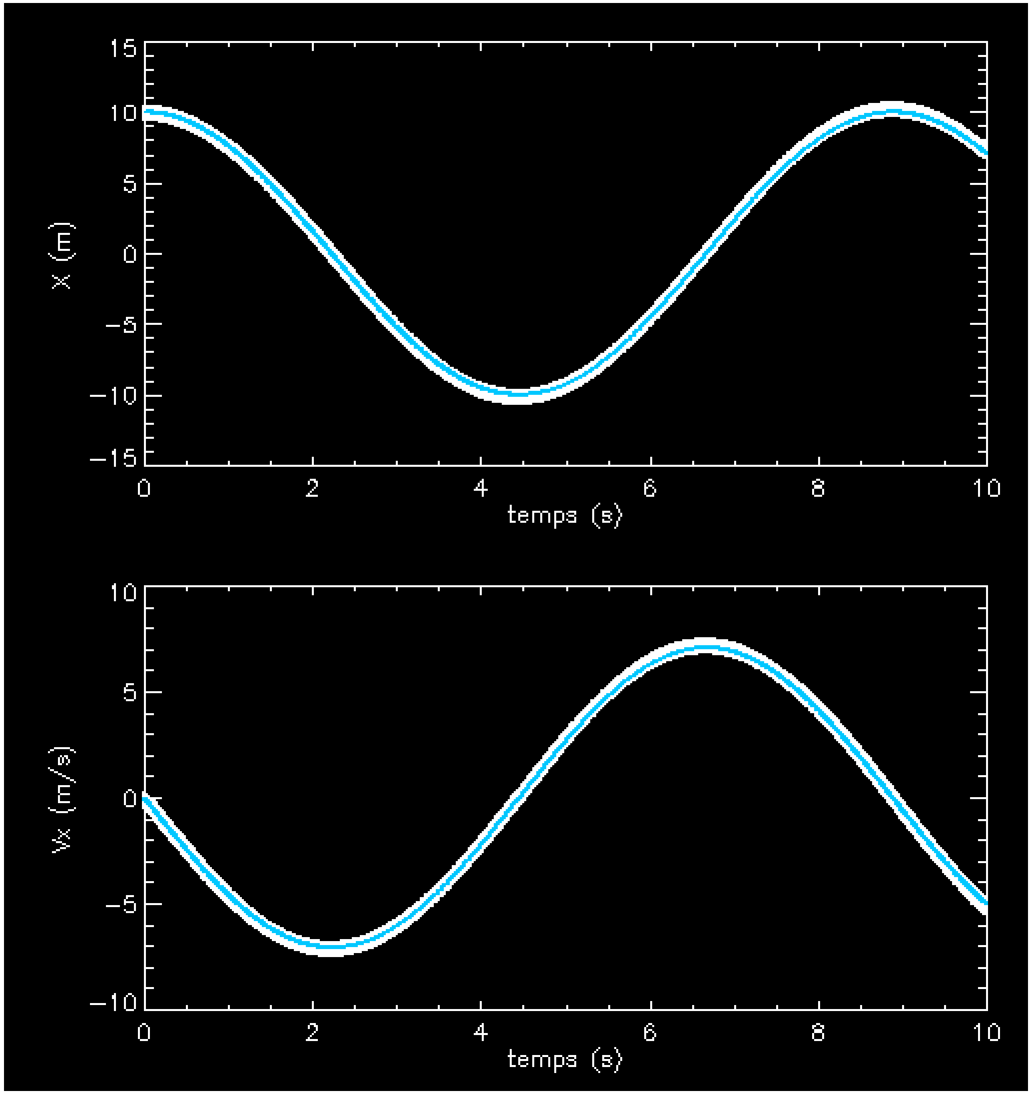
Any numerical solution
is only approximate

The accuracy depends on the
time-step of integration

For larger dt: computing is faster
BUT is less accurate

And vice versa ...

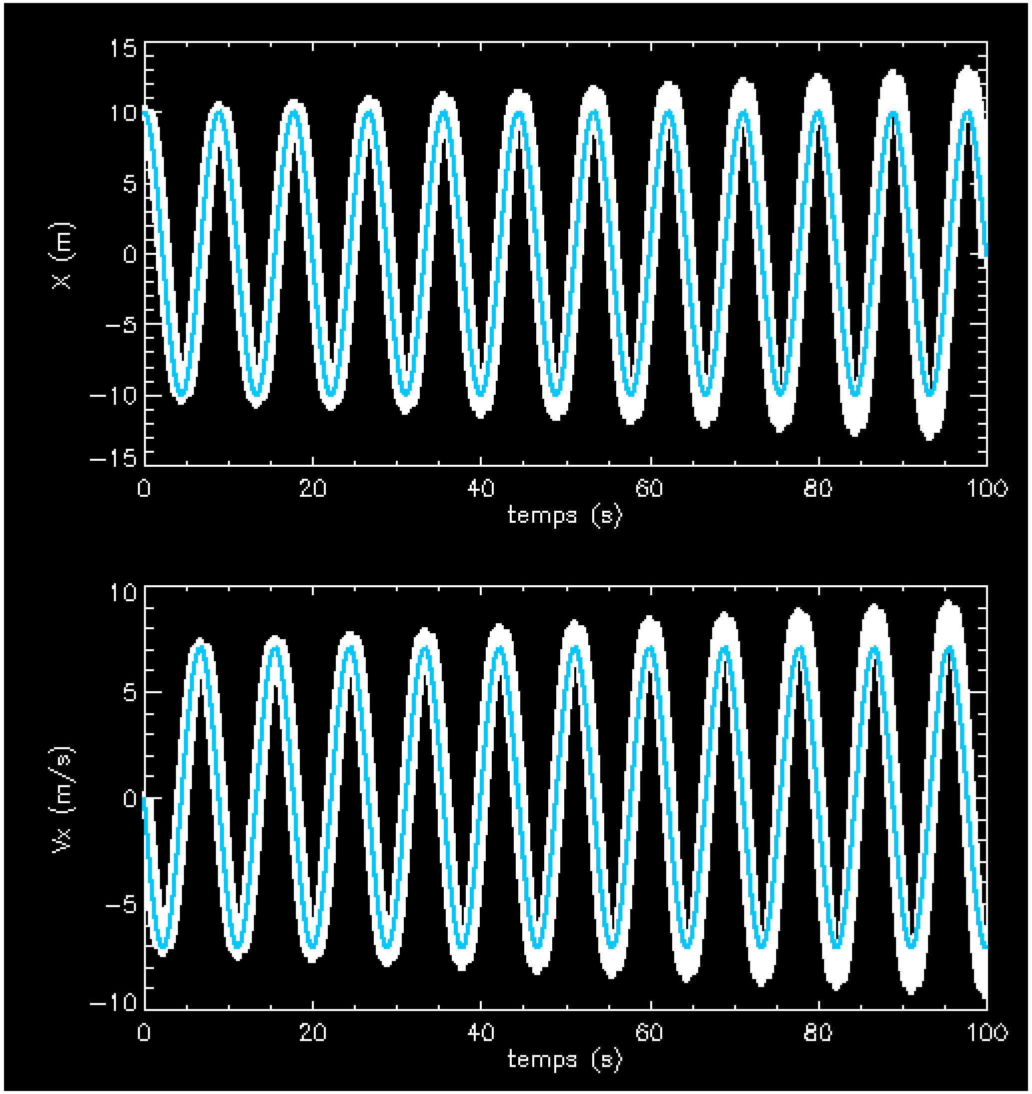
The accuracy decreases as the number of stages of calculations:



Dt = 0.01 s

1000 calculation steps

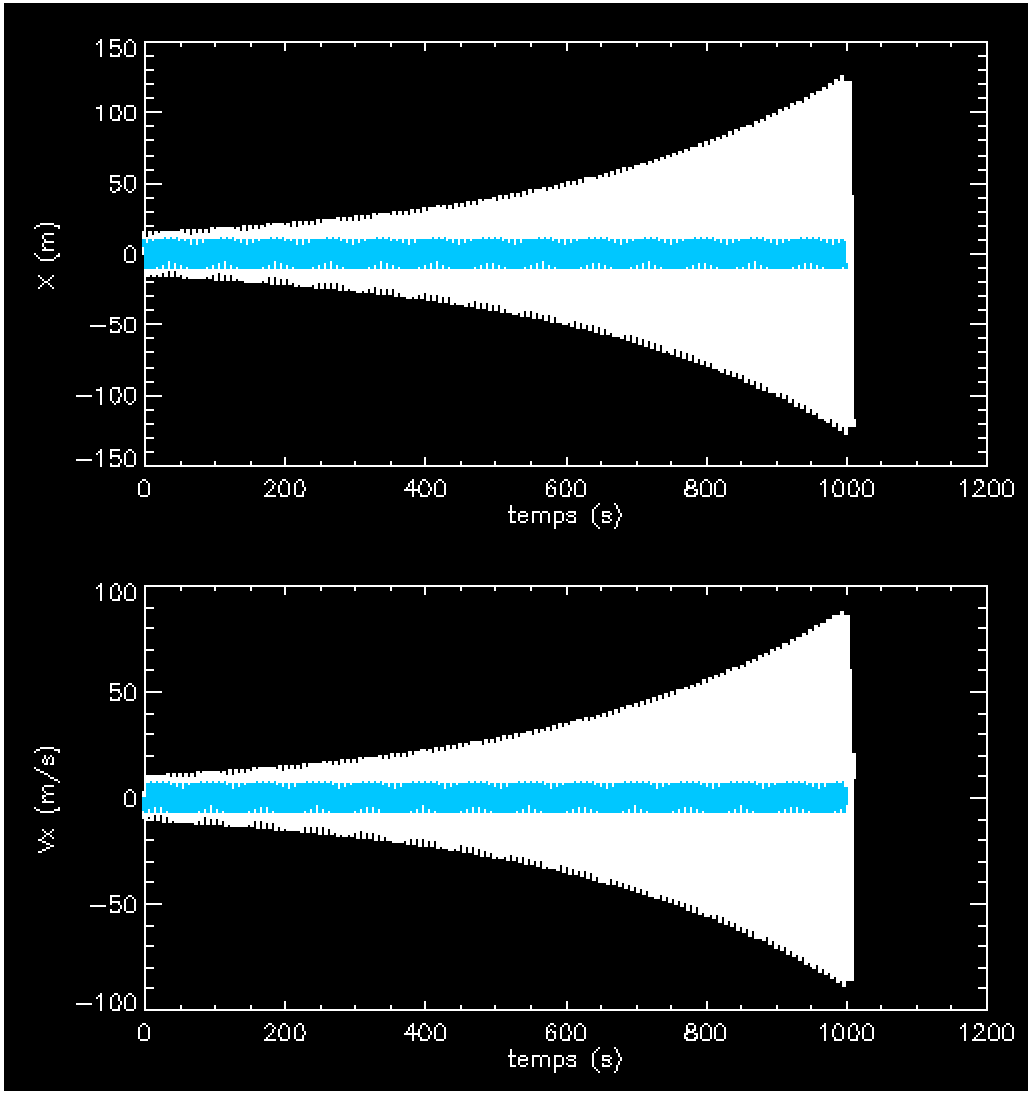
Looks good !



