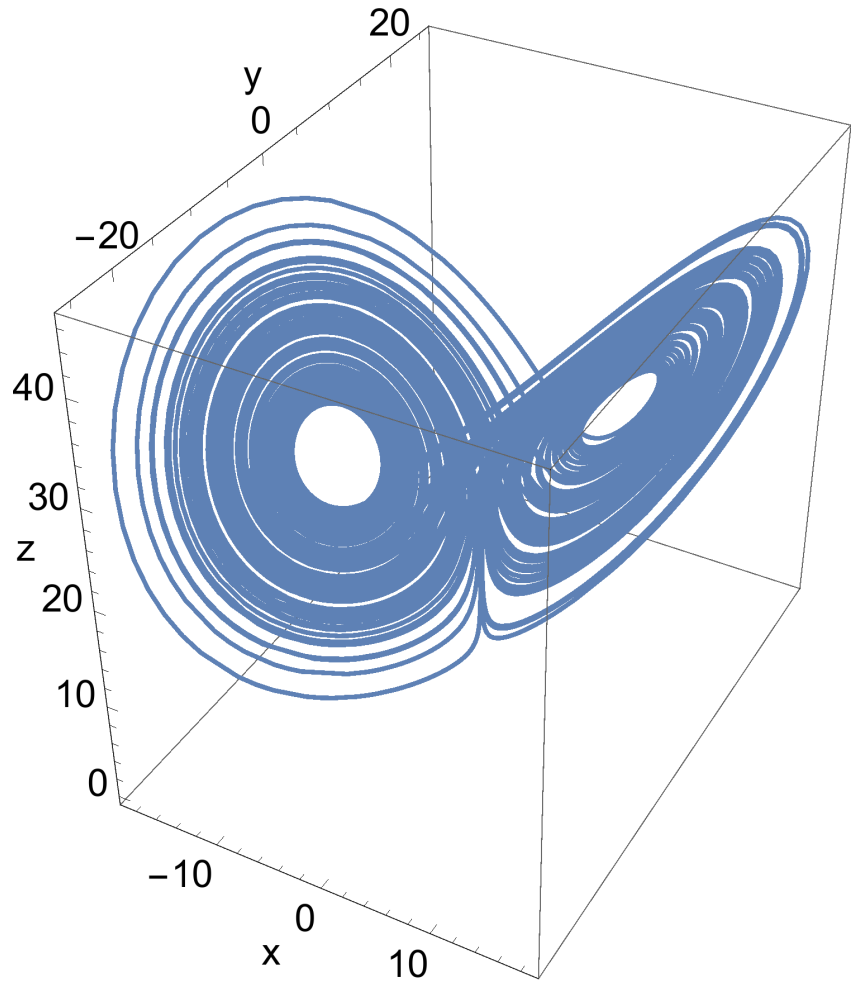
Programación y Métodos Numéricos

Ecuaciones diferenciales ordinarias

Profesor: B. Toledo

11 de mayo de 2023

Problema general



$$\dot{x} = \sigma(y - x),\ \dot{y} = x(\rho - z) - y,\ \dot{z} = x\,y - \beta\,z.$$

Problema general

En general estudiaremos el problema

$$\dot{\vec{\xi}} = \vec{F}(\vec{\xi}),$$

es decir, debemos transformar el problema diferencial que tengamos en uno de primer orden. Por ejemplo, para el oscilador armónico,

$$\ddot{x} + \omega^2 x = 0,$$

definimos $\dot{x} = y$, con lo que resulta el sistema,

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= -\omega^2 x, \end{aligned}$$

donde vemos que

$$\vec{\xi} = \begin{pmatrix} x \\ y \end{pmatrix}, \quad \vec{F}(\vec{\xi}) = \begin{pmatrix} y \\ -\omega^2 x \end{pmatrix}.$$

Problema general

Ya que $\xi(t)$ es desconocida, podemos encontrar distintas aproximaciones usando su expansión de Taylor,

$$\xi(t+\Delta t) = \xi(t) + \dot{\xi}(t)\Delta t + \frac{1}{2}\ddot{\xi}(t)(\Delta t)^2 + \dots$$

si suponemos que el resto de Taylor converge a cero,

$$\xi(t+\Delta t) = \sum_{k=0}^{\infty} \frac{1}{k!}\xi^{(k)}(t)(\Delta t)^k.$$

Si ahora suponemos que el sistema es lineal, resulta

$$\dot{\vec{\xi}} = \vec{F}(\vec{\xi}),\ \ddot{\vec{\xi}} = DF(\vec{\xi})\dot{\vec{\xi}} = DF(\vec{\xi})\vec{F}(\vec{\xi}), \dots$$

y sustituir en la serie.

