**Introducción a python para computación numérica I**

Utilizando unos ejemplos, vamos a repasar varios conceptos python e introducir módulos específicos de computación numérica:

* [Gráfica de una función de una variable](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#g)
  + [Módulos python para computación numérica:](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#mod) numpy y matplotlib
  + [Funciones matemáticas elementales](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#fe) numpy
  + [Ejemplos de construción de arrays numpy unidimensionales:](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#ze) linspace, zeros\_like, ones\_like
  + [Funciones](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#lambda) **lambda**
  + [Funciones python usando](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#def) **def**
* [Series de Taylor](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#taylor)
  + [Bucles](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#for) **for**
  + [Bucles](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#while) **while**
  + [Los operadores](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#sum) +=, \*=, -= y /=
  + [Vectorización con](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#vect) numpy
  + [Operadores lógicos](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#log) **and**, **or**, **not**
  + [Animación](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#animacion)
  + [Ejercicio 1](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#ej1)
  + [Ejercicio 2](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#ej2)
* [Ejercicios propuestos](https://www.unioviedo.es/compnum/laboratorios_py/new/01_Intro1.html#prop)

**Gráfica de una función de una variable**

Para insertar las figuras en este notebook añadimos

%**matplotlib** inline

Importamos los módulos matplotlib.pyplot y numpy.

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

La función plot del módulo matplotlib.pyplot dibuja la función aproximada. Sustituye el dibujo de la función por el dibujo de una poligonal de puntos de la función. Veamos un ejemplo:

Definimos la función a dibujar 𝑓(𝑥)=𝑒𝑥f(x)=ex

f = **lambda** x: np.exp(x)

Decidimos el intervalo [𝑎,𝑏][a,b] en el que vamos a crear la función

a = -1.; b = 1.

Creamos una malla de 5 puntos equiespaciados en el intervalo [𝑎,𝑏][a,b] que se almacena en un array numpy

x = np.linspace(a,b,5)

print(x)

[-1. -0.5 0. 0.5 1. ]

Obtenemos el array numpy correspondiente a las ys

y = f(x)

print(y)

[0.36787944 0.60653066 1. 1.64872127 2.71828183]

Recordemos la sintaxis de un bucle **for**

**for** i **in** range(4):

print(i)

0

1

2

3

Esta forma de operar, donde hacemos operaciones en bloque para todos los elementos de un array numpy, se llama **vectorización** y tiene la ventaja de ser más rápido que utilizar un bucle que recorra todos los elementos y realice la misma operación. Veamos un ejemplo

**import** **time**

z = np.linspace(-10,10,1000000) *# vector con un millón de elementos*

zy = np.zeros\_like(z) *# vector de ceros con la misma estructura que z*

t = time.time()

**for** i **in** range(len(z)):

zy[i] = f(z[i])

t1 = time.time()-t

print('Sin vectorización: ', t1, ' segundos')

t = time.time()

zy = f(z)

t1 = time.time()-t

print('Con vectorización: ', t1,' segundos')

Sin vectorización: 1.6120328903198242 segundos

Con vectorización: 0.01084589958190918 segundos

Dibujamos la función

plt.figure()

plt.plot(x,y)

plt.show()

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Con solo 5 puntos la aproximación es muy basta. Por defecto, linspace crea una malla de 50 puntos. Repitamos el proceso con 50 puntos.

x = np.linspace(a,b)

y = f(x)

plt.figure()

plt.plot(x,y)

plt.show()

Imagen que contiene Forma

Descripción generada automáticamente

Sigue siendo una poligonal, pero ahora el mallado es lo suficiente fino para que dé la impresión de ser una curva suave.

