F210 Programación Aplicada a Finanzas
Abril 25 2024

Este examen dura 3 horas. Tiene terminantemente prohibido hacer consultas a cualquier entidad humana o no humana que le pueda responder. Todo el material de consulta y los TPs que haya traido en su pendrive y descargue en la computadora son de uso libre. Lo mismo si tiene algun cuaderno o libro de consulta.

No se admitirán preguntas individuales. Cualquier pregunta de enunciado debe realizarse en voz alta, a fin de que la respuesta sirva para todos. Solamente se responderán preguntas relacionadas con ambiguedad en el enunciado. Si no son de enunciado, no insista, no se responderá.

Cada uno de los dos problemas debe ser subido al terminar el examen en su respectivo archivo .py con el siguiente nombre: **APELLIDO_NOMBRE_PROBLEMAX.py**.

En el archivo debe tener todo el codigo necesario para que el programa sea ejecutable. Si falta alguna librería que se quedo en su directorio o pendrive... ya sabe. Asi que cerciorese que esté completo cada script que suba.

Es SUPER IMPORTANTE QUE CUMPLA CON LA ESPECIFICACION DE LA CONSIGNA. Si resuelve el problema, pero no de la manera que se le pide que lo haga, la solución será considerada inválida. No habrá puntos por ello. Asi que no lo intente. Limitese a responder lo que se requiere.

Por último, administre bien su tiempo. Suerte!!!

1. (4 pts) **Recursividad.** Diseñe una función **totalmente recursiva** en Python base que reciba una lista de tuplas (x_i, y_i) y devuelva a la salida otra lista con las mismas tuplas, pero ordenadas de acuerdo con el criterio usual para tuplas:

   - El par (x_i, y_i) debe preceder al par (x_j, y_j) siempre que $x_i < x_j$.
   - Cuando las tuplas (x_i, y_i) y (x_j, y_j) sean tales que $x_i = x_j$, tendrá precedencia la que tenga el menor valor de y.
   - De haber dos tuplas identicas, da igual el orden de precedencia.

   Construya la función solicitada y aplíquela para ordenar los pares $[(0,1),(1,2),(0,2),(0,3),(1,1),(1,0),(1,1)]$

   Se busca que diseñe un algoritmo **completamente recursivo**. No puede valerse de funciones auxiliares dentro ni fuera de la función, ni de bucles, ni de la utilizacion de variables auxiliares globales o que pasen por argumento. La funcion debe tener el signature `ordenar(lista)` y la recursividad hacer el resto.

   De más está decir que no puede usar la función `sort()` de python base. Es justamente lo que se pretende que construya de manera puramente recursiva.

2. (6 pts) **Interpolacion por Splines.** Como habrá concluido del TP, los polinomios de Lagrange no son la mejor forma de interpolación cuando el número de puntos muestrales (x_i, y_i) es grande, pues da lugar a polinomios de grado muy alto.

   Una forma de evitar este problema es creando varias funciones polinomicas de grado bajo, que se unen entre si y aproximan por tramos al dataset.

   PARTE 1.(3 pts) El caso mas simple es el llamado *interpolación mediante una funcion lineal por tramos* , donde una dada muestra de puntos $[(x_0, y_0), (x_1, y_1) ..., (x_N, y_N)]$ con $x_i < x_{i+1}\ \forall i$ la función interpolante es construida por un conjunto de N funciones rectilineas $L_i(x)$ tal que

   $$L_i(x) = \begin{cases} y_i + \frac{y_{i+1}-y_i}{x_{i+1}-x_i}(x - x_i) & \text{si } x_i \leq x < x_{i+1} \\ 0 & \text{en otro caso} \end{cases}$$

   dando lugar a la función interpolante $L(x) = \sum_{i=0}^{N-1} L_i(x)$. La interpolación funcionará en el intervalo $[x_0, x_N)$, por la forma en que están definidos los $L_i(x)$

(a) Se le pide entonces construir en primer lugar una subclase de `poly, interpoly,`que se inicialice con signature `__init__(self,coefs,n,a,b)` (con `coefs` y `n` los parametros propios de un polinomio $p(x)$ de tipo `poly` a ser inicializados via `super()`, más dos atributos de instancia de clase adicionales a y $b \in \mathbb{R}$, con $a < b$).

