Clase 5 - Sympy

March 25, 2017

1 Introducción

En esta última clase haremos una breve revisión de las posibilidades de cálculo simbólico que ofrece la librería `sympy`. Combinaremos estas posibilidades con el poder de graficación que viene con `numpy` y con `matplotlib`.

Importamos `sympy` con la siguiente convención:

```
In [2]: import sympy as sym
```

`Sympy` nos permite modificar la impresión de los resultados. Como estaremos trabajando con polinomios y funciones, recomendaría usar el siguiente comando:

```
In [3]: sym.init_printing(use_latex=True)
```

2 Manejo de expresiones en `sympy`

2.1 Símbolos

Las moléculas de `sympy` (es decir, los objetos más comunes y sobre los cuales se opera) son los símbolos.

```
In [5]: x = sym.Symbol('x')
        f = x**2 - 1
```

2.2 Simplificación

La función `sympy.simplify` simplifica expresiones simbólicas: cancela términos que se puedan cancelar, aplica identidades trigonométricas, entre otros.

```
In [9]: g = sym.simplify((x**2 + 2*x + 1)/(x+1)**2)
        print(g)
```

```
1
```

```
In [8]: sym.simplify(sym.cos(x)**2 + sym.sin(x)**2)
```

Out[8]:

$$1$$

Si no simplificamos, `sympy` toma lo que le pasemos de forma literal.

```
In [7]: h = (x**2 + 2*x + 1)/(x+1)**2
        print(h)
```

```
(x**2 + 2*x + 1)/(x + 1)**2
```

2.3 Factorización y expansión

Si tenemos una expresión, podemos factorizarla o expandirla usando `sympy.factor` y `sympy.expand`:

```
In [10]: f = x**2 - 1
```

```
In [11]: sym.factor(f)
```

Out[11]:

$$(x - 1)(x + 1)$$

```
In [12]: g = (x+1)*(x-2)**2
```

```
In [13]: sym.expand(g)
```

Out[13]:

$$x^3 - 3x^2 + 4$$

```
In [14]: h = sym.exp(x)*x**2 + sym.exp(2*x)
         sym.factor(h)
```

Out[14]:

$$\left(x^2 + e^x\right)e^x$$

2.4 Evaluación

Cuando tenemos una expresión podemos evaluarla o sustituir una variable por un valor.

```
In [15]: raizdetres = sym.sqrt(3)
         raizdetres
```

Out[15]:

$$\sqrt{3}$$

```
In [16]: raizdetres.evalf()
```

Out[16]:

$$1.73205080756888$$

```
In [17]: f = x**2 + 1
         f.subs(x, 3)
```

Out[17]:

$$10$$

Podemos sustituir más de una variable, pero la sustitución no se hace de forma simultánea por defecto. Si necesitamos que la sustitución sea simultánea, pasamos el argumento `simultaneous=True`.

```
In [18]: y = sym.Symbol('y')
```

```
In [19]: g = x*y + x**2 - 1
         print(g.subs([(x,y), (y,3)]))
         print(g.subs([(x,y), (y,3)], simultaneous=True))
```

```
17
y**2 + 3*y - 1
```

3 Cálculo diferencial usando `sympy`

3.1 Derivadas

Con `sympy` podemos derivar (parcialmente) expresiones usando el comando `diff`:

```
In [20]: f = x**2*sym.exp(x)
```

```
In [22]: f.diff()
```

Out[22]:

$$x^2e^x + 2xe^x$$

```
In [23]: f.diff(x,2)
```

Out[23]:

$$\left(x^2 + 4x + 2\right)e^x$$

Podemos especificar con respecto a qué variable queremos derivar, también:

```
In [24]: g = x*y**2 + 2*x*y
```

```
In [26]: g.diff(y)
```

Out[26]:

$$2xy + 2x$$

3.2 Ejercicio: Aproximando funciones con polinomios usando el teorema de Taylor.

Recordemos que una función analítica se puede escribir como su serie de Taylor. Si una función f es analítica,

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \cdots$$

Implementemos una función que aproxime una función a través de su polinomio de Taylor hasta cierto grado n en cierto punto $a \in \mathbb{R}$, y que grafique el resultado y el polinomio para cada grado en un intervalo centrado en a.

Dividamos esta tarea en subtareas: 1. Dado cierto orden $n \in \mathbb{N}$ y una función f, construir una lista con todas las derivadas de f (desde 0 hasta n). 2. Iteramos sobre esta lista, evaluando y sumando a un acumulable. En cada iteración, graficamos el polinomio aproximante. 3. Graficamos la función original.

```
In [27]: # 1.
         f = sym.exp(x) + sym.cos(x)
         n = 5
         a = 0
         lista_de_derivadas = [f.diff(x,k) for k in range(n+1)]
```

```
In [28]: from math import factorial
```

```
In [29]: import numpy as np
         import matplotlib.pyplot as plt
```