Añadamos algunos detalles al dibujo:

ox = 0\*x *# array numpy con 50 ceros. Para dibujar el eje OX*

plt.figure()

plt.plot(x,y, label = 'f')

plt.plot(x,ox,'k', label = 'Eje OX') *# k, de "black" para que nos dibuje el eje OX en negro*

plt.title('Ejemplo dibujo función f') *# usando los "label", añade una leyenda*

plt.legend()

plt.show()

Imagen que contiene Diagrama de Venn

Descripción generada automáticamente

**Polinomio de Taylor**

Si desarrollamos en serie de McLaurin (polinomio de Taylor centrado en 𝑥0=0x0=0) la función 𝑓(𝑥)=𝑒𝑥f(x)=ex tenemos

𝑒𝑥=1+𝑥11!+𝑥22!+𝑥33!+𝑥44!+𝑥55!+⋯ex=1+x11!+x22!+x33!+x44!+x55!+⋯

y, teóricamente, añadiendo suficientes términos podemos aproximar la función tanto como queramos. De hecho cada uno de los polinomios

𝑃0(𝑥)𝑃1(𝑥)𝑃2(𝑥)𝑃3(𝑥)𝑃4(𝑥)=====…11+𝑥11!1+𝑥11!+𝑥22!1+𝑥11!+𝑥22!+𝑥33!1+𝑥11!+𝑥22!+𝑥33!+𝑥44!P0(x)=1P1(x)=1+x11!P2(x)=1+x11!+x22!P3(x)=1+x11!+x22!+x33!P4(x)=1+x11!+x22!+x33!+x44!…

aproxima mejor la función que el anterior. Calculemos el valor del polinomio de grado 3 en 𝑥0=0.5x0=0.5. Para ello recordemos que 𝑥00=1x00=1 y que 0!=10!=1

x0 = 0.5

polinomio = 0.

factorial = 1.

**for** i **in** range(4):

sumando = x0\*\*i/factorial

polinomio += sumando

factorial \*= i+1

print('P3(0.5) = ', polinomio)

print('np.exp(0.5) = ', np.exp(x0))

P3(0.5) = 1.6458333333333333

np.exp(0.5) = 1.6487212707001282

Si convertimos el programa en función y probamos un polinomio de grado mayor

**def** P(x0,grado):

polinomio = 0.

factorial = 1.

**for** i **in** range(grado+1):

sumando = x0\*\*i/factorial

polinomio += sumando

factorial \*= i+1

**return** polinomio

print('P(0.5, 10) = ', P(0.5, 10))

print('np.exp(0.5) = ', np.exp(0.5))

P(0.5, 10) = 1.6487212706873655

np.exp(0.5) = 1.6487212707001282

Si ahora introducimos un array numpy, la salida es también un array numpy

a = -1.; b = 1.

x = np.linspace(a,b,5)

print('x = ', x)

print('P(x, 10) = ', P(x, 10))

print('np.exp(x) = ', np.exp(x))

x = [-1. -0.5 0. 0.5 1. ]

P(x, 10) = [0.36787946 0.60653066 1. 1.64872127 2.7182818 ]

np.exp(x) = [0.36787944 0.60653066 1. 1.64872127 2.71828183]

Si en lugar de 5 puntos, hacemos esto con 50 puntos, podremos dibujar el polinomio. Dibujemos el polinomio de grado 2 con línea roja, junto con la función.

a = -1.; b = 1.

f = **lambda** x: np.exp(x)

x = np.linspace(a,b)

ox = 0\*x

plt.figure()

plt.plot(x,f(x), label = 'f')

plt.plot(x,ox,'k')

plt.plot(x,P(x,2),'r', label = 'P2')

plt.title('Ejemplo dibujo de la función y el polinomio')

plt.legend()

plt.show()

Imagen que contiene Forma

Descripción generada automáticamente

Y ahora, cambiemos el intervalo a [−3,3][−3,3] y dibujemos, usando un bucle, varios polinomios

a = -3.; b = 3.

f = **lambda** x: np.exp(x)

x = np.linspace(a,b)

ox = 0\*x

plt.figure()

plt.plot(x,f(x), label = 'f')

plt.plot(x,ox,'k')

**for** grado **in** range(1,5):

plt.plot(x,P(x,grado), label = 'P'+str(grado))

plt.title('Ejemplo dibujo función y polinomios')

plt.legend()

plt.show()

Imagen que contiene Forma

Descripción generada automáticamente

Añadiendo una línea (plt.pause(1)) podemos hacer una pequeña animación por pantalla usando **spyder**. Para ello:

* Abrir **spyder**.
* En el menú:
  + *Español:* Herramientas →→ Preferencias →→ Terminal IPython →→ Gráficas →→ Salida Gráfica →→ Salida: →→ Automática.
  + *Inglés:* Tools →→ Preferences →→ IPython Console →→ Graphics →→ Graphics backend →→ Backend: →→ Automatic.
* Reiniciamos **spyder**.
* Copiamos y ejecutamos el código de debajo.