La razón de ser de a y b es que la instancia de interpoly $\widetilde{p}(x)$ debe ser **callable** y devolver el valor del polinomio $p(x)$ cuando $a \leq x < b$, y 0 en caso contrario.

$$\widetilde{p}(x) = \begin{cases} p(x) & \text{si } a \leq x < b \\ 0 & \text{en otro caso} \end{cases}$$

(b) Seguidamente, debe diseñar una clase `linterp()` que tome como datos de entrada una coleccion de puntos $[(x_0, y_0), (x_1, y_1) ..., (x_N, y_N)]$ arbitraria (donde la unica restriccion es que no haya dos puntos con igual coordenada x_i) y construya con ellos la lista de `interpolys` asociados lineales $L_i(x)$ que los interpolan linealmente por tramos. Se le deja a su criterio el diseño del objeto.

Si se le pide que las instancias de clase de `linterp()`deben ser **callable**, devolviendo para un valor x de entrada la suma $L(x) = \sum_{i=0}^{N-1} L_i(x)$.

Asimismo, las instancias de `linterp()` deben tener su propio método de graficación, que denominará `interplot(self,xmin,xmax,n=100)` donde se deben graficar los puntos de la lista $[(x_0, y_0), (x_1, y_1) ..., (x_N, y_N)]$ conjuntamente con $L(x)$ para el rango $x_{\min} \leq x \leq x_{\max}$, para una grilla equiespaciada de n puntos entre $x_{\min}$ y $x_{\max}$(use una list comp acá para generar la grilla). Los puntos de la lista de datos que queden fuera del rango $x_{\min} \leq x \leq x_{\max}$ no deben ser graficados.

(c) Construya la instancia `linterp` asociada a la siguiente lista de puntos [(1,2.718) , (9,1.118), (5,1.221), (3,1.396), (7,1.154)] . Use `interplot` para el rango $[0, 10]$ y el rango $[3, 9]$.No hace falta que haga captura de pantalla, deberia correr para nosotros tambien el algoritmo y dar los plots. Verifique visualmente que la funcion interpolante pasa efectivamente por los puntos de la lista, es continua en todos lados, pero no es derivable en los puntos de interpolación (se producen quiebres en dichos puntos, que llamaremos *nodos*).

PARTE 2. (3 pts) Como es poco deseable construir funciones interpolantes no diferenciables como ocurre en `linterp`,se diseñó otro método: el de interpolación por *splines cúbicos.* Este método propone construir una función $f(x)$, que posea las siguientes propiedades, dado un conjunto inicial de N+1 puntos $[(x_0, y_0), (x_1, y_1) ..., (x_N, y_N)]$:

(a) La función $f(x)$ debe pasar por todos los nodos (x_i, y_i), $f(x_i) = y_i$ para $i = 0, ...N$.

(b) En cada subintervalo $[x_{i-1}, x_i]$ $f(x)$ es un polimonio de grado 3, para $i = 1, ...N$.

(c) La función $f(x)$ es continuamente diferenciable, esto quiere decir que es continua y tiene derivadas primeras y segundas continuas en todos los nodos (x_i, y_i) interiores $i = 1, ..., N-1$

Para cumplir con esas condiciones, vamos a usar una nueva clase, `cub_interp(),` donde a diferencia de `linterp` vamos a guardar una lista de N `interpolys` cúbicos en vez de lineales, donde el polinomio viene dado por la expresion:

$$C_i(x) = \begin{cases} a_i x^3 + b_i x^2 + c_i x + d_i & \text{si } x_i \leq x < x_{i+1} \\ 0 & \text{en otro caso} \end{cases}.$$

Cada instancia de `cub_interp()`, al igual que en el caso de linterp debe ser **callable**, devolviendo para un valor x de entrada la suma $C(x) = \sum_{i=0}^{N-1} C_i(x)$ .La función $f(x)$ que buscamos es precisamente la $C(x)$.

El problema es encontrar los coeficientes de los N interpolys, dada una muestra de N+1 puntos $(x_0, y_0), (x_1, y_1) ..., (x_N, y_N)]$.Para ello, en el `__init__` method de la clase `cub_interp` deberá resolver un sistema de ecuaciones que satisfaga las condiciones a,b,c.