Problema general

Tomemos como ejemplo el oscilador armónico,

$$\ddot{x} + \omega^2 x = 0,$$

resulta,

$$x^{(3)} + \omega^2 \dot{x} = 0, \quad x^{(4)} + \omega^2 \ddot{x} = 0, \ldots, x^{(n)} + \omega^2 x^{(n-2)} = 0.$$

Luego,

$$\begin{aligned} x(t+\Delta t) &= x(t) + \dot{x}(t)\Delta t - \frac{1}{2!}\omega^2 x(t)\Delta t^2 - \frac{1}{3!}\omega^2 \dot{x}(t)\Delta t^3 + \\ &\quad + \frac{1}{4!}\omega^4 x(t)\Delta t^4 + \frac{1}{5!}\omega^4 \dot{x}(t)\Delta t^5 + \ldots \end{aligned}$$

Problema general

$$
\begin{aligned}
x(t+\Delta t) &= x(t) + \dot{x}(t)\Delta t - \frac{1}{2!}\omega^2 x(t)\Delta t^2 - \frac{1}{3!}\omega^2 \dot{x}(t)\Delta t^3 + \\
&\quad + \frac{1}{4!}\omega^4 x(t)\Delta t^4 + \frac{1}{5!}\omega^4 \dot{x}(t)\Delta t^5 + \ldots \\
&= x(t)\left(1 - \frac{1}{2!}\omega^2\Delta t^2 + \frac{1}{4!}\omega^4\Delta t^4 + \ldots\right) + \\
&\quad \dot{x}(t)\left(\Delta t - \frac{1}{3!}\omega^2\Delta t^3 + \frac{1}{5!}\omega^4\Delta t^5 + \ldots\right) \\
&= x(t)\left(1 - \frac{1}{2!}\omega^2\Delta t^2 + \frac{1}{4!}\omega^4\Delta t^4 + \ldots\right) + \\
&\quad \frac{\dot{x}(t)}{\omega}\left(\omega\Delta t - \frac{1}{3!}\omega^3\Delta t^3 + \frac{1}{5!}\omega^5\Delta t^5 + \ldots\right) \\
&= x(t)\cos(\omega\Delta t) + \frac{\dot{x}(t)}{\omega}\sin(\omega\Delta t).
\end{aligned}
$$

Problema general

Ahora tomamos $t = 0$ y $\Delta t = t$, resultando,

$$\begin{aligned} x(t) &= x(0)\cos(\omega\, t) + \frac{\dot{x}(0)}{\omega}\sin(\omega\, t), \\ &= A\,\cos(\omega\, t) + B\,\sin(\omega\, t), \end{aligned}$$

En principio, para problemas *lineales*, la suma en Taylor se puede hacer, pero no siempre es fácil!

Algoritmos

Euler: Sin perdida de generalidad podemos considerar un problema de una dimensión espacial, del tipo

$$\ddot{x} = f(x),$$

expandiendo en Taylor,

$$\begin{aligned} x(t+\Delta t) &= x(t) + \dot{x}(t)\Delta t + \ddot{x}(t)\frac{\Delta t^2}{2} + \ldots, \\ &= x(t) + \dot{x}(t)\Delta t + f\left(x(t)\right)\frac{\Delta t^2}{2} + \ldots, \end{aligned}$$

derivando esta última expresión resulta,

$$v(t+\Delta t) = v(t) + \dot{v}(t)\Delta t + f'\left(x(t)\right)v(t)\frac{\Delta t^2}{2} + \ldots,$$

Algoritmos: Euler

y usaremos la notación $x(t_n) = x_n$, $\dot{x}(t_n) = v_n$, y $\ddot{x}(t_n) = a_n$. Con esto, tenemos

$$x_{n+1} = x_n + v_n\,\Delta t + \frac{a_n}{2}\,\Delta t^2 + \ldots$$

$$v_{n+1} = v_n + a_n\,\Delta t + f'\left(x_n\right) v_n\,\frac{\Delta t^2}{2} + \ldots,$$

si nos restringimos a variaciones lineales en Δt, resulta el algoritmo buscado,

$$\boxed{\begin{aligned} x_{n+1} &= x_n + v_n\,\Delta t \\ v_{n+1} &= v_n + a_n\,\Delta t \end{aligned}}$$

Para observar su estabilidad, consideremos el caso simple $f(x) = \lambda\,x$.