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

*#--------------------------------------*

*# Definimos función f*

f = **lambda** x: np.exp(x)

*#--------------------------------------*

*# Definimos polinomios*

**def** P(x0,grado):

polinomio = 0.

factorial = 1.

**for** i **in** range(grado+1):

sumando = x0\*\*i/factorial

polinomio += sumando

factorial \*= i+1

**return** polinomio

*#--------------------------------------*

a = -3.; b = 3.

x = np.linspace(a,b)

OX = 0\*x

plt.figure()

plt.plot(x,f(x), label = 'f')

plt.plot(x,OX,'k')

**for** grado **in** range(1,7):

plt.plot(x,P(x,grado), label = 'P'+str(grado))

plt.title('Ejemplo dibujo de la función y los polinomios')

plt.legend()

plt.pause(1)

plt.show()

Otra estructura de control es **while**

i = 0

**while** i < 5:

print(i)

i += 1

0

1

2

3

4

**Ejercicio 1**

Utilizando un bucle **while**, escribir un programa, que utilizando el desarrollo de McLaurin para la función 𝑓(𝑥)=𝑒𝑥f(x)=ex, calcule su valor aproximado en el punto **x0 = -0.4**. El criterio de parada será que el valor del último sumando añadido **en valor absoluto** es menor que una tolerancia **tol=1.e-8** y que el número máximo de sumandos es 100, es decir **maxNumSum=100**. Comparar su valor con el valor de la función **lambda** f. Dar el número de sumandos utilizados.

**Nota:**

* Usar algún operador lógico **and**, **or**, **not**, para la condición del **while**.
* El valor absoluto de un número real a puede obtenerse con **np.abs(a)**
* Recordamos que el desarrollo de McLaurin para la función 𝑓(𝑥)=𝑒𝑥f(x)=ex es

𝑒𝑥=1+𝑥11!+𝑥22!+𝑥33!+𝑥44!+𝑥55!+⋯ex=1+x11!+x22!+x33!+x44!+x55!+⋯

%**run** Ejercicio1.py

Valor de la función en -0.4 = 0.6703200460356393

Valor de la aproximación en -0.4 = 0.67032004600776

Número de iteraciones = 10

**Ejercicio 2**

Modificar el programa de forma que calcule simultáneamente la aproximación de la función de los 50 valores contenidos en **x = np.linspace(-1,1)**. El criterio de parada será ahora que, el máximo del último sumando en valor absoluto (ahora tenemos 50 𝑥0x0 distintas y 50 últimos sumandos distintos), sea menor que que una tolerancia **tol=1.e-8** y que el número máximo de sumandos es 100, **maxNumSum=100**.

Convertir este programa en una función **funExp(x, tol, maxNumSum)** cuyos argumentos de entrada sean el array numpy **x** que contiene los 50 valores, **tol** y **maxNumSum** y cuyo argumento de salida sea un array numpy **y** que contenga los valores aproximados de la función obtenidos con el polinomio de McLaurin. Utilizar esta función para dibujar la función. Dibujar también la función utilizando la función **lambda** f definida a partir de **np.exp(x)**

**Notas:**

* **np.max(np.abs(sumando))** nos da el valor máximo de los valores absolutos de los valores contenidos en un array numpy llamado **sumando**.
* Para dibujar una línea gruesa amarilla **plt.plot(x,y,'y', linewidth = 4)**
* Para dibujar una línea azul discontinua **plt.plot(x,y,'b--')**

%**run** Ejercicio2.py

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**Ejercicios propuestos**

**Ejercicio 3**

Utilizando un bucle **while**, escribir un programa, que utilizando el desarrollo de McLaurin para la función 𝑓(𝑥)=sen(𝑥)f(x)=sen(x), calcule su valor aproximado en el punto **x0 = np.pi/4**. El criterio de parada será que el valor del último sumando añadido en valor absoluto es menor que una tolerancia **tol=1.e-8** y que el número máximo de sumandos es 100, es decir **maxNumSum=100**. Comparar su valor con el valor de la función **lambda** f. Dar el número de sumandos utilizados.

