Hola

Clase computacional

Introducción al cálculo simbólico y numérico

Mecánica Clásica — Universidad de Buenos Aires · 27 de agosto

Primero: cosas burocráticas sobre el TP

- El TP es opcional.
- El TP se puede hacer en grupos de 2 personas (máximo y recomendado).
- Si lo aprueban obtienen entre 1 y 2 puntos en el segundo parcial o primer recuperatorio.
- El TP consta de dos partes: una entrega escrita (aprox 2 hojas) y una defensa oral de 5m o 10m.
- La entrega escrita tiene como fecha límite el **Jueves 24 de Septiembre para temas del primer parcial** y **Lunes 26 de Octubre para temas de cuerpo rígido**.
- La **exposición oral es el Jueves 29 de Octubre**.

Introducción al cálculo simbólico

Imports

Librería de cálculo simbólico:

```
import sympy as sp
```

Notación de derivadas temporales con punto: $\frac{\mathrm{d}u}{\mathrm{d}t} \rightarrow \dot{u}$

```
from sympy.physics.vector import init_vprinting
init_vprinting()
```

Derivar simbólicamente

- Derivada explícita

```
x = sp.symbols("x")
f = x**3
f.diff(x)
```

$$3x^2$$

- Derivada genérica

```
x, y = sp.symbols("x y")
f = sp.Function("f")(x, y)
f.diff(x)
```

$$\frac{\partial}{\partial x} f(x, y)$$

Integrar simbólicamente

```
x = sp.symbols("x")
f = x**3
```

- Integrar directo:

```
f.integrate()
```

$x^4/4$

```
sp.Integral(f, x).doit()
```

$x^4/4$

Integrar simbólicamente

```
x = sp.symbols("x")
f = x**3
```

- Declarar la integral y luego integrar:

```
integral = sp.Integral(f, x)
```

$$\int x^3 \, \mathrm{d}x$$

```
integral.doit()
```

$$x^4/4$$

Integrar simbólicamente y declaración de variables

- Sin dar información sobre a

```
x = sp.symbols("x")
a = sp.symbols("a")
integral = sp.Integral(1/(x**2 + a**2), x)
```

$$\int \frac{1}{x^2 + a^2}\,\mathrm{d}x$$

```
integral.doit()
```

$$\frac{-\frac{i\log(-ia+x)}{2} + \frac{i\log(ia+x)}{2}}{a}$$

Integrar simbólicamente y declaración de variables

- Diciendo que a es una variable positiva

```
x = sp.symbols("x")
a = sp.symbols("a", positive=True)
integral = sp.Integral(1/(x**2 + a**2), x)
```

$$\int \frac{1}{x^2 + a^2}\,\mathrm{d}x$$

```
integral.doit()
```

$$\frac{\arctan\left(\frac{x}{a}\right)}{a}$$

Despejes algebraicos

```
x = sp.symbols("x")
eq = sp.Eq(4*x**2 + 2*x + 1, 0)
```

$$4x^2 + 2x + 1 = 0$$

```
sp.solve(eq)
```

$$\left[-\frac{1}{4} - \frac{\sqrt{3}i}{4},\ -\frac{1}{4} + \frac{\sqrt{3}i}{4}\right]$$

Despejes algebraicos

```
x = sp.symbols("x")
a, b, c = sp.symbols("a b c")
eq = sp.Eq(a*x**2 + b*x + c, 0)
```

$$ax^2 + bx + c = 0$$

```
sp.solve(eq, x)
```

$$\left[\frac{-b - \sqrt{-4ac + b^2}}{2a}, \frac{-b + \sqrt{-4ac + b^2}}{2a}\right]$$

Despejes algebraicos: máquina de Atwood

```
t = sp.symbols("t")
x1, x2 = sp.Function("x1")(t), sp.Function("x2")(t)
m1, m2, g, L = sp.symbols("m1 m2 g L", positive=True)
T = sp.Function("T")(t)

eq1 = sp.Eq(m1*x1.diff(t, 2), m1*g - T)
eq2 = sp.Eq(m2*x2.diff(t, 2), m2*g - T)
constraint = sp.Eq(x1.diff(t, 2), -x2.diff(t, 2))
```

$$m_1\ddot{x}_1 = gm_1 - T, \qquad m_2\ddot{x}_2 = gm_2 - T, \qquad \ddot{x}_1 = -\ddot{x}_2$$

```
sol = sp.solve([eq1, eq2, constraint], [x1.diff(t, 2), x2.diff(t, 2), T])
```

$$\left\{T : \frac{2gm_1m_2}{m_1 + m_2},\ \ddot{x}_1 : \frac{gm_1 - gm_2}{m_1 + m_2},\ \ddot{x}_2 : \frac{-gm_1 + gm_2}{m_1 + m_2}\right\}$$

