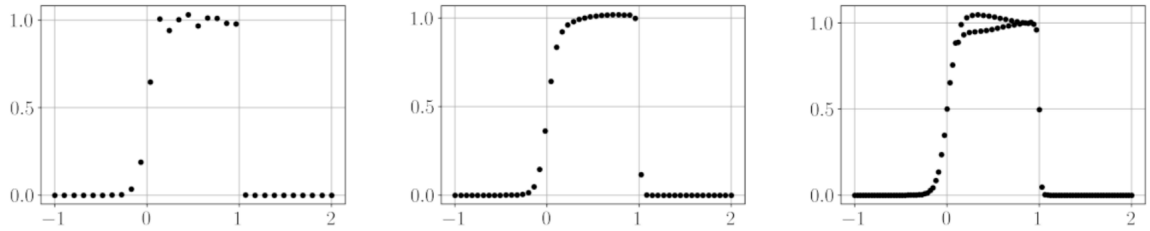
Ayudantía
SCT - Semana - 14

Integración Numérica

1. El procedimiento clásico que uno pudiera utilizar para graficar una función unidimensional en Python consiste en los siguientes pasos:

```
Paso 1: Importar librerias adecuadas
import numpy as np
import matplotlib.pyplot as plt
# Paso 2: Definicion de la funcion a graficar
f = lambda x: np.sin(x) * np.cos(x)
# Paso 3: Definicion de la cantidad de puntos equiespaciados a graficar
n = 20
# Paso 4: Definicipn del intervalo a graficar y generacion de los puntos equiespaciados
x = np.linspace(0, 10, n)
# Paso 5: Evaluacion 'vectorial' de la funcion a graficar
y = f(x)
# Paso 6: Construccion de la grafica
plt.figure()
plt.plot(x, y, '.')
plt.grid()
plt.show()
```

Sin embargo, al construir la gráfica, se tuvo que decidir la cantidad de puntos n a utilizar. En general, no existe una regla absoluta para definir n, sino que uno debe modificar el valor de n hasta obtener una gráfica adecuada y mientras la capacidad computacional lo permita. Por ejemplo, considere las siguientes 3 gráficas de una función $g(x)$. En el gráfico (a), se observa el resultado cuando se consideraron 30 puntos equiespaciados; en el gráfico (b), con 50 puntos; y, finalmente, en el gráfico (c), con 100 puntos equiespaciados.



(a) 30 puntos equiespaciados (b) 50 puntos equiespaciados (c) 100 puntos equiespaciados

Figura 1: Gráfico de $g(x)$ con distinta cantidad de puntos equiespaciados.

Como se puede apreciar, el considerar puntos equiespaciados en la abscisa (x) puede inducir puntos consecutivos muy distantes en la ordenada (y). Se puede intentar aumentar la cantidad de puntos equiespaciados para reducir este efecto, pero esto puede traer consigo una representación numérica de $g(x)$ con muchos puntos innecesarios.

Para reducir la cantidad de puntos innecesarios y, al mismo tiempo, evitar que queden puntos consecutivos muy distantes en la ordenada, se propone elegir puntos no equiespaciados, sino puntos tales que la longitud de arco entre puntos sea igual a una constante ℓ. Si la longitud de arco entre x_i y x_{i+1} está dada por $\int_{x_i}^{x_{i+1}} \sqrt{1+(g'(x))^2}\mathrm{d}x$, esto significa que los puntos en la abscisa x_i para $i \in \{1, 2, ...m-1\}$ deben satisfacer las siguientes condiciones:

$$\begin{aligned} \ell &= \int_{x_i}^{x_{i+1}} \sqrt{1+(g'(x))^2}\mathrm{d}x \\ x_i &\in [a,b] \\ x_1 &= a \\ x_m &= b \\ x_i &< x_{i+1} \end{aligned}$$

La única excepción corresponde al último intervalo que debe cumplir que $\int_{x_{m-1}}^{b} \sqrt{1+(g'(x))^2}\mathrm{d}x < \ell$. En este caso, la cantidad de puntos m es desconocida. Por tanto, se debe determinar también.