Dt = 0.01 s

10000 steps

still good !



Dt = 0.01s

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 :

$$\boldsymbol{X_{n+1} = X_n + \int_t^{t+dt} f(t, X(t), ..)dt = X_n + dtF(t, X_n)}$$

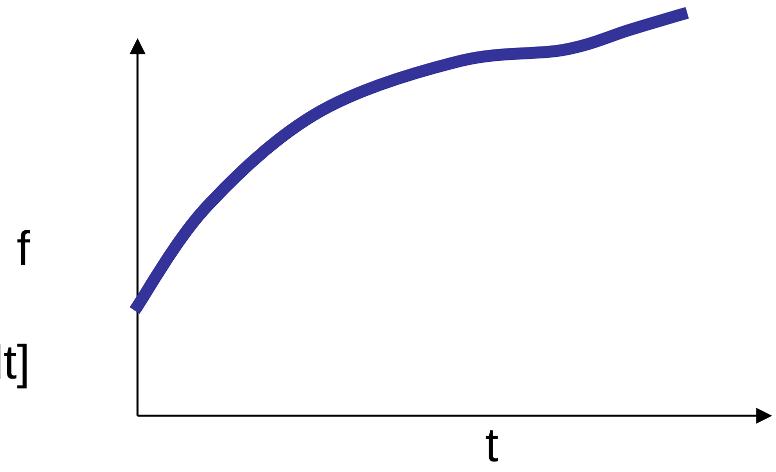
therefore

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

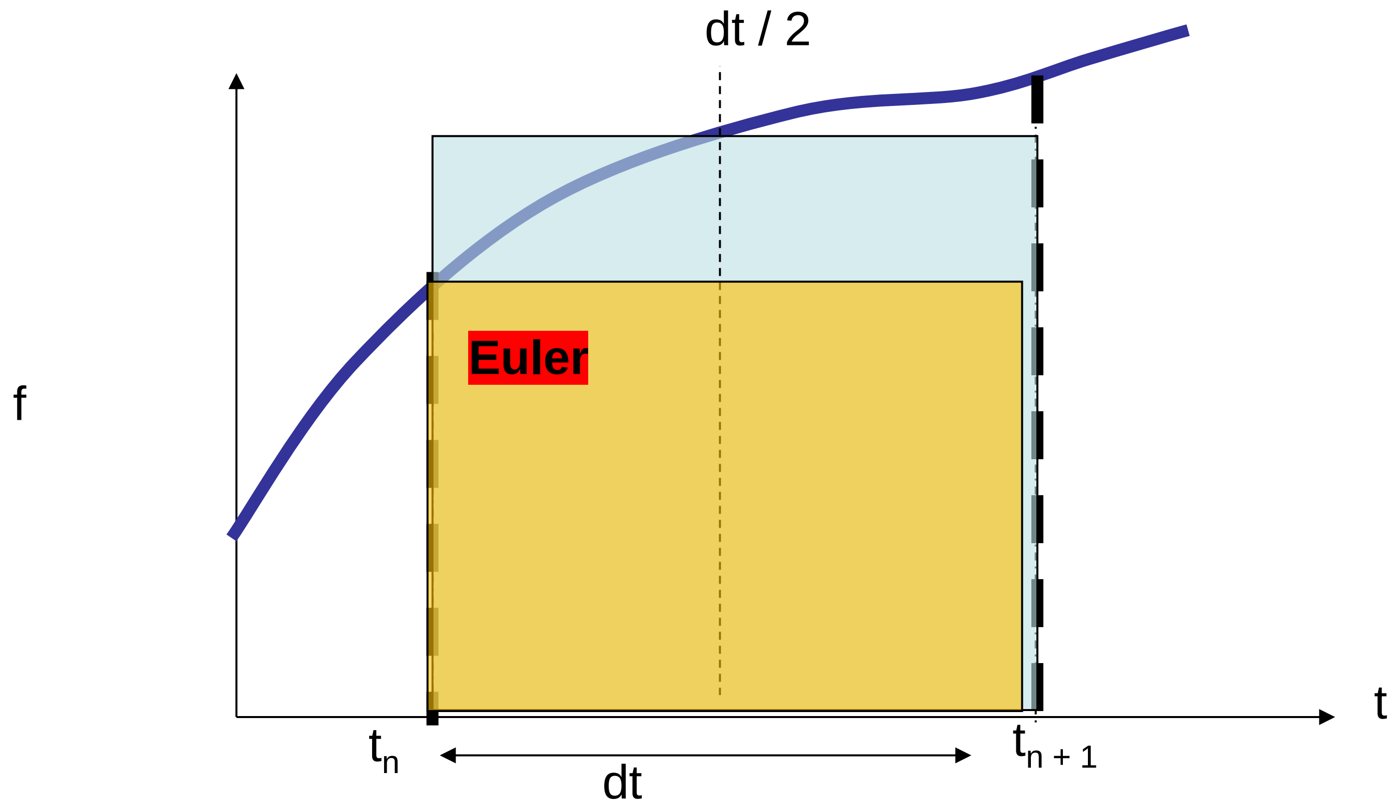
F is the area under the curve divided by dt

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 ~ (blue area) / dt

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

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***

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

Algorithm

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

$$X_{n+1} = X_n + \mathrm{dt} * \mathrm{f}\left(\mathrm{t} + \frac{\mathrm{dt}}{2}, \mathrm{k}_1\right)$$

Modifid 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

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

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

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

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

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

$$\Leftrightarrow$$

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

We restart at dt
But take the derivative at point dt/2

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

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

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

U (t = 0.1) = 0.740818 **Real result**

Euler: U (t = 0.1) = 0.7225

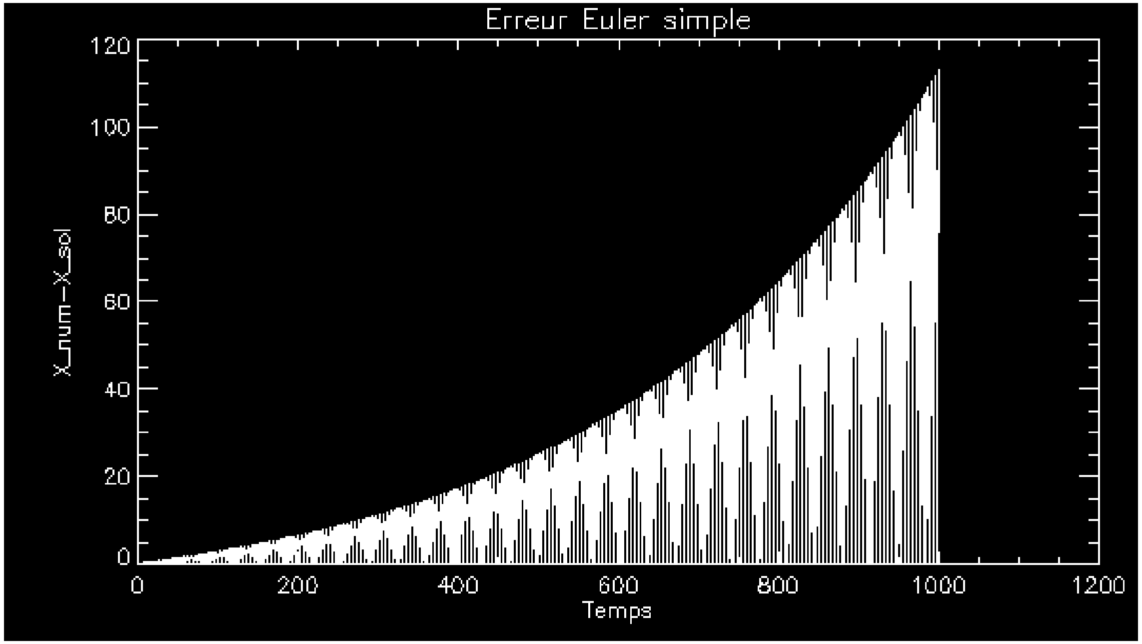
Euler modified
U (t = 0.1) = 0.740818

Example with same timestep

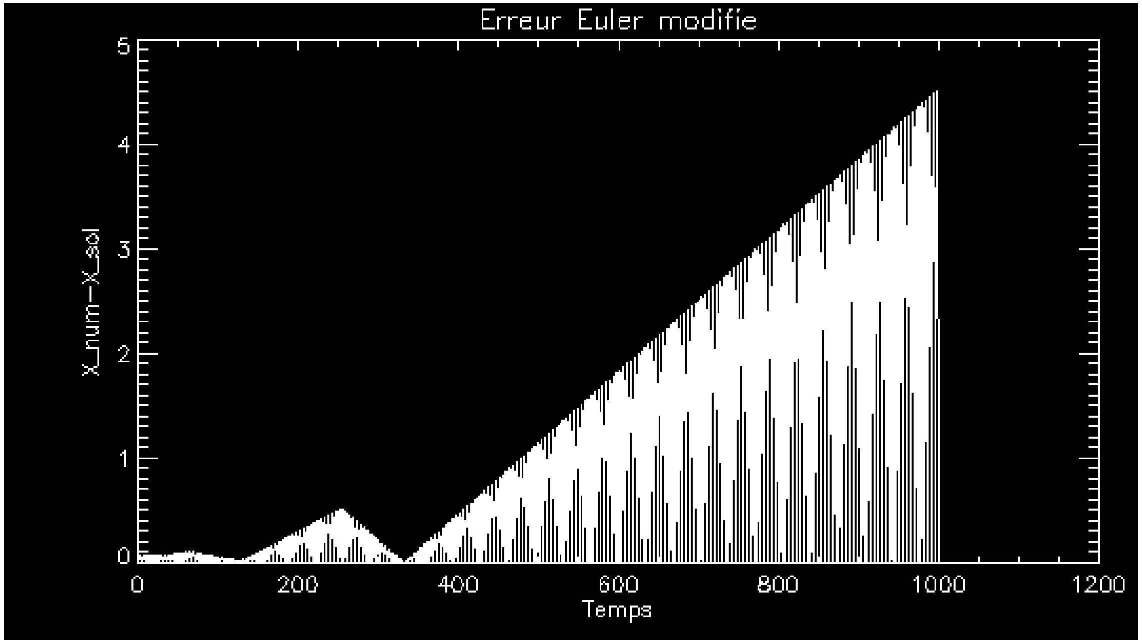
Example of the spring

Abs (X_vrai-X_approx)