Algoritmos: Euler

Entonces lo podemos escribir en forma matricial,

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ \lambda \Delta t & 1 \end{pmatrix} \begin{pmatrix} x_n \\ v_n \end{pmatrix}$$

Es fácil ver que,

$$\begin{pmatrix} x_n \\ v_n \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ \lambda \Delta t & 1 \end{pmatrix}^n \begin{pmatrix} x_0 \\ v_0 \end{pmatrix},$$

además, sabemos que,

$$A^n = (J^{-1}\, D\, J)(J^{-1}\, D\, J) \ldots (J^{-1}\, D\, J) = J^{-1}\, D^n\, J,$$

donde D es una matriz diagonal y J la matriz de cambio de base.

Algoritmos: Euler

Los autovalores son,

$$\sigma_+ = 1 + \lambda^{1/2}\Delta t, \quad \sigma_- = 1 - \lambda^{1/2}\Delta t,$$

entonces,

$$D^n = \begin{pmatrix} \sigma_+^n & 0 \\ 0 & \sigma_-^n \end{pmatrix},$$

si $\Delta t = T/n$, y tomamos $n \to \infty$,

$$\sigma_+^n = \left(1 + \lambda^{1/2}\frac{T}{n}\right)^n \xrightarrow[n\to\infty]{} \Sigma_+ = e^{\lambda^{1/2}T}, \quad \sigma_-^n \xrightarrow[n\to\infty]{} \Sigma_- = e^{-\lambda^{1/2}T},$$

si $\Re(\lambda^{1/2}) \neq 0$, entonces al menos en un caso, $|\Sigma| \to \infty$, cuando $T \to \infty$. Luego el algoritmo es *incondicionalmente inestable*, incluso para $\lambda < 0$, porque no podemos tomar $\Delta t \to 0$.

Algoritmos: Euler

Para Δt finito y $\lambda < 0$,

$$\sigma_{\pm} = 1 \pm \mathbf{i}|\lambda|^{1/2}\Delta t \Rightarrow |\sigma_{\pm}|^2 = 1 + \lambda^2 \Delta t^2,$$

por lo que el vector solución diverge.

Algoritmos: Euler-Cromer

Es el siguiente,

$$\boxed{\begin{aligned} v_{n+1} &= v_n + a_n\,\Delta t \\ x_{n+1} &= x_n + v_{n+1}\,\Delta t \end{aligned}}$$

Nuevamente estudiamos su estabilidad para $a_n = f_n = f(x_n) = \lambda x_n$, resulta

$$\begin{aligned} v_{n+1} &= v_n + \lambda x_n\,\Delta t \\ x_{n+1} &= x_n + (v_n + \lambda\, x_n\,\Delta t)\,\Delta t = (1 + \lambda\Delta t^2)x_n + v_n\Delta t \end{aligned}$$

escrito matricialmente es,

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} 1 + \lambda\Delta t^2 & \Delta t \\ \lambda\Delta t & 1 \end{pmatrix} \begin{pmatrix} x_n \\ v_n \end{pmatrix}$$

Algoritmos: Euler-Cromer

Sus autovalores son,

$$\sigma_{\pm} = 1 + \frac{\lambda}{2}\Delta t^2 \pm \frac{\Delta t}{2}\sqrt{\lambda(4 + \lambda\Delta t^2)}.$$

Analíticamente, el caso estable se da cuando $\lambda < 0$, los autovalores ahora son

$$\sigma_{\pm} = 1 - \frac{|\lambda|}{2}\Delta t^2 \pm \frac{\mathbf{i}\Delta t}{2}\sqrt{|\lambda|(4 - |\lambda|\Delta t^2)},$$

y para ambos se cumple que $|\sigma_{\pm}| = 1$. Esto implica que el algoritmo es *incondicionalmente estable*.

