Clase 3:
Integración numérica

Integración numérica

- Motivación en el marco de la materia
- Métodos de integración numérica (problemas de valor inicial)
  - Euler
  - Runge-Kutta orden 4 (RK4)
  - Odeint (Scipy)
- Bibliografía

Motivación en el marco de la materia

- Sistemas dinámicos, autónomos, unidimensionales, regidos por ODE

  $\dot{x} = dx/dt = f(x)$ → campo vector

- Podíamos buscar puntos fijos y ver su estabilidad

  $\dot{x} = f(x) = 0$ → raíces de f(x)

  $\left.\frac{df}{dx}\right|_{x^*} = f'(x^*)$

- Pero qué pasaría si pudiéramos directamente **integrar**? $x(t)$

Resolver ODE con c.i.: **problema de valor inicial**

Método de integración de Euler

Tenemos una ODE $$\frac{dx}{dt} = f(t, x)$$

Aproximamos la derivada por el cociente incremental $$\frac{dx}{dt} \simeq \frac{\Delta x}{\Delta t}$$

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} = f(t, x)$$

$$x(t + \Delta t) = x(t) + f(t, x)\Delta t$$

Regla iterativa para integrar en pasos de tiempo h (chico)

$$x_{n+1} = x_n + hf(t_n, x_n)$$ **(Taylor)**

Error (truncamiento)

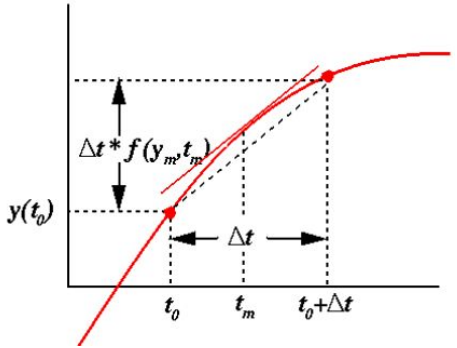
$$\mathcal{O}(h^2)$$

Método de punto medio

Euler puede generalizarse para Taylor de orden superior, pero requiere derivadas

Una alternativa es lo que se conoce como método de punto medio, en el que avanzo en el paso h con Euler pero usando la derivada en el punto medio

$$x_{n+1} = x_n + hf(t_n + \frac{h}{2}, x_n + \frac{h}{2} f(t_n, x_n))$$



Entra dentro de la familia de métodos de **Runge-Kutta**, y es de orden 2

Métodos de Runge-Kutta de orden 4 (RK4)

Siguiendo esta lógica de métodos generalizados, podemos plantear el orden 4, con las evaluaciones intermedias

$$k_1 = f(t_n, x_n)$$
$$k_2 = f\left(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_2\right)$$
$$k_4 = f(t_n + h, x_n + hk_3)$$

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Odeint (Scipy)

scipy.integrate.solve_ivp (admite también RK)

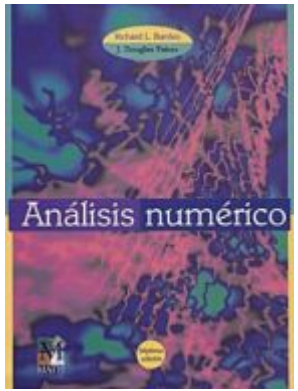
```
from scipy.integrate import odeint
solucion = odeint(campo_vector, x0, vector_tiempos)
```

Corre un solver llamado LSODA de la librería Odepack de FORTRAN, que adapta el método dependendiendo de la "dificultad" del campo vector:

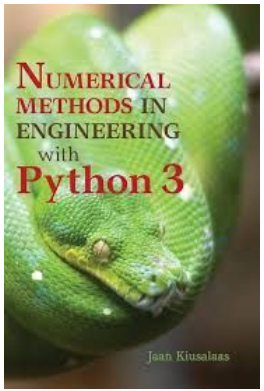
> **LSODA**, written jointly with L. R. Petzold, solves systems dy/dt = f with a dense or banded Jacobian when the problem is stiff, but it automatically selects between nonstiff (Adams) and stiff (BDF) methods. It uses the nonstiff method initially, and dynamically monitors data in order to decide which method to use.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

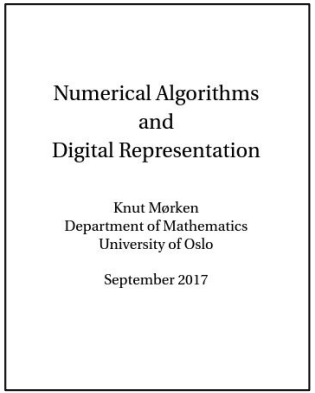
Bibliografía recomendada



Burden & Faires 2010



Kiusalaas 2013



Morken 2017

Google

stackoverflow

towards data science

YouTube