Euler :
Dt = 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) *implicit*	$U_{n+1} = U_n + \frac{dt}{2} \cdot \left(f(t_n, U_n) + f(t_{n+1}, U_{n+1}) \right)$
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)	$U_{n+1} = U_n + dt \cdot \left(\frac{5}{12} f(t_{n+1}, U_{n+1}) + \frac{8}{12} f(t_n, U_n) - \frac{1}{12} f(t_{n-1}, U_{n-1}) \right)$

K1 = angular term of the straight line in 1st point
k2 = angular term in the second point

Runge Kutta (2)

$$\begin{cases} k_1 = dt \cdot f(t_n, U_n) \\ k_2 = dt \cdot f(t_n + dt, U_n + k1) \\ U_{n+1} = U_n + \frac{1}{2}(k_1 + k_2) \end{cases}$$

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)

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

Easy to implement, "Relatively" stable ..

Quite "slow" because 4 calls to the derivative.

Principle

RK4:
4 evaluations in
of the time

order 4

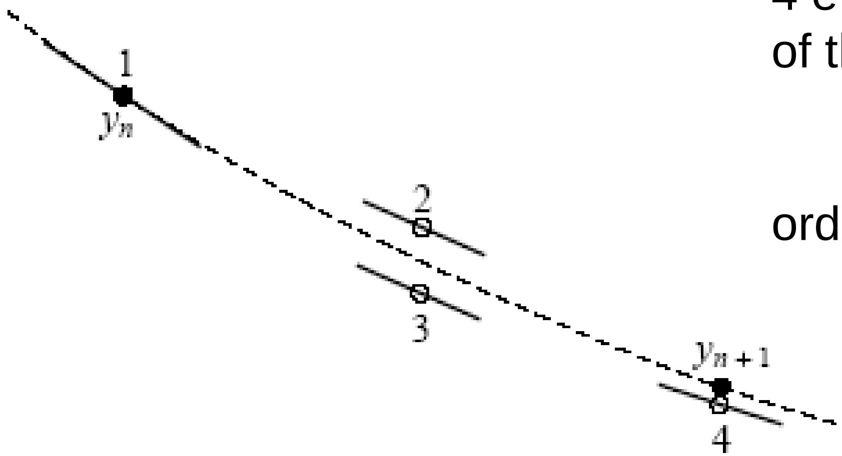


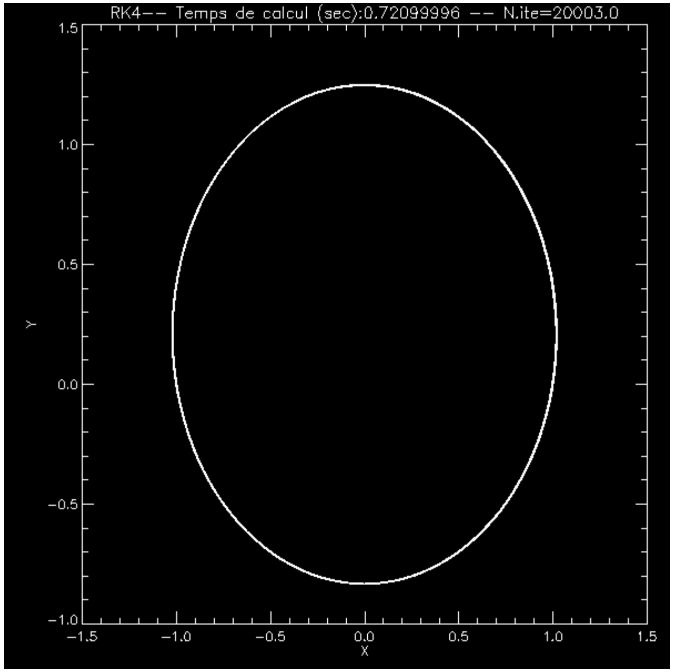
Figure 16.1.3. Fourth-order Runge-Kutta method. In each step the derivative is evaluated four times: once at the initial point, twice at trial midpoints, and once at a trial endpoint. From these derivatives the final function value (shown as a filled dot) is calculated. (See text for details.)

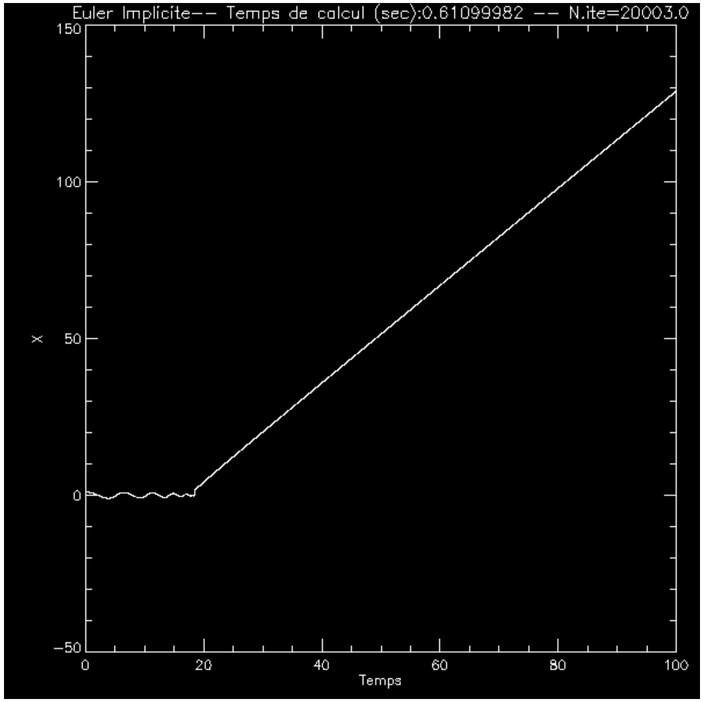
Consider a complicated system:

The motion of a planet around the Sun:

A = - GM u / r ^ 2

Dt = 0.1, Dynamic Time = 2PI

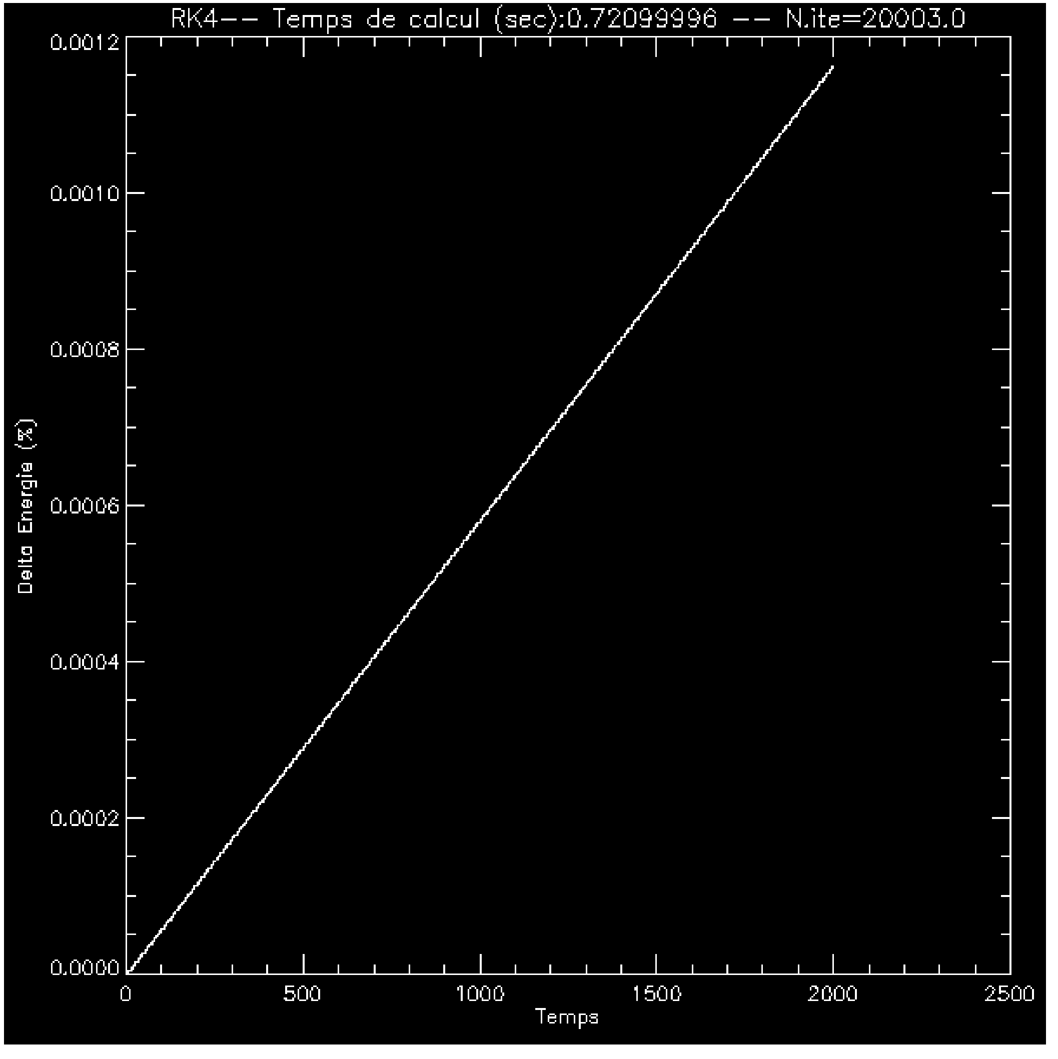




RK4
Everything goes well priori

Implicit Euler

A simple control parameter: Energy
The total energy should be kept: $E = 1/2\ mV^2 - Gm / r$