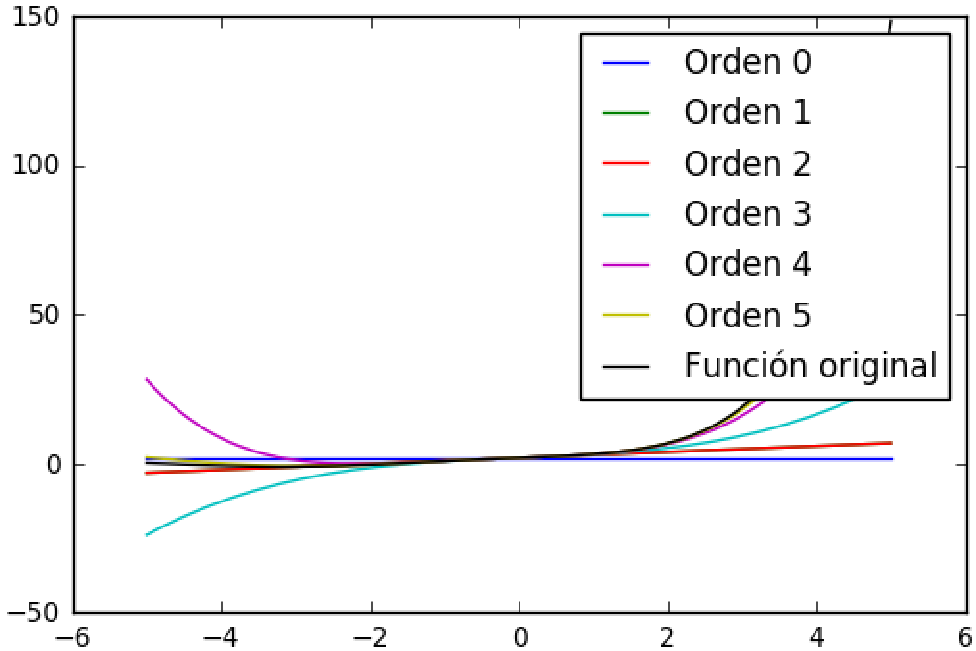
```
In [30]: dominio = np.linspace(0,1,100)
         Y = [f.subs(x, valor) for valor in dominio]
```

```
In [31]: # 2.
         polinomio = 0
         dominio = np.linspace(a-5, a+5, 100)
         plt.figure()
         for k, derivada in enumerate(lista_de_derivadas):
             polinomio += (derivada.subs(x, a)/factorial(k))*(x-a)**k
             Y = [polinomio.subs(x, valor) for valor in dominio]
             plt.plot(dominio, Y, label='Orden {}'.format(k))
```

```
In [32]: Yoriginal = [f.subs(x, valor) for valor in dominio]
         plt.plot(dominio, Yoriginal, label='Función original')
         plt.legend()
```

```
Out[32]: <matplotlib.legend.Legend at 0x7f04cef6c978>
```