Simplificar ecuaciones

```
a, b, c = sp.symbols("a b c", positive=True)
x, y = sp.symbols("x y")

eq = sp.Eq(a*b*x + a*c*x, -a*y)
```

$$abx + acx = -ay$$

```
sp.simplify(eq)
```

$$ay = -ax(b + c)$$

Simplificar ecuaciones y un poco de magia negra

No importa que entiendan mucho esta función (le pueden preguntar a su IA de preferencia cómo funciona), pero lo único que hace es mostrar las ecuaciones de forma más *natural*.

```
def clean_eq(eq):
    expr = sp.factor(eq.lhs - eq.rhs)
    keep = [f for f in sp.Mul.make_args(expr) if f.is_nonzero is not True]
    return sp.Eq(sp.Mul(*keep), 0)
```

Simplificar ecuaciones y un poco de magia negra

```
a, b, c = sp.symbols("a b c", positive=True)
x, y = sp.symbols("x y")

eq = sp.Eq(a*b*x + a*c*x, -a*y)
```

$$abx + acx = -ay$$

```
clean_eq(eq)
```

$$bx + cx + y = 0$$

Convertir ecuaciones simbólicas a numéricas

```
sol = sp.solve([eq1, eq2, constraint], [x1.diff(t, 2), x2.diff(t, 2), T])
```

$$\left\{T : \frac{2gm_1m_2}{m_1+m_2},\ \ddot{x}_1 : \frac{gm_1-gm_2}{m_1+m_2},\ \ddot{x}_2 : \frac{-gm_1+gm_2}{m_1+m_2}\right\}$$

```
sol_T = sol[T]
```

$$\frac{2gm_1m_2}{m_1+m_2}$$

Convertir ecuaciones simbólicas a numéricas

```
sol_T = sol[T]
```

$$\frac{2gm_1m_2}{m_1 + m_2}$$

```
args = (m1, m2, g)
f_T = sp.lambdify(args, sol_T)
```

$f_T(m_1, m_2, g)$

```
f_T(1, 2, 9.8)
```

13.0666666666667

Introducción al cálculo numérico

Reducir a un problema de $2N$ ecuaciones de primer orden

Dada una ecuación de segundo orden:

$$\ddot{x} = F(x, \dot{x}, t)$$

Tomando $v = \dot{x}$ podemos escribirla como dos ecuaciones de primer orden:

$$\dot{x} = v$$
$$\dot{v} = F(x, v, t)$$

Reducir a un problema de $2N$ ecuaciones de primer orden

Dadas muchas ecuaciones de segundo orden:

$$\ddot{\mathbf{q}} = \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}, t)$$

Tomando $\mathbf{v} = \dot{\mathbf{q}}$ podemos escribirlas como el doble de ecuaciones de primer orden:

$$\dot{\mathbf{q}} = \mathbf{v}$$
$$\dot{\mathbf{v}} = \mathbf{F}(\mathbf{q}, \mathbf{v}, t)$$

De forma vectorial podemos escribir:

$$\mathbf{y} = \begin{pmatrix} \mathbf{q} \\ \mathbf{v} \end{pmatrix}, \quad \dot{\mathbf{y}} = \begin{pmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \mathbf{F}(\mathbf{q}, \mathbf{v}, t) \end{pmatrix} = \mathbf{\Phi}(\mathbf{y}, t)$$

Reducir el problema de un péndulo a dos ODEs

$$\ddot{\theta} = F\Big(\theta, \dot{\theta}, t\Big) = -\frac{g}{l}\sin(\theta) \quad \Longrightarrow \quad \begin{cases} \dot{\theta} = v \\ \dot{v} = F(\theta, v, t) = -\frac{g}{l}\sin(\theta) \end{cases}$$

$$\mathbf{y} = (\theta, v), \qquad \dot{\mathbf{y}} = \mathbf{\Phi}(\mathbf{y}, t) = \Big(\dot{\theta}, \dot{v}\Big) = \left(v, -\frac{g}{l}\sin(\theta)\right)$$