Antes de abordar las preguntas, considere lo siguiente:

- Si necesita resolver un sistema de ecuaciones lineales, debe definir claramente la matriz y el lado derecho respectivo, e indicar qué algoritmo utilizaría y las razones para elegirlo.
- Si necesita construir una interpolación polinomial, solo defina los los puntos x_j y y_j respectivos y mencione el algoritmo a utilizar. No es necesario que lo describa completamente.
- Si necesita realizar una búsqueda de raíces en 1D, utilice el método de Newton, para lo cual es necesario que describa explícitamente el input requerido, es decir, si consideramos la firma `r = Newton1D(f, fp, r0)`, usted debe explicitar los parámetros de `Newton1D`, donde `f` corresponde a la implementación callable de la función $f(x)$ a la cual se le busca la raíz; `fp` corresponde corresponde a la implementación callable de la derivada de $f(x)$, es decir, $f'(x)$; `r0` corresponde al initial guess; y `r` corresponde a la raíz, es decir, se cumple que $f(r) = 0$.
- Si necesita integrar numéricamente alguna función, debe especificar claramente el algoritmo que utilizará, la función que integrará, la definición explícita de los nodos y de los pesos que se utilizarán. En este caso, se sugiere definir como q la cantidad de nodos y pesos a utilizar, es decir, debería ser un parámetro del algoritmo.
- Si necesita utilizar algún otro algoritmo, por favor incluya claramente todos los detalles respectivos y defina todos los inputs necesarios.

a) Proponga un algoritmo que, dados una función $g(x)$, su derivada $g'(x)$, un punto x_i, un largo de curva ℓ y el extremo derecho del intervalo b, entregue el siguiente valor x_{i+1} y su evaluación $g(x_{i+1})$. Recuerde que usted debe definir explícitamente todas las componentes del algoritmo para recibir el puntaje completo. No recibe puntaje el llamar a funciones genéricas con parámetros abstractos o funciones no definidas.

b) Implemente en Python el algoritmo anteriormente descrito, utilizando adecuadamente la capacidad de vectorización de NumPy. Si fuera necesario, considere que tiene a su disposición la función `x, w = gaussian_nodes_and_weights(q)` que recibe, como input, la cantidad de puntos `q` que se utilizarán en la cuadratura Gaussiana y retorna los nodos `x` y pesos `w` en el intervalo $[-1, 1]$.

2. En la tabla 1, se puede observar algunos de los valores más altos y más bajos alcanzados por la criptomoneda Ethereum en un día específico:

Fecha	High	Low
27-nov-23	2069.14	2002.88
26-nov-23	2094.10	2038.60
25-nov-23	2091.34	-1
24-nov-23	2132.48	2061.00
23-nov-23	-1	2041.46
22-nov-23	2089.51	1933.16
21-nov-23	2035.04	-1
20-nov-23	2066.41	1996.04
19-nov-23	-1	1944.90
18-nov-23	1971.46	1921.06
17-nov-23	1990.05	-1
16-nov-23	2088.66	1940.57
15-nov-23	2061.99	1968.77
⋮	⋮	⋮

Tabla 1: Tabla con algunos valores de la criptomoneda Ethereum.

A los operadores de bolsa les interesa el operador Spread, la diferencia entre el valor más alto y más bajo de la moneda en un determinado tiempo: $\text{Spread}(t) = \text{High}(t) - \text{Low}(t)$. En particular, los operadores de bolsa buscan obtener un valor medio ponderado del Spread en una ventana de tiempo, de manera que les permita decidir su próxima estrategia financiera. Sin embargo, los operadores de bolsa notan que los datos con los cuales necesitan trabajar están incompletos. En la Tabla 1, hay datos con valor -1, lo cual significa que no están disponibles.

En resumen, los operadores de bolsa se deben enfrentar a dos problemas:

- Dadas las series $y_{\text{low}}(t_k)$ e $y_{\text{high}}(t_k)$ con $k \in \{0, ..., N\}$, se debe completar los datos faltantes que se necesitan. Considere que estos datos faltantes solamente pueden aparecer en los tiempos t_j para $j \in \{2, ..., N-2\}$.
- Calcular el valor medio ponderado del Spread, que viene dada por la siguiente expresión:

$$I_{\text{spread}} = \frac{\displaystyle\int_{t_0}^{t_N} \text{Spread}(t)\omega(t)\mathrm{d}t}{\displaystyle\int_{t_0}^{t_N} \omega(t)\mathrm{d}t},$$

  donde $\omega(t) > 0$ para $t \in [t_0, t_N]$.