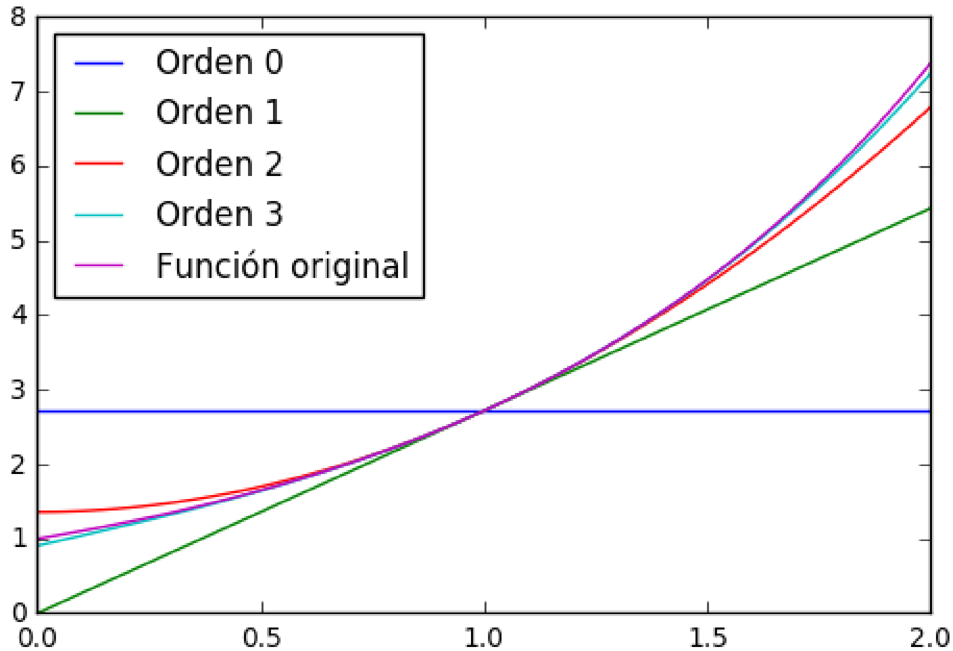
```
In [33]: plt.show()
```



Con todo esto, podemos juntar los resultados en una función y podemos pulir la gráfica cerrando más el dominio y acotando lo que aparece en el eje y:

```
In [34]: def tayloraprox(f, a, n, legendflag = True):
             lista_de_derivadas = [f.diff(x,k) for k in range(n+1)]
             polinomio = 0
             dominio = np.linspace(a-1, a+1, 100)
             Yoriginal = [f.subs(x, valor) for valor in dominio]
             minval = min(Yoriginal)
             maxval = max(Yoriginal)
             plt.figure()
             for k, derivada in enumerate(lista_de_derivadas):
                 polinomio += (derivada.subs(x, a)/factorial(k))*(x-a)**k
                 Y = [polinomio.subs(x, valor) for valor in dominio]
                 plt.plot(dominio, Y, label='Orden {}'.format(k))
             plt.plot(dominio, Yoriginal, label='Función original')
             plt.ylim(int(minval)-1, int(maxval)+1)
             if legendflag:
                 plt.legend(loc='best')
             plt.show()
             return polinomio

In [35]: tayloraprox(sym.exp(x), 1, 3)
```



Out[35]:

$$\frac{e}{6}(x-1)^3+\frac{e}{2}(x-1)^2+e(x-1)+e$$

3.3 Integración

Así como derivadas, `sympy` permite integrar con `integrate`:

```
In [36]: f = sym.cos(x)
```

```
In [37]: f.integrate()
```

Out[37]:

$$\sin(x)$$

Podemos hacer también integrales definidas:

```
In [38]: f.integrate((x,0, sym.pi/2))
```

Out[38]:

$$1$$

4 Ecuaciones lineales y no lineales en sympy

Si quieremos escribir una ecuación en sympy usamos sympy.Eq:

```
In [39]: ecuacion = sym.Eq(x**2, 1)
```

```
In [40]: ecuacion
```

Out[40]:

$$x^2 = 1$$

Para solucionar una ecuación, usamos los comandos sympy.solve y sympy.solveset.

```
In [41]: sym.solve(ecuacion)
```

Out[41]:

$$[-1, \quad 1]$$

```
In [42]: ecuacion2 = sym.Eq(x-1)
```

```
In [43]: sym.solveset(ecuacion, x)
```

Out[43]:

$$\{-1, 1\}$$

5 Matrices en sympy

sympy tiene la clase sympy.Matrix y, con ésta, muchos métodos asociados.

```
In [44]: M = sym.Matrix([(0,1),(1,0)])
```

```
In [45]: M
```

Out[45]:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
In [46]: M.diagonalize() # Saca P y D tales que M = P^-1 * D * P con D diagonal.
```

Out[46]:

$$\left(\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

```
In [47]: M.det() # Determinante
```

```
Out[47]:
```

$$-1$$

```
In [48]: M.T # Transpuesta
```

```
Out[48]:
```

$$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$$

```
In [49]: M[0,1] # Entrar a los elementos (empezando en 0)
```

```
Out[49]:
```

$$1$$

```
In [50]: M.inv() # Inversa.
```

```
Out[50]:
```

$$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$$

```
In [51]: N = sym.Matrix([(1,1),(2,2)])
```

```
In [52]: N.nullspace()
```

```
Out[52]:
```

$$\left[\begin{bmatrix}-1\\1\end{bmatrix}\right]$$

```
In [53]: N.columnspace()
```

```
Out[53]:
```

$$\left[\begin{bmatrix}1\\2\end{bmatrix}\right]$$

5.1 Ejercicio: construir la matriz de diferencias finitas.

Al intentar solucionar un problema de valor en la frontera con diferencias finitas, es común encontrarse con la siguiente matriz:

$$\begin{bmatrix}2 & 1 & 0 & 0 & \cdots & 0\\1 & 2 & 1 & 0 & \cdots & 0\\0 & 1 & 2 & 1 & \cdots & 0\\\vdots & \vdots & \ddots & \ddots & \ddots & \vdots\\0 & 0 & \cdots & 1 & 2 & 1\\0 & 0 & \cdots & 0 & 1 & 2\end{bmatrix}_{n\times n}$$

Construyámosla usando sympy. El ejercico consiste en escribir una función que pida el orden n y devuelva la matriz.

Primero, realicemos el proceso para un n fijo:

In [54]:
```python
n = 5
M = sym.zeros(n)
M
```

Out[54]:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In [55]:
```python
for i in range(n):
    for j in range(n):
        if i == j:
            M[i,j] = 2
        elif i == j-1:
            M[i,j] = 1
        elif i-1 == j:
            M[i,j] = 1
```

In [56]:
```python
M
```

Out[56]:

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

Ya tenemos la idea lista, ahora:

In [57]:
```python
def matrizespecial(n):
    M = sym.zeros(n)
    for i in range(n):
        for j in range(n):
            if i == j:
                M[i,j] = 2
            elif i == j-1:
                M[i,j] = 1
            elif i-1 == j:
                M[i,j] = 1
    return M
```

In [58]:
```python
matrizespecial(8)
```

Out[58]:

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$