**Nota:**

* Usar algún operador lógico **and**, **or**, **not**, para la condición del **while**.
* Recordamos que el desarrollo de McLaurin para la función 𝑓(𝑥)=sen(𝑥)f(x)=sen(x) es

sen(𝑥)=𝑥11!−𝑥33!+𝑥55!−𝑥77!+⋯sen(x)=x11!−x33!+x55!−x77!+⋯

%**run** Ejercicio3.py

Valor aprox = 0.7071067811796194

Valor exacto = 0.7071067811865475

Número de iteraciones = 6

**Ejercicio 4**

Modificar el programa del ejercicio anterior de forma que calcule simultáneamente la aproximación de la función de los 50 valores contenidos en **x = np.linspace(-np.pi,np.pi)**. El criterio de parada será ahora que, el máximo del último sumando en valor absoluto (ahora tenemos 50 𝑥0x0 distintas y 50 últimos sumandos distintos), sea menor que que una tolerancia **tol=1.e-8** y que el número máximo de sumandos es 100, **maxNumSum=100**.

Convertir este programa en una función **funSin(x, tol, maxNumSum)** cuyos argumentos de entrada sean el array numpy **x** que contiene los 50 valores, **tol** y **maxNumSum** y cuyo argumento de salida sea un array numpy **y** que contenga los valores aproximados de la función obtenidos con el polinomio de McLaurin. Utilizar esta función para dibujar la función. Dibujar también la función utilizando la función **lambda** f definida a partir de **np.sin(x)**

%**run** Ejercicio4.py

Imagen que contiene Diagrama

Descripción generada automáticamente

**Ejercicio 5**

Teniendo en cuenta que el desarrollo de McLaurin de la función seno hiperbólico es la serie

sinh𝑥=𝑥11!+𝑥33!+𝑥55!+𝑥77!+𝑥99!+⋯sinhx=x11!+x33!+x55!+x77!+x99!+⋯

el coseno hiperbólico

cosh𝑥=1+𝑥22!+𝑥44!+𝑥66!+𝑥88!+⋯coshx=1+x22!+x44!+x66!+x88!+⋯

y que la tangente hiperbólica es

tanh=sinh𝑥cosh𝑥tanh=sinhxcoshx

calcular la tangente hiperbólica de 𝑥0=0.5x0=0.5 a partir de los desarrollos del senh y el cosh, usando el mismo número de términos (sumandos en el numerador y en el denominador). En el paso uno, usa un término para cada desarrollo, en el paso dos, dos sumandos y así sucesivamente. Parar cuando la diferencia en valor absoluto entre dos aproximaciones sucesivas de la tangente hiperbólica sea menor que 10−410−4.

%**run** Ejercicio5.py

Valor aprox = 0.4621171922898218

Valor exacto = 0.46211715726000974

**Ejercicio 6**

Modificar el programa del ejercicio anterior de forma que calcule simultáneamente la aproximación de la función de los 50 valores contenidos en **x = np.linspace(-3,3)**. El criterio de parada será ahora que, la máxima diferencia entre dos iteraciones consecutivas para todos los puntos (ahora tenemos 50 puntos distintos), sea menor que que una tolerancia **tol=1.e-8**.

Convertir este programa en una función **funTanh(x, tol)** cuyos argumentos de entrada sean el array numpy **x** que contiene los 50 valores, **tol** y cuyo argumento de salida sea un array numpy **y** que contenga los valores aproximados de la función obtenidos con el polinomio de McLaurin. Utilizar esta función para dibujar la función. Dibujar también la función utilizando la función **lambda** f definida a partir de **np.tanh(x)**

%**run** Ejercicio6.py

Diagrama

Descripción generada automáticamente