Algoritmos: Verlet

Comencemos nuevamente con,

$$x_{n+1} = x_n + v_n\,\Delta t + \frac{a_n}{2}\,\Delta t^2 + \ldots$$

y ahora en reversa,

$$x_{n-1} = x_n - v_n\,\Delta t + \frac{a_n}{2}\,\Delta t^2 + \ldots$$

sumando estas dos últimas expresiones, resulta

$$x_{n+1} + x_{n-1} = 2x_n + a_n\,\Delta t^2 + \ldots$$

es decir

$$\boxed{x_{n+1} = 2x_n - x_{n-1} + a_n\,\Delta t^2 + \ldots}$$

restándolas ahora da

$$x_{n+1} - x_{n-1} = 2v_n\,\Delta t + \ldots$$

Algoritmos: Verlet

$$\boxed{\begin{aligned} x_{n+1} &= 2x_n - x_{n-1} + a_n\,\Delta t^2 + \ldots \\ v_n &= \frac{x_{n+1} - x_{n-1}}{2\,\Delta t} + \ldots \end{aligned}}$$

Un problema con este algoritmo es que no se puede iniciar dando un solo punto, toca hacer más álgebra!
Tomemos,

$$x_{n-1} = x_{n+1} - 2v_n\Delta t,$$

entonces,

$$x_{n+1} = 2x_n - x_{n-1} + a_n\,\Delta t^2 = 2x_n - (x_{n+1} - 2v_n\Delta t) + a_n\,\Delta t^2$$

de donde resulta

$$\boxed{x_{n+1} = x_n + v_n\Delta t + a_n\,\frac{\Delta t^2}{2}}$$

Algoritmos: Verlet

Tomemos ahora,

$$x_{n+2} = 2x_{n+1} - x_n + a_{n+1}\,\Delta t^2$$
$$v_{n+1} = \frac{x_{n+2} - x_n}{2\,\Delta t}$$

resulta,

$$v_{n+1} = \frac{x_{n+2} - x_n}{2\,\Delta t} = \frac{(2x_{n+1} - x_n + a_{n+1}\,\Delta t^2) - x_n}{2\,\Delta t}$$

simplificando queda,

$$v_{n+1} = \frac{x_{n+1} - x_n}{\Delta t} + \frac{1}{2}a_{n+1}\,\Delta t = \frac{v_n \Delta t + a_n\,\frac{\Delta t^2}{2}}{\Delta t} + \frac{1}{2}a_{n+1}\,\Delta t$$

Algoritmos: Verlet

entonces,

$$v_{n+1} = v_n + \frac{1}{2}(a_n + a_{n+1})\,\Delta t,$$

y resulta el esquema,

$$\boxed{\begin{aligned} x_{n+1} &= x_n + v_n \Delta t + a_n \frac{\Delta t^2}{2} \\ v_{n+1} &= v_n + \frac{1}{2}(a_n + a_{n+1})\,\Delta t, \end{aligned}}$$

que comienza dando un solo punto y se conoce como *velocity-Verlet*.

Su análisis de estabilidad se deja como ejercicio.

Algoritmos: Runge-Kutta

Derivaremos el método de segundo orden. Tomemos el sistema,

$$\vec{Y}' = \vec{F}(\vec{Y}, t),$$

y expandamos en serie de Taylor,

$$\vec{Y}(t + \Delta t) = \vec{Y}(t) + \vec{Y}'(t)\Delta t + \vec{Y}''(t)\frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3),$$

notemos que

$$\vec{Y}'(t) = \vec{F}(\vec{Y}, t),$$
$$\vec{Y}''(t) = \vec{F}_{,t}(\vec{Y}, t) + \vec{F}_{,\vec{Y}}(\vec{Y}, t)\vec{Y}'(t)$$

Algoritmos: Runge-Kutta

sustituyendo resulta,

$$\vec{Y}(t+\Delta t) = \vec{Y}(t) + \vec{F}(\vec{Y},t)\Delta t + \left(\vec{F}_{,t}(\vec{Y},t) + \vec{F}_{,\vec{Y}}(\vec{Y},t)\vec{Y}'(t)\right)\frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3),$$

que se puede reescribir así,

$$\vec{Y}(t+\Delta t) = \vec{Y}(t) + \vec{F}(\vec{Y},t)\frac{\Delta t}{2} + \left(\vec{F}(\vec{Y},t) + \vec{F}_{,t}(\vec{Y},t)\Delta t + \vec{F}_{,\vec{Y}}(\vec{Y},t)\vec{Y}'(t)\Delta t\right)\frac{\Delta t}{2} + \mathcal{O}(\Delta t^3),$$