a) Proponga un algoritmo que permita aproximar, para algún tiempo t_k, el valor faltante de $\text{Low}(t_k)$ y/o $\text{High}(t_k)$, considerando los dos valores anteriores en los tiempos t_{k-1} y t_{k-2} y los dos valores posteriores en los tiempos t_{k+1} y t_{k+2}. Las aproximaciones se deben realizar con la función $p(t) = a + b(t - t_k) + c(t - t_k)^2$, tanto para $\text{Low}(t_k)$ como para $\text{High}(t_k)$. Luego, esta función se evalúa en t_k para obtener el valor faltante en $\text{Low}(t_k)$ o $\text{High}(t_k)$, respectivamente. Luego, para un siguiente valor faltante, se realiza lo mismo, pero con sus propios datos vecinos.

b) Proponga un algoritmo que permita calcular I_{spread}, considerando las series $y_{\text{low}}(t_k)$ e $y_{\text{high}}(t_k)$ con $k \in \{0, ..., N\}$ y la función de ponderación $\omega(t)$.

c) Implemente en Python, utilizando adecuadamente la librería NumPy (en especial su capacidad de vectorización), los procedimientos propuestos anteriormente. Para su implementación, considere que solo tiene a su disposición las siguientes funciones de la librería NumPy, además de las operaciones elementales, ciclos y condicionales propios de Python:

- `np.arange(n)`: Dado un entero positivo n, entrega un vector de largo n con números enteros desde 0 hasta $n-1$.
- `np.dot(a, b)`: Obtiene el producto interno entre los vectores `a` y `b`. En caso de que `a` sea una matriz, entrega el producto matriz-vector respectivo. Para esto último, también es posible utilizar el operador `@`.
- `np.zeros((n_rows, n_cols))`: Entrega un ndarray de dimensión `(n_rows, n_cols)`, donde cada coeficiente es igual a 0. En caso de que solo se entrega un número entero como input, es decir, `np.zeros(n)`, entonces retorna un vector de largo n con 0s en cada coeficiente.
- `np.ones((n_rows, n_cols))`: Entrega un ndarray de dimensión `(n_rows, n_cols)`, donde cada coeficiente es igual a 1. En caso de que solo se entrega un número entero como input, es decir, `np.ones(n)`, entonces retorna un vector de largo n con 1s en cada coeficiente.

- `np.concatenate((a1, a2, ...), axis=0)`: Entrega un nuevo ndarray a partir de la concatenación de una secuencia de ndarrays a lo largo de un eje existente `axis` que, por defecto, es 0 (concatena a lo largo de las filas). Por ejemplo:

```
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6], [7, 8]])
>>> np.concatenate((a, b), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
>>> np.concatenate((a, b), axis=1)
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

- `q, r = np.linalg.qr(A, mode='reduced')`: Factorización reducida QR de `A`.
- `x = triangular_solve(U, b)`: Resuelve $U\mathbf{x} = \mathbf{b}$ para $\mathbf{x}$, donde U es una matriz triangular superior.
- `trapezoid_discrete(t, y, a, b)`: Esta función entrega el resultado de aplicar el método del trapecio para integración numérica sobre las evaluaciones discretas de la abscisa t_i y ordenada y_i almacenadas en los vectores `t` e `y`, respectivamente. `a` corresponde al límite inferior de la integral definida y `b` corresponde al límite superior de la integral definida.

Al momento de implementar, usted debe decidir qué componentes se deben vectorizar y qué componentes no, considerando las funciones de NumPy antes mencionadas. Considere la siguiente firma:

```
'''
input:
N : (integer) that defines the N+1 timesteps.
t : (ndarray) (N+1)-dimensional vector data $\mathbf{t}$.
i_low : (ndarray) indices where there is a missing value in the time series Low.
For instance, in Table 1, i_low = [2,6,10,...].
i_high : (ndarray) indices where there is a missing value in the time series High.
For instance, in Table 1, i_high = [4,8,...].
w : (callable) weighting function w(t).
y_low : (ndarray) (N+1)-dimensional vector data $\mathbf{y}_{low}$.
y_high : (ndarray) (N+1)-dimensional vector data $\mathbf{y}_{high}$.

output:
i_spread : (float) weighted average value of the Spread.
y_low_complete : (ndarray) (N+1)-dimensional vector data $\mathbf{y}_{low}$ with the
missing values approximated.
y_high_complete : (ndarray) (N+1)-dimensional vector data $\mathbf{y}_{high}$ with the
missing values approximated.
'''
def compute_spread(N, t, i_low, i_high, w, y_low, y_high):
    # Your own code.
    return i_spread, y_low_complete, y_high_complete
```