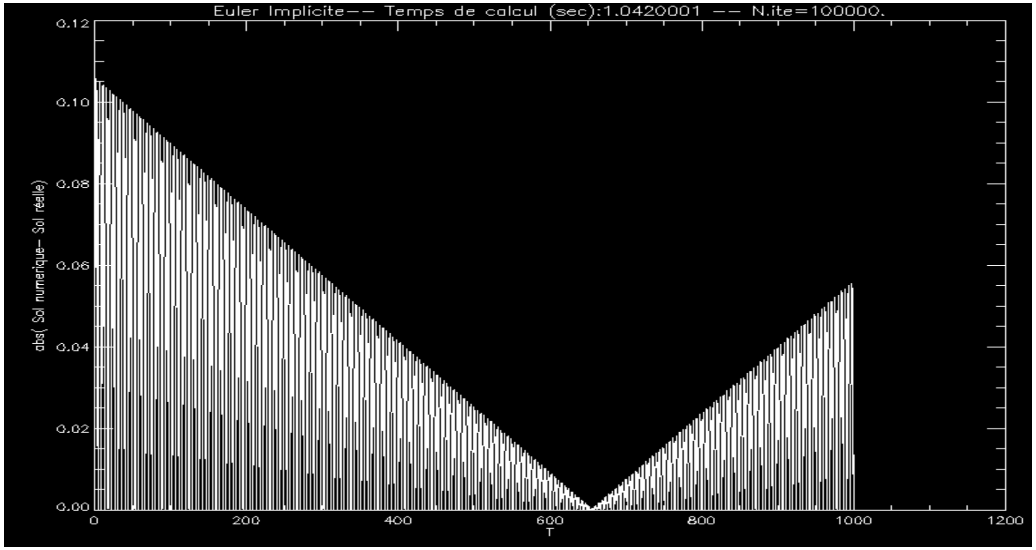


dt = 0.1
1 orbit = 2PI

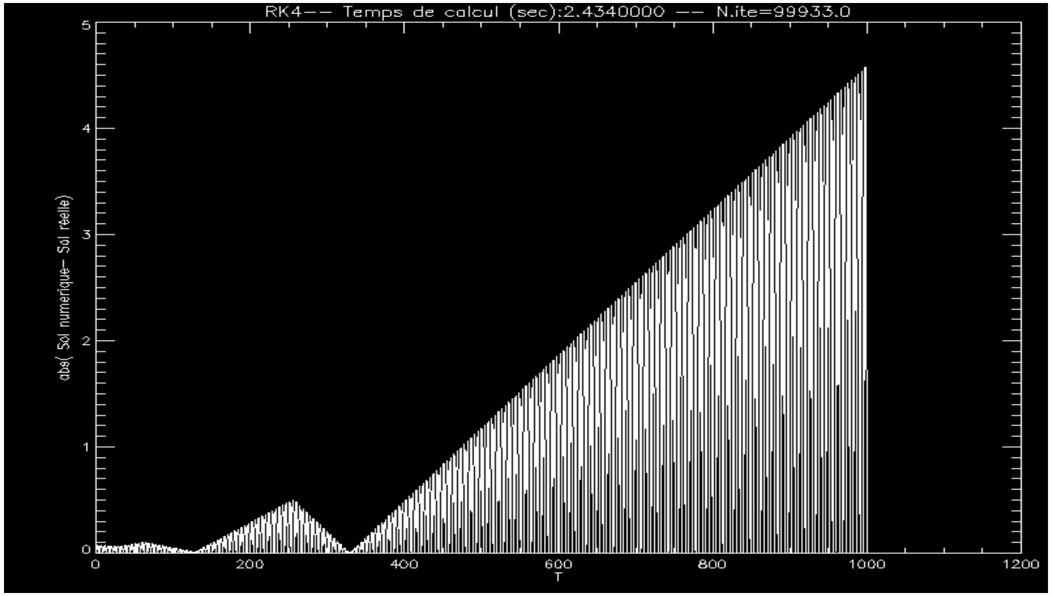
Energy artificially increased from 0.12% in 250 orbits (250 dynamic time)

⇒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** :

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

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 s calculated directly.

ANOTHER family of solvers are the

« **implicit solvers »** :

Example: Cranck Nicolson

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

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 straigthforward ! => 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-term

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:

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

This is solved by substitution

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

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

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

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

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.**

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

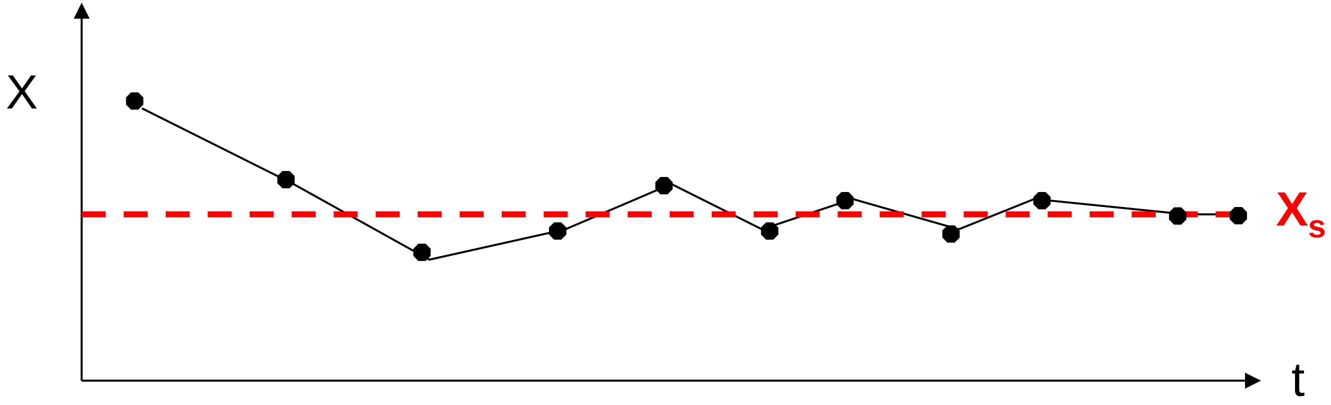
or

How fast are small errors amplified by the scheme ?

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

To quantify the **stability**, we=> implicit scheme.
assume that X **is adjacent to a stationary point Xs**

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



EXAMPLE spring uwing Euler exoplicit method (Intrinsically linear method) :

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

We use matrix notation for systems of ODEs

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

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

$$\boldsymbol{X_{n+1}} = \boldsymbol{A}\, \boldsymbol{X_n}$$

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 ?

$$X_{n+1} = F(X_n) = (1 + dt \cdot f) X_n = AX_n$$
$$e_{n+1} = X_{n+1} - X_n \qquad \Rightarrow$$
$$e_{n+1} = AX_n - AX_{n-1} \qquad \Rightarrow$$
$$e_{n+1} = A(X_n - X_{n-1}) \quad \Rightarrow$$
$$e_{n+1} = Ae_n$$

$$\boxed{1 + dt \cdot f}$$

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

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

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

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

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

Vectors in the eigen base A

In the eigen base

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

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

So for STABILITY (⇔ no amplification of error)
all the eigenvalues of the amplification matrix A must be <1 in absolute value.

Let V = MAX [abs (v1), abs (v2), abs (v3)]. V depends of the time dt

- If V <1 dt < dt_{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

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

Form: $X_{n+1} = AX_{not}$

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

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

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

Amplification matrix A

λ Eigen values : solution of characteristic polynomial
λ^2- trace(A)x λ + determinant = 0 =>

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

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

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

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 sprin equation is *unconditionally unstable (dt> 0)*

Example with the spring + implicit Euler method

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

Substitution

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

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

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

Matrix amplification

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

Form: $X_{n+1} = AX_{not}$

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

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

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

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

Calculating the amplification A matric (A such that $X_{n+1} = AX_{not)}$

The eigen values are:

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

$$= \frac{1 \pm idt\sqrt{\frac{k}{m}}}{\left(1+idt\sqrt{\frac{k}{m}}\right)\left(1-idt\sqrt{\frac{k}{m}}\right)}$$