Imports

Algunas librerías de cálculo numérico que vamos a usar:

```
import numpy as np
from scipy.integrate import solve_ivp
```

Y para graficar:

```
import matplotlib.pyplot as plt
```

Resolución de ODEs numéricamente

Queremos resolver la ecuación: $\frac{\mathrm{d}y}{\mathrm{d}t} = -y$

Para esto definimos la función *right-hand side* con la signatura `fun(t, y, *)` y se la damos a `solve_ivp`:

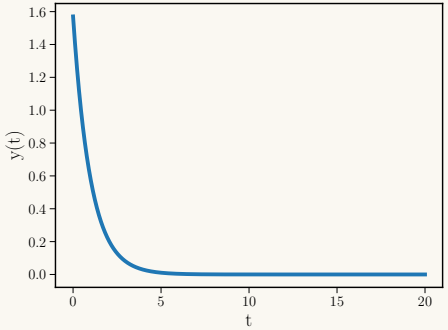
```
def rhs(t, y):
    return -y

t_eval = np.linspace(0, 20, 4001)
sol = solve_ivp(rhs, t_span=[0, 20], t_eval=t_eval, y0=[np.pi / 2])
```

```
>>> t: [ 0.000e+00  5.000e-03 ...  2.000e+01  2.000e+01]
    y: [[ 1.571e+00  1.563e+00 ...  3.675e-08  3.657e-08]]
```

Resolución de ODEs numéricamente

```
plt.xlabel("t")
plt.ylabel("y(t)")
plt.plot(sol.t, sol.y[0])
```



Resolución de ODEs numéricamente

Queremos resolver la ecuación: $\ddot{\theta} = -\frac{g}{l}\sin(\theta) \quad \Longrightarrow \quad \dot{\theta} = v, \quad \dot{v} = -\frac{g}{l}\sin(\theta)$

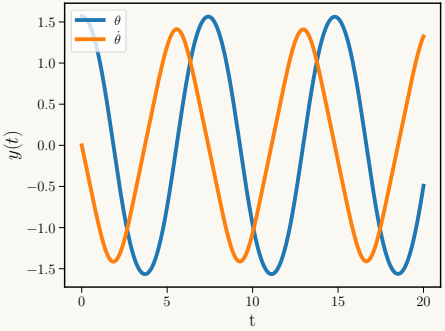
```python
def rhs(t, y, *params):
    g, l = params
    theta, v = y
    theta_dot = v
    v_dot = -g/l*np.sin(theta)
    return [theta_dot, v_dot]

t_eval = np.linspace(0, 20, 4001)
y0 = [np.pi / 2, 0] # theta, dot theta
sol = solve_ivp(rhs, t_span=[0, 20], t_eval=t_eval, y0=y0, args=(9.8, 1))
```

```
>>> t: [ 0.000e+00  5.000e-03 ...  2.000e+01  2.000e+01]
    y: [[ 1.571e+00  1.571e+00 ... -4.978e-01 -4.912e-01]
        [ 0.000e+00 -5.000e-03 ...  1.322e+00  1.324e+00]]
```

Resolución de ODEs numéricamente

```
plt.xlabel("t")
plt.ylabel(r"$y(t)$")
plt.plot(sol.t, sol.y[0], lw=4, label=r"$\theta$")
plt.plot(sol.t, sol.y[1], lw=4, label=r"$\dot\theta$")
```



Reducir el problema de los dos péndulos a 4 ODEs

$$\begin{cases} \ddot{\theta}_1 = F_1\big(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, t\big) \\ \ddot{\theta}_2 = F_2\big(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, t\big) \end{cases} \implies \begin{cases} \dot{\theta}_1 = v_1 \\ \dot{\theta}_2 = v_2 \\ \dot{v}_1 = F_1(\theta_1, \theta_2, v_1, v_2, t) \\ \dot{v}_2 = F_2(\theta_1, \theta_2, v_1, v_2, t) \end{cases}$$

$$\mathbf{y} = (\theta_1, \theta_2, v_1, v_2), \qquad \dot{\mathbf{y}} = \mathbf{\Phi}(\mathbf{y}, t) = \big(\dot{\theta}_1, \dot{\theta}_2, \dot{v}_1, \dot{v}_2\big)$$

Eso es todo