Algoritmos: Runge-Kutta

que se puede reescribir así,

$$\vec{Y}(t+\Delta t) = \vec{Y}(t) + \vec{F}(\vec{Y},t)\frac{\Delta t}{2} + \\ + \left(\vec{F}(\vec{Y},t) + \vec{F}_{,t}(\vec{Y},t)\Delta t + \vec{F}_{,\vec{Y}}(\vec{Y},t)\vec{Y}'(t)\Delta t\right)\frac{\Delta t}{2} \\ + \mathcal{O}(\Delta t^3),$$

Ahora, notemos que

$$\vec{F}(\vec{Y} + \vec{F}\Delta t, t+\Delta t) = \vec{F}(\vec{Y},t) + \vec{F}_{,t}(\vec{Y},t)\Delta t + \\ + \vec{F}_{,\vec{Y}}(\vec{Y},t)\vec{F}(\vec{Y},t)\Delta t + \mathcal{O}(\Delta t^2)$$

Algoritmos: Runge-Kutta

Entonces,

$$\vec{Y}(t+\Delta t) = \vec{Y}(t) + \vec{F}(\vec{Y},t)\frac{\Delta t}{2} + \\ + \vec{F}(\vec{Y} + \vec{F}(\vec{Y},t)\Delta t, t+\Delta t)\frac{\Delta t}{2} + \mathcal{O}(\Delta t^3)$$

Con esto, podemos escribir el algoritmo RK2, así

$$\boxed{\begin{aligned} \vec{Y}_{n+1} &= \vec{Y}_n + \frac{1}{2}\left(\vec{K}_1 + \vec{K}_2\right)\Delta t, \\ \vec{K}_1 &= \vec{F}(\vec{Y}_n, t_n), \\ \vec{K}_2 &= \vec{F}(\vec{Y}_n + \vec{K}_1\Delta t, t_n + \Delta t). \end{aligned}}$$

Algoritmos: Runge-Kutta

De manera análoga se obtiene el algoritmo RK4,

$$
\begin{aligned}
\vec{Y}_{n+1} &= \vec{Y}_n + \frac{1}{6}\left(\vec{K}_1 + 2\vec{K}_2 + 2\vec{K}_3 + \vec{K}_4\right)\Delta t, \\
\vec{K}_1 &= \vec{F}(\vec{Y}_n, t_n), \\
\vec{K}_2 &= \vec{F}\left(\vec{Y}_n + \vec{K}_1\frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right), \\
\vec{K}_3 &= \vec{F}\left(\vec{Y}_n + \vec{K}_2\frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right), \\
\vec{K}_4 &= \vec{F}\left(\vec{Y}_n + \vec{K}_3\Delta t, t_n + \Delta t\right).
\end{aligned}
$$

Algoritmos: PEFRL

Position Extended Forest-Ruth like,

$$
\begin{aligned}
\vec{Y}_1 &= \vec{Y}_n + \xi\, \vec{Y}'_n\, \Delta t, \\
\vec{Y}'_1 &= \vec{Y}'_n + \frac{1-2\lambda}{2m}\, \vec{F}(\vec{Y}_1, t_n)\, \Delta t, \\
\vec{Y}_2 &= \vec{Y}_1 + \chi\, \vec{Y}'_1\, \Delta t, \\
\vec{Y}'_2 &= \vec{Y}'_1 + \frac{\lambda}{m}\, \vec{F}(\vec{Y}_2, t_n)\, \Delta t, \\
\vec{Y}_3 &= \vec{Y}_2 + (1-2(\chi+\xi))\, \vec{Y}'_2\, \Delta t, \\
\vec{Y}'_3 &= \vec{Y}'_2 + \frac{\lambda}{m}\, \vec{F}(\vec{Y}_3, t_n)\, \Delta t, \\
\vec{Y}_4 &= \vec{Y}_3 + \chi\, \vec{Y}'_3\, \Delta t,
\end{aligned}
$$

Algoritmos: PEFRL

Position Extended Forest-Ruth like,

$$\boxed{\begin{aligned}\vec{Y}'_{n+1} &= \vec{Y}'_3 + \frac{1-2\lambda}{2m}\vec{F}(\vec{Y}_4, t_n)\,\Delta t,\\ \vec{Y}_{n+1} &= \vec{Y}_4 + \xi\,\vec{Y}'_{n+1}\,\Delta t,\end{aligned}}$$

donde

$$\boxed{\begin{aligned}\xi &= +0{,}1786178958448091E+00,\\ \lambda &= -0{,}2123418310626054E+00,\\ \chi &= -0{,}6626458266981849E-01.\end{aligned}}$$