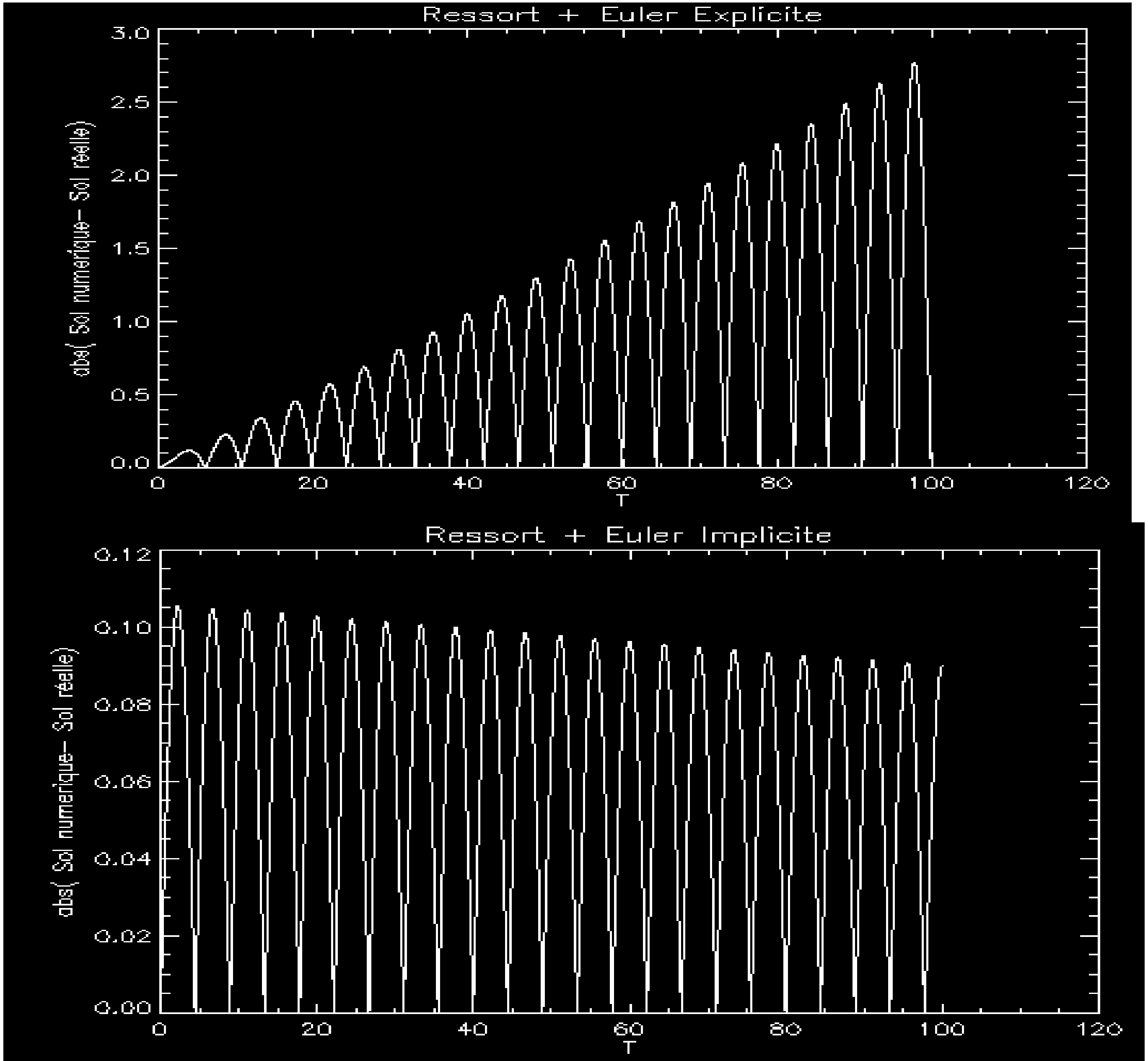
(Where we use identity : $a^2 + b^2 = (a + ib)(a - ib)$)

Eigen values norm are: $$\frac{1}{\left(1+dt\frac{k}{m}\right)^{1/2}}$$

They are always <1

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

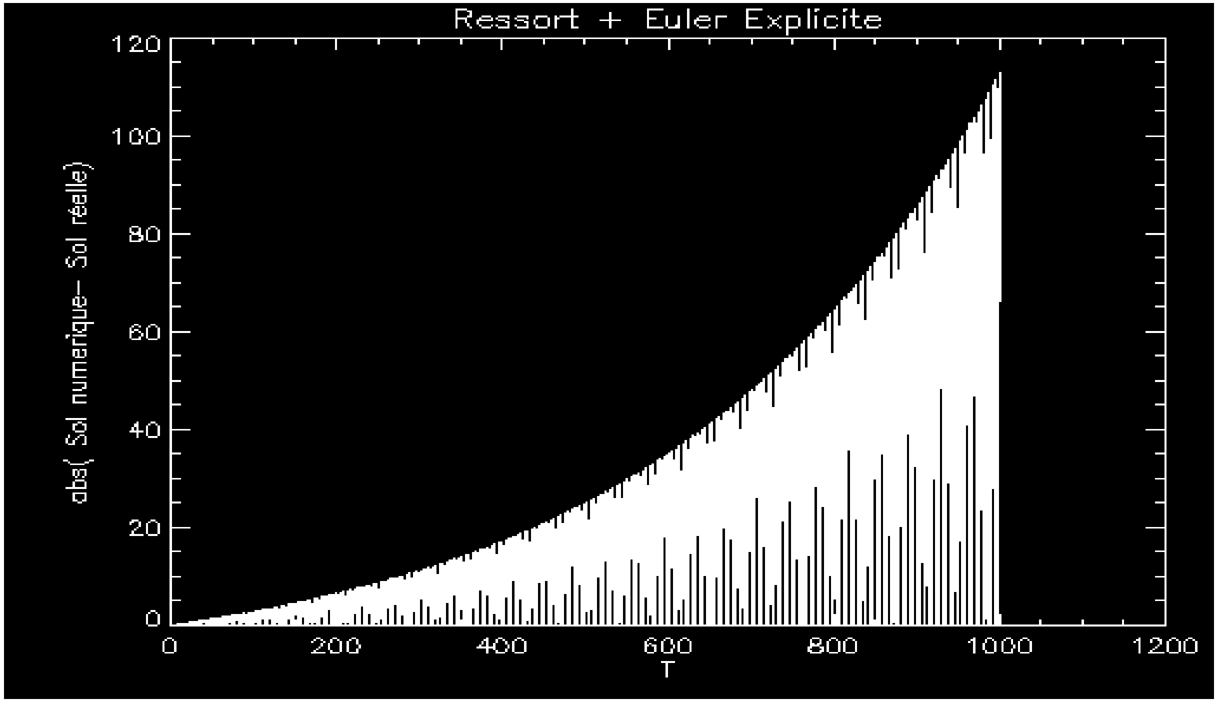
Compare Euler Explicit vs Implicit: Spring Case



error Explicit

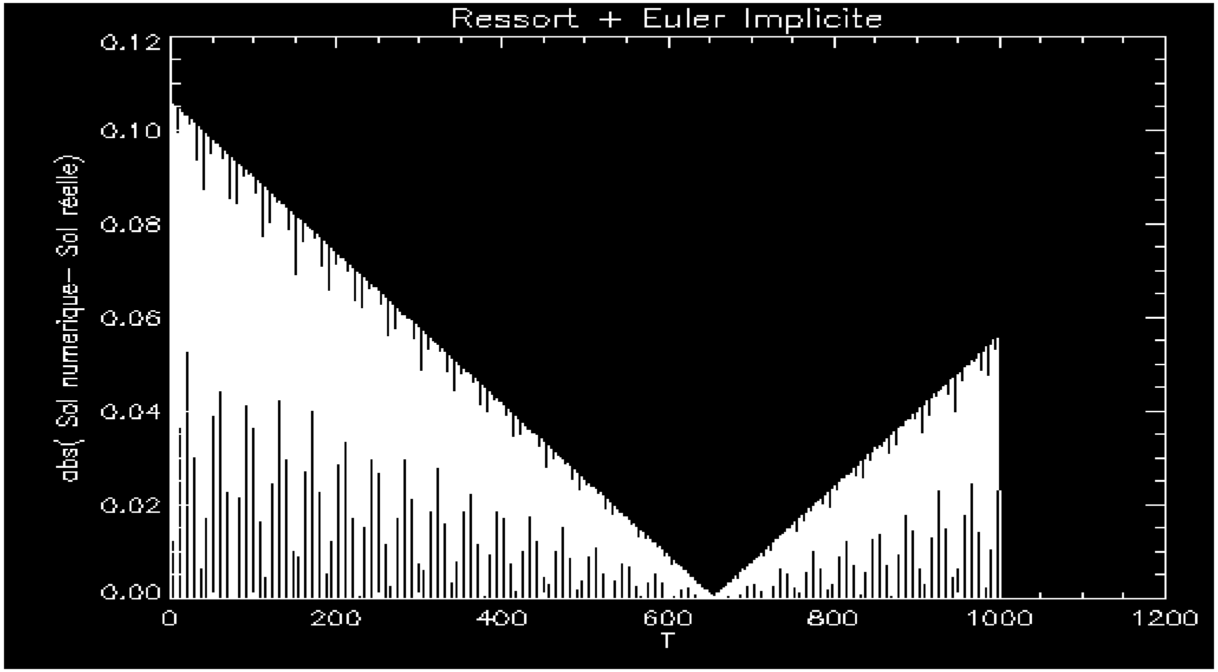
Dt = 0.01
10000 iterations

Implicit error



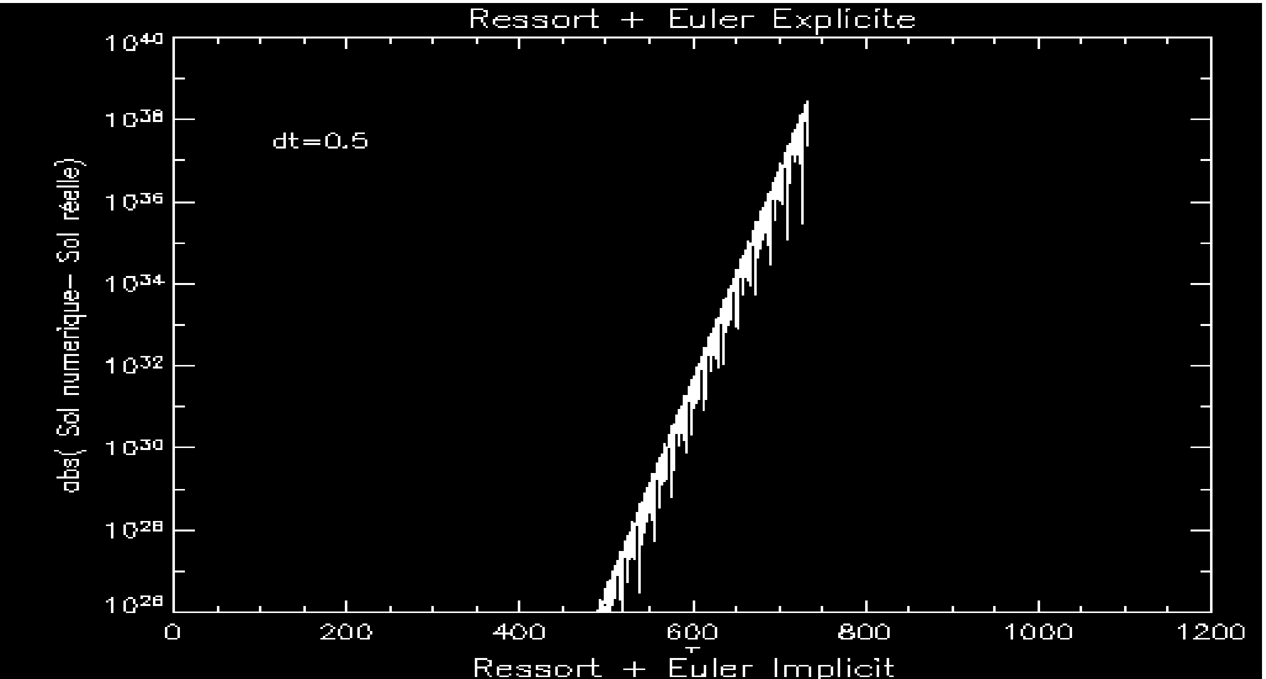
error Explicit

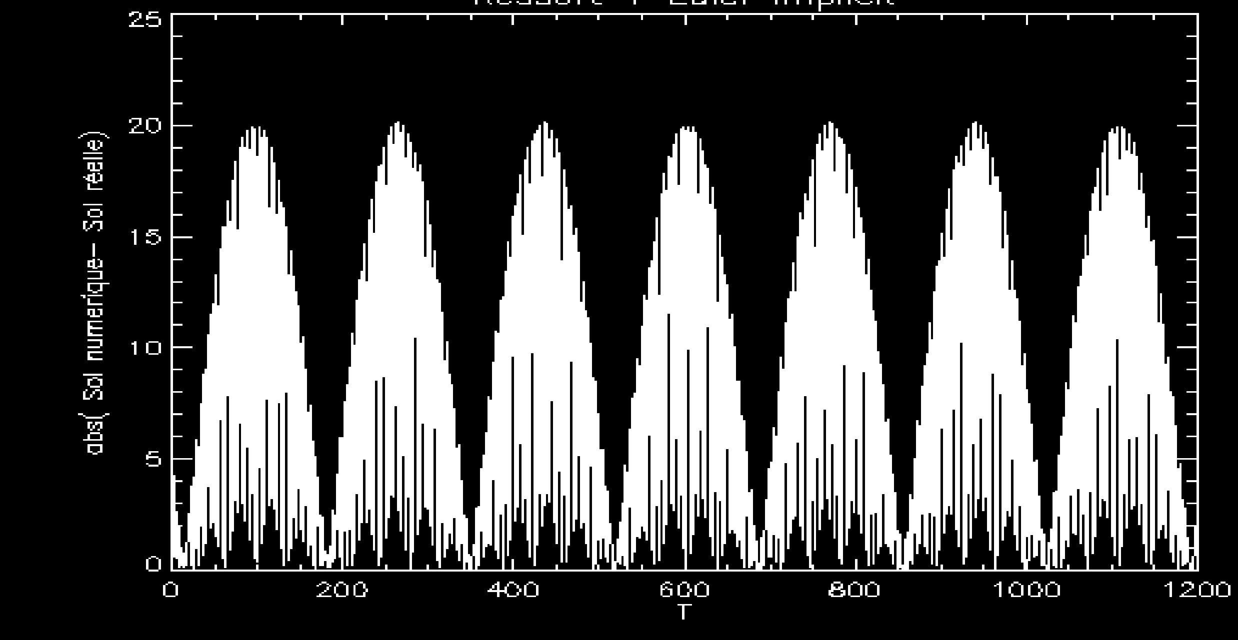
Dt = 0.01
100000 iterations



Implicit error

Large time step: 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, eventhough the solution might be wrong..

THEREFORE

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

What solver choose?

$\Rightarrow$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: dt << 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,

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

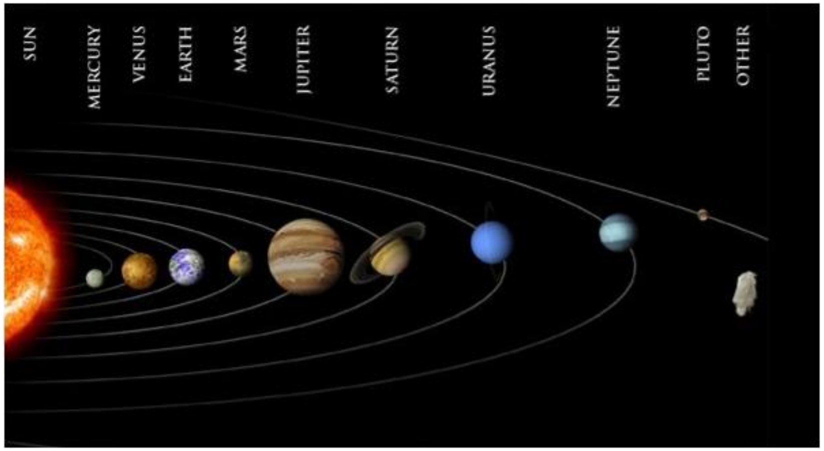
So we should take dt << T

In our example: m = 1, k = 1 => T = 6.28 seconds.

With dt = 0.01 s OK
With dt = 0.5s 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 << 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 false because different variables the systems will not be SYNCHRONIZED !!

For example:

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

Earth 2dt 4dT 6dT 8 dt 10 dt

Jupiter 3dt 6dT 9dT

Plunto: 5dT 10dT

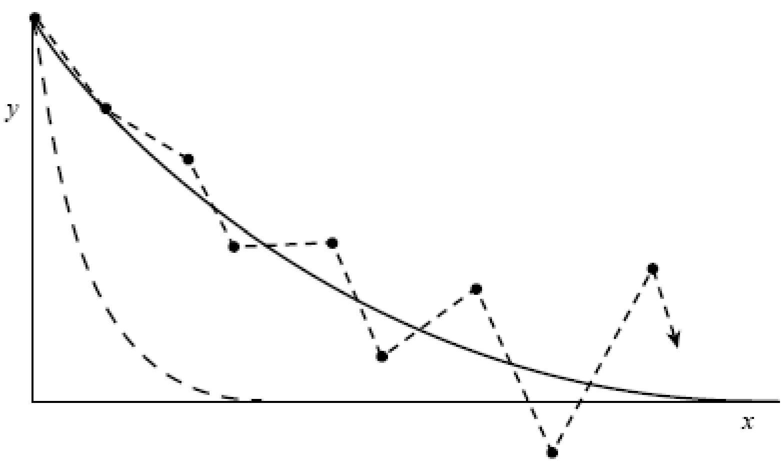
PB: For Mercury we need to know ALL the position planets to each value of dt ...

Another example: A system with two variables

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

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 ...
$\Rightarrow$Often this is wrong

In a system where ALL the variables are coupled together to others, dt << 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 (Xn)) = Solution, when dt -> 0
So when one is far from the solution (contrapositive) F highly dependent on no 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**

1era technical : Make 2 evaluations result, taking dt and DT / 2. (Double computation time). If both results are equal roughlyΔ_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 suing 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 ~ 1.4
40% slower

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

Felhberg studied RK5. It requires 6 calls to the derivative

The RK5 is as follows

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

$$k_2 = hf(x_n + a_2 h, y_n + b_{21} k_1)$$

$$\dots$$

$$k_6 = hf(x_n + a_6 h, y_n + b_{61} k_1 + \cdots + b_{65} k_5)$$

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

The result is accurate to order 5

But Fehlberg shows that other combination coefficients gives a result with 4th order accuraty (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 k1 to k6, we can have two different assessments of the result:

Y_{not} to about 5
Y^*_{not} to order 4

=> ABS (Y^*_{not}-Y_{not}) Is an estimate of the error in the order of 5

Coefficients table for RK5

Cash-Karp Parameters for Embedded Runga-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

$$
\begin{aligned}
f_0 &= f(x_i, y_i) \\
f_1 &= f\left(x_i + \alpha_1 h, y_i + h\beta_{10} f_0\right) \\
\cdots & \quad \cdots \\
f_k &= f\left(x_i + \alpha_k h, y_i + h(\beta_{k0} f_0 + \beta_{k1} f_1 + \cdots + \beta_{k,k-1} f_{k-1}\right)
\end{aligned}
$$

$$y_{i+1} = y_i + h\left(c_0 f_0 + c_1 f_1 + \cdots + c_k f_k\right)$$

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 $\quad\frac{1}{2}$	$\frac{1}{3}$
3	1	0 $\quad$ 0 $\quad$ 1	$\frac{1}{6}$

$$
\begin{aligned}
f_0 &= f(x_i, y_i), \\
f_1 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2} f_0), \\
f_2 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2} f_1), \\
f_3 &= f(x_i + h, y_i + h f_2) \\
y_{i+1} &= y_i + \frac{h}{6}\left(f_0 + 2f_1 + 2f_2 + f_3\right).
\end{aligned}
$$

Suppose we have two different estimates of the result,

X_n and X^*_n

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

We seek the new timestep so that Δ_0 / Δ =Precision required

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

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

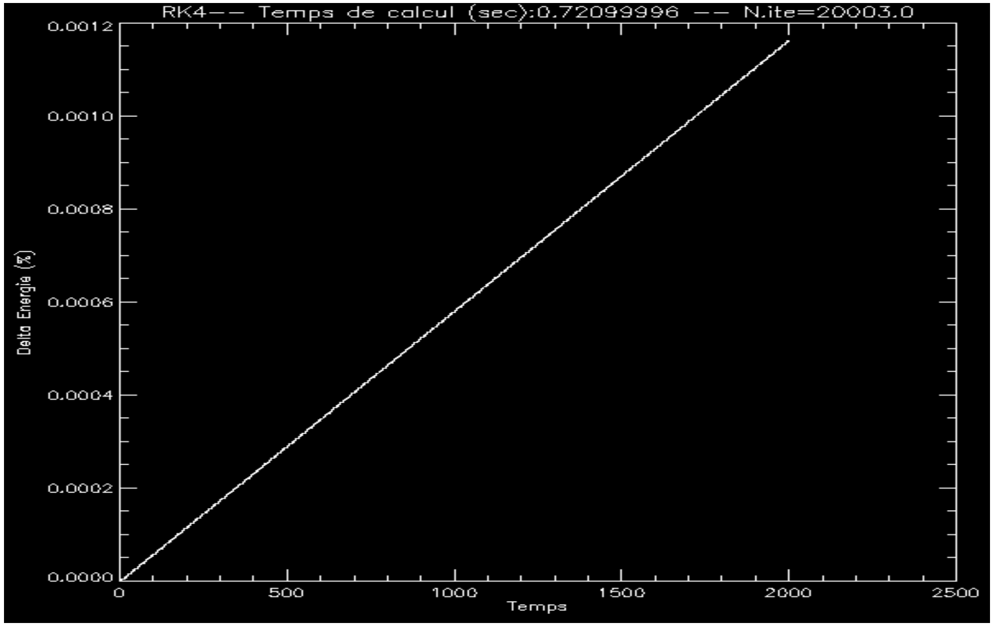
This serves two purposes:
1- If the error is too large time step decreases
2- If the error is too small time step increases => saves time!