Algoritmos: Paso variable

Ya que en cualquier algoritmo que provenga de la serie de Taylor se tendrá un error de truncamiento local, interesa mantener este error acotado (al menos intentarlo!). Recordemos que el resto de Taylor en la forma de Lagrange es,

$$R_k(x) = \frac{f^{(k+1)}(\xi_L)}{(k+1)!}x^{k+1},$$

donde, para una expansión en torno a cero, $\xi_L \in (0, x)$, con $x \ll 1$. Por otro lado, la notación $\mathcal{O}(\Delta t^n)$, puede inducir a error ya que no toma en cuenta la derivada en el resto, suponiendo que,

$$\frac{f^{(k+1)}(\xi_L)}{(k+1)!} \sim 1,$$

donde ξ_L es desconocido.

Algoritmos: Paso variable

Para *estimar* este error, una posibilidad es la siguiente,

$$\vec{Y}(t_n) \begin{cases} \xrightarrow[\Delta t]{\text{paso largo}} & \vec{Y}_b(t_n + \Delta t) \\ \xrightarrow[\Delta t/2]{\text{paso pequeño}} \xrightarrow[\Delta t/2]{\text{paso pequeño}} & \vec{Y}_s(t_n + \Delta t) \end{cases}$$

y observamos el comportamiento de la cantidad,

$$\Delta_e = \|Y_b(t_n + \Delta t) - Y_s(t_n + \Delta t)\|.$$

Supongamos ahora que llamamos Δ_i al error "ideal" que tendría cada paso.

Algoritmos: Paso variable

Para el caso RK4, el error de truncamiento local es $\mathcal{O}(\Delta t^5)$, luego proponemos el paso adaptable

$$\Delta t_{\text{est}} = \left| \frac{\Delta_i}{\Delta_e} \right|^{1/5} \Delta t,$$

de modo que el error local se mantenga "cerca" del valor ideal propuesto. Ahora, para mejorar el comportamiento del cambio de paso, introducimos los factores $S_1 = 0{,}9$ y $S_2 = 4{,}0$, y actualizamos el paso según,

$$\Delta t_{\text{nuevo}} = \begin{cases} \Delta t / S_2, & S_1 \Delta t_{\text{est}} > \Delta t / S_2 \\ S_2 \Delta t, & S_1 \Delta t_{\text{est}} < S_2 \Delta t \\ S_1 \Delta t_{\text{est}} & \text{otros casos,} \end{cases}$$

entonces $\Delta t \to \Delta t_{\text{nuevo}}$, y se repita la estimación al siguiente paso.

integrador.h

```
#ifndef INTG_H
#define INTG_H

#include <iostream>
#include <_vector>
#include <_file>
#include <string>

using namespace std;
```

integrador.h

```
class Integrador
{
  private:
  string tipo;
  Vector<double> X, Y1, Y2, Ya, Yb, temp;
  double dt, dt2, x0, xo, v0, a;
  double error_actual, tolerancia,
         dt_estimado, dt_old, S1, S2;
  double Xi, Lambda, Chi;
  Vector<double> k1,k2,k3,k4,f1,f2,f3,f4;
  Vector<double> (*f)(Vector<double>);
  size_t sz;
```

integrador.h

```
  public:
  Integrador(
  Vector<double>
  (*f)(Vector<double>),
  Vector<double>,
  double, string, double=1e-9);

  ~Integrador();
  Vector<double> step();
  void intervalo(File &,double,double);
  void propaga(double);
  Vector<double> x();
};

#endif
```

test.cc

```
#include "integrador.h"
#include <_file>
#include <_vector>

Vector<double> f(Vector<double> x)
{
  Vector<double> temp(x.size());
  temp[0] =  x[1];
  temp[1] = 5*(1-x[0]*x[0])*x[1]-x[0];
  return temp;
}
```

test.cc

```
int main()
{
  Vector<double> X(2);
  X[0] = 0;
  X[1] = 1;

  Integrador I(f,X,0.01,"rk4a",1e-5);

  File file("rk4a.dat",'e');

  I.intervalo(file,0,20);

  return 0;
}
```