A typical adaptive RK5. scheme

1. **Evaluates Y_n (5th order) and Y^*_n (4th order)**
2. **Calculate Δ=Abs(Y_n- Y^*_n)** (for exameple)= **the error control parameter**
3. **Calculate dt '**
4. **if dt >dt' : Reject solution**. **Replace dt by dt' and go back to 1**
   If dt <dt' :accept sol. Y_n is the next U_n, Replace dt by dt'. Go to next step
5. **Return to 1 (the next time step)**

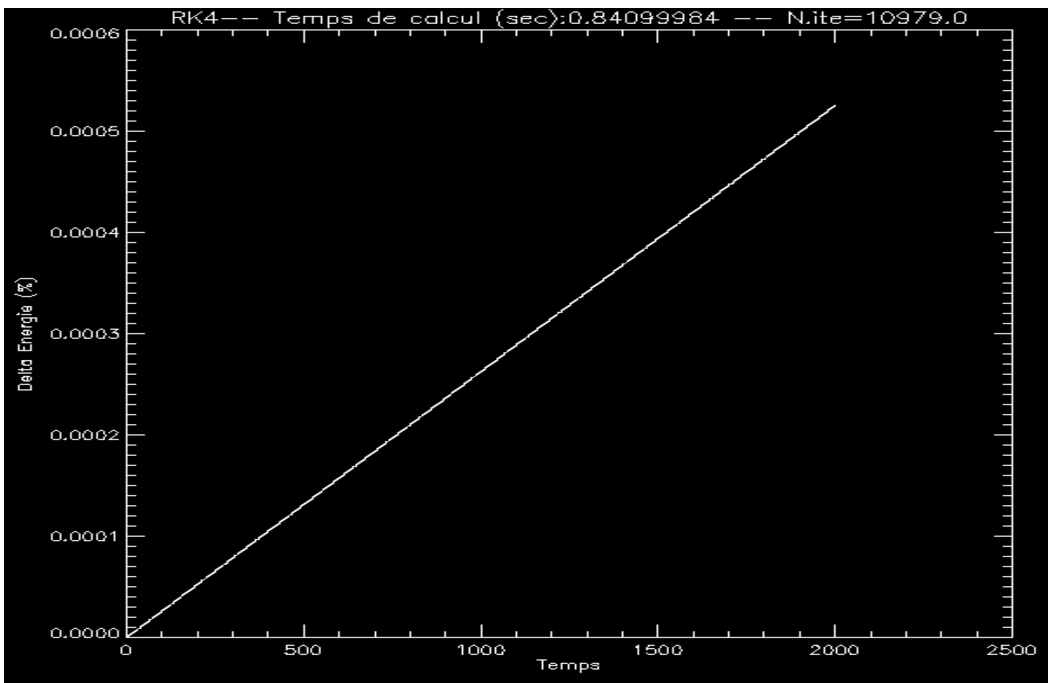
CAUTION: If you work with adaptive time step you must have an additional Control parameter (like energy) to make sure the calculation is globally valid

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



RK4 Energy

Dt = 0.2

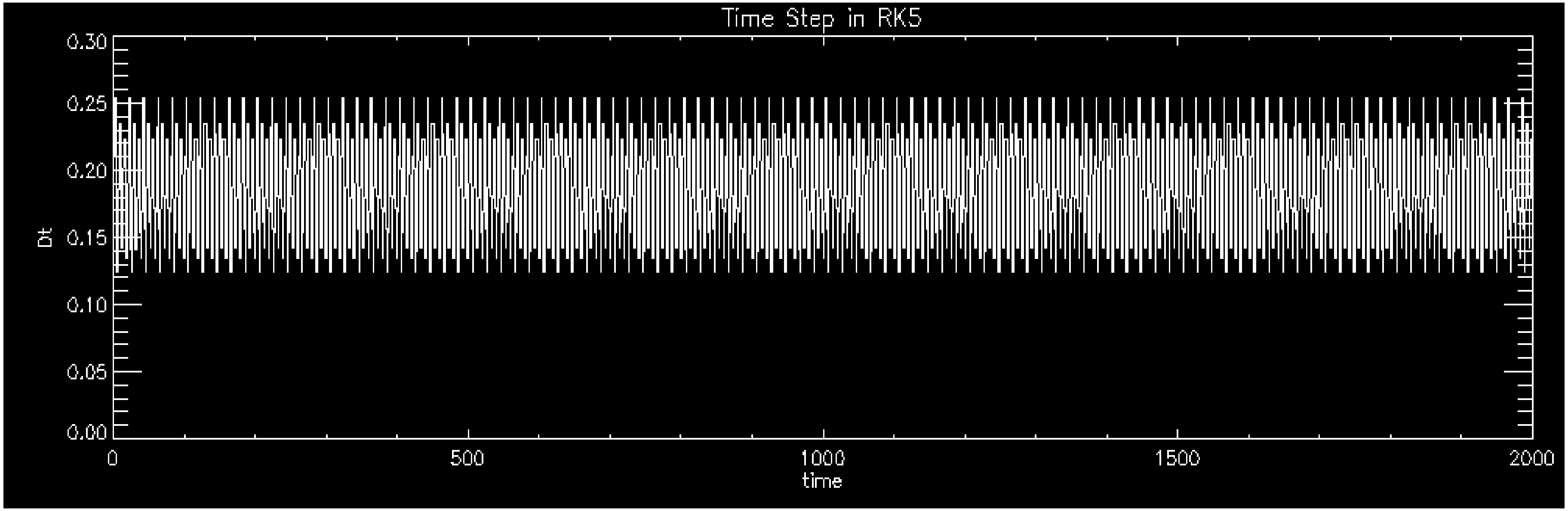


Adaptive Energy RK5

The error on E increases
2 times slower

With comparable computing time

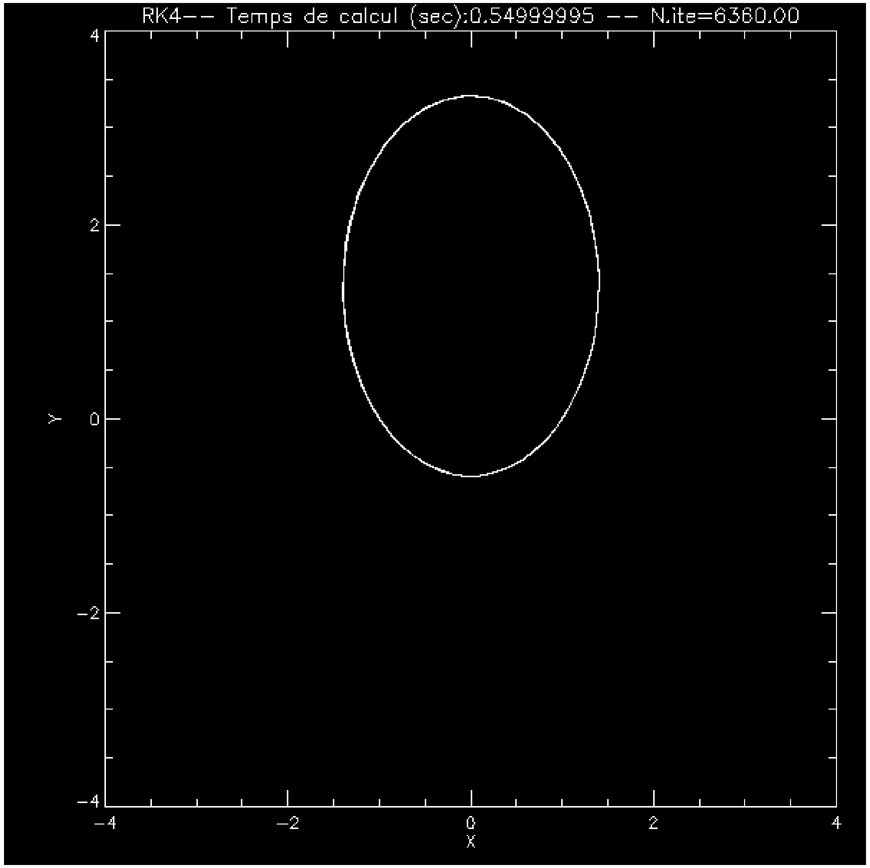
Here is the time step of the Adaptive RK5



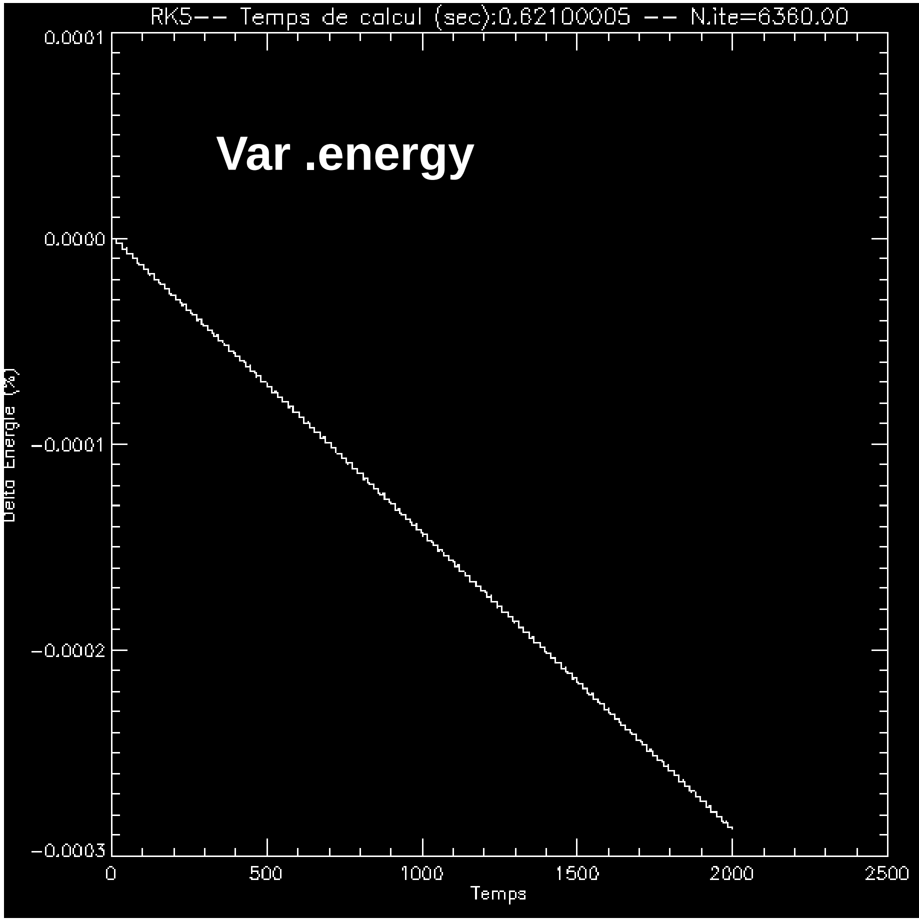
dt decreases as the planet accelerates (perihelion)

dt increases when the planet decelerates (Aphelion)

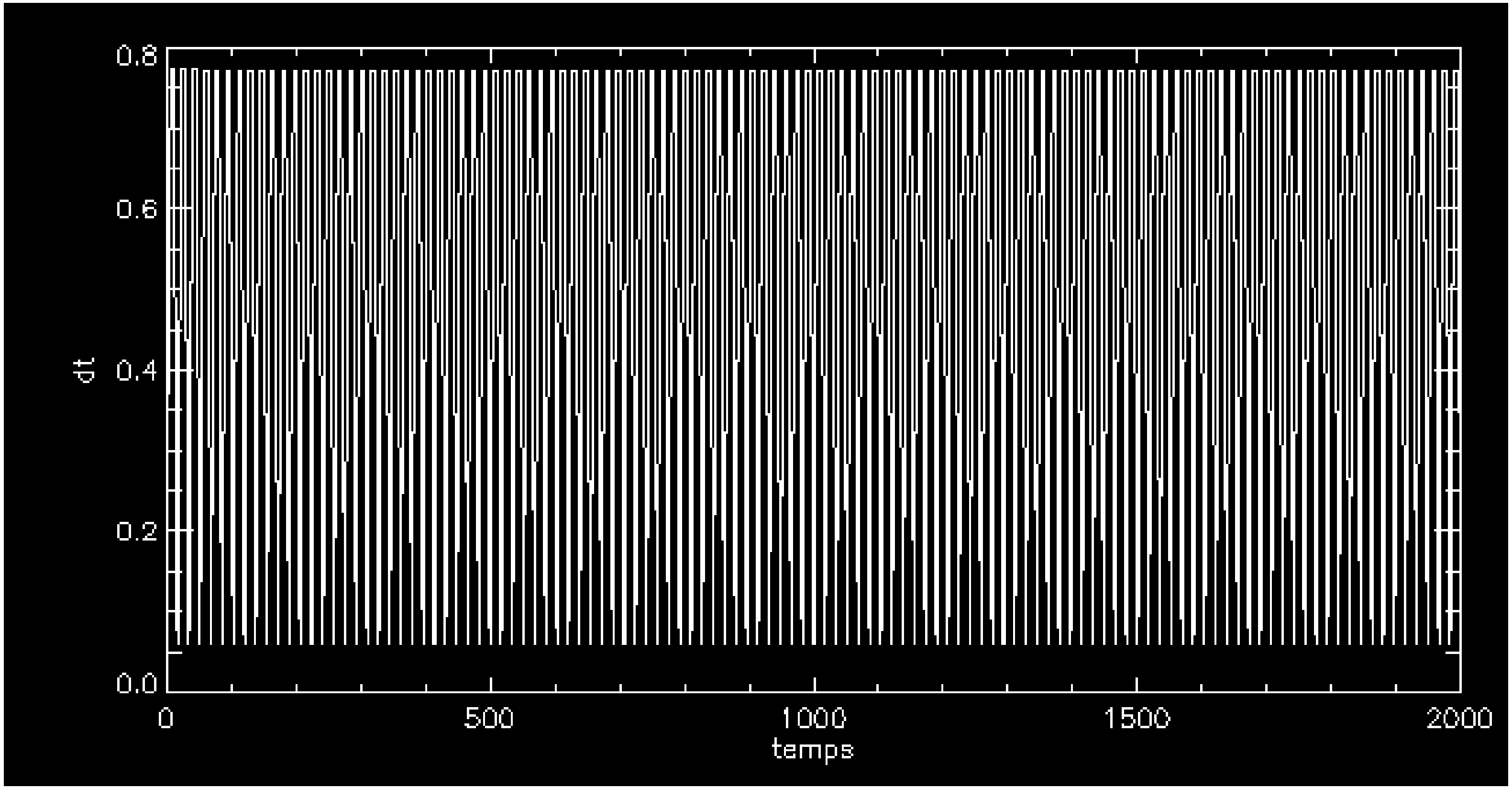
Consider a VERY elongated orbit (difficult to integrate)



adaptive RK5

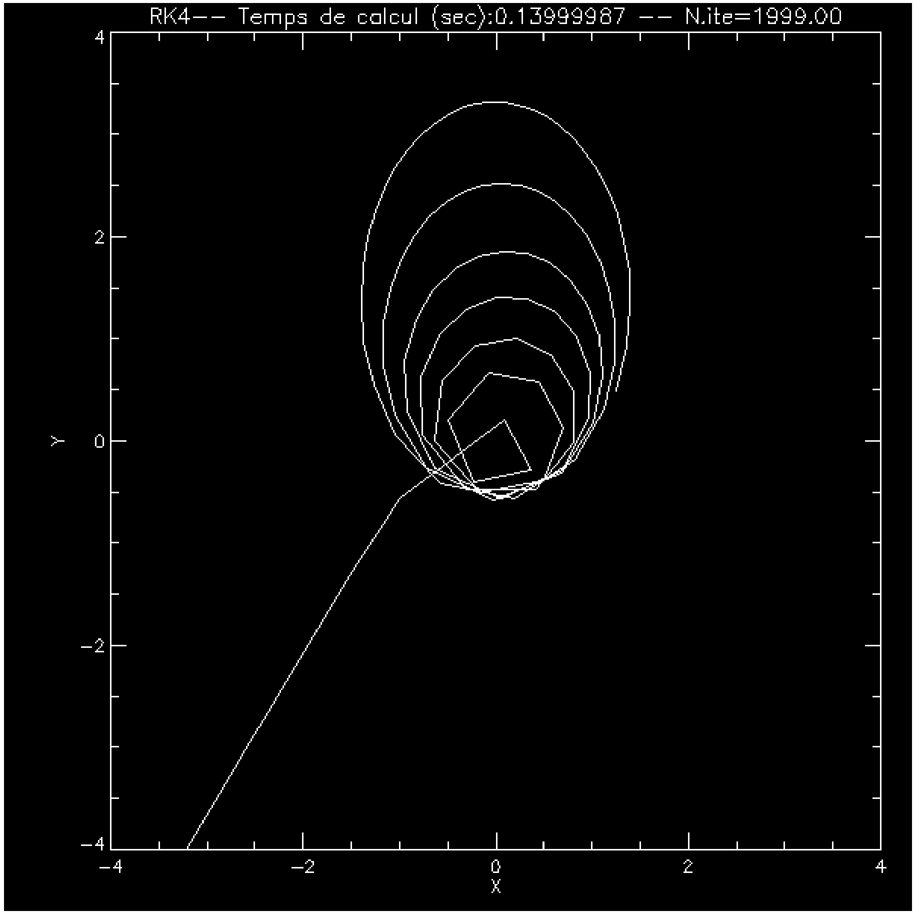


Evolution of adaptive RK5 time step:



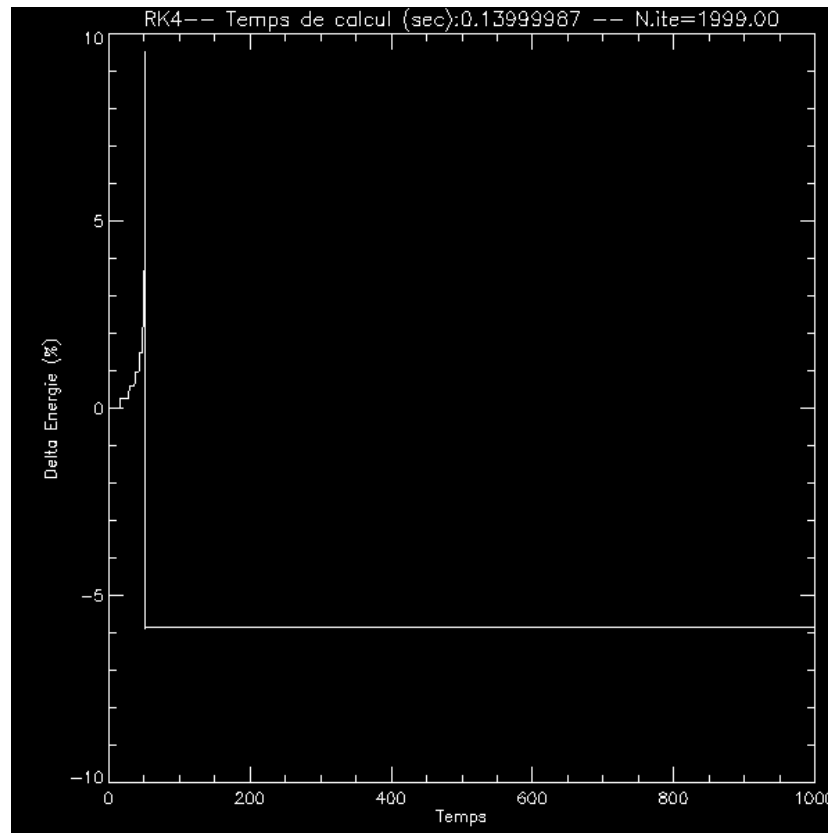
The time step adapts to the orbit.
Initial time step: 0.5

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