De las tres propiedades enumeradas arriba, dada la forma de los interpolys cúbicos, resulta un sistema de ecuaciones que describiremos a continuacion:

(a) De la condición (a) y de la condicion de continuidad en (c) tenemos que debe cumplirse que $C_i(x_i) = y_i$ y $C_i(x_{i+1}) = y_{i+1} \ \ \forall i = 0, ...N-1,$ lo que implica que

$$\begin{aligned} y_i &= a_i x_i^3 + b_i x_i^2 + c_i x_i + d_i \qquad \forall i = 0, ..., N-1. \text{(N ecuaciones)} \\ y_{i+1} &= a_i x_{i+1}^3 + b_i x_{i+1}^2 + c_i x_{i+1} + d_i \quad \forall i = 0, ..., N-1. \text{(N ecuaciones)} \end{aligned}$$

(b) De la condicion (c) tenemos :

1. Continuidad de la derivada en nodos internos: $C_i'(x_{i+1}) = C_{i+1}'(x_{i+1}) \ \ \forall i = 0, ..., N-2$ lo que implica que

$$a_i \ 3x_{i+1}^2 + b_i 2x_{i+1} + c_i = \ a_{i+1} 3x_{i+1}^2 + b_{i+1} 2x_{i+1} + c_{i+1} \ \forall i = 0, ..., N-2 \text{ (N-1 ecuaciones)}$$

o lo que es lo mismo

$$a_i \ 3x_{i+1}^2 + b_i 2x_{i+1} + c_i - \ a_{i+1} 3x_{i+1}^2 - b_{i+1} 2x_{i+1} - c_{i+1} = 0 \ \ \forall i = 0, ..., N-2 \text{ (N-1 ecuaciones)}$$

2. Continuidad de la segunda derivada en nodos internos: $C_i''(x_{i+1}) = C_{i+1}''(x_{i+1})$, lo que implica

$$a_i \ 6x_{i+1} + b_i \ 2 = \ a_{i+1} 6x_{i+1} + b_{i+1} \ 2 \ \forall i = 0, ..., N-2 \ \ \text{(N-1 ecuaciones)}$$

o lo que es lo mismo

$$a_i \ 6x_{i+1} + b_i \ 2 - \ a_{i+1} 6x_{i+1} - b_{i+1} \ 2 \ = 0 \quad \forall i = 0, ..., N-2 \ \ \text{(N-1 ecuaciones)}$$

En total tenemos un sistema de ecuaciones con 4N incógnitas, los coeficientes de nuestros polinomios $C_i(x)$ (tenemos N polinomios cúbicos con 4 parámetros cada uno).

Podemos escribir ese vector de coeficientes en la forma

$$x = \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ \vdots \\ a_{N-1} \\ b_{N-1} \\ c_{N-1} \\ d_{N-1} \end{bmatrix}$$

Para encontrar esos 4N coeficientes, tenemos apenas 4N-2 ecuaciones, por lo que nos faltan dos ecuaciones para cerrar el sistema. Supondremos que las derivadas segundas en los extremos son 0. Esto es

$$\begin{aligned} C_0''(x_0) &= a_0 \ 6x_0 + b_0 \ 2 = 0 \\ C_{N-1}''(x_N) &= a_{N-1} \ 6x_N + b_{N-1} \ 2 = 0 \end{aligned}$$

Con lo que ahora tendremos un sistema de 4N ecuaciones con 4N incógnitas.

Para armar el sistema de ecuaciones podemos escribir en primer lugar todas las ecuaciones asociadas a la condicion (a) $C_i(x_i) = y_i$ y $C_i(x_{i+1}) = y_{i+1}$; luego todas las asociadas a las continuidad de las derivadas en nodos interiores $C_i'(x_{i+1}) = C_{i+1}'(x_{i+1})\ \forall i = 0, ..., N-2$; a continuacion la continuidad de las derivadas segundas $C_i''(x_{i+1}) = C_{i+1}''(x_{i+1})\ \forall i = 0, ..., N-2$, y para concluir las dos condiciones finales

$$\begin{aligned} C_0''(x_0) &= a_0\ 6x_0 + b_0\ 2 = 0 \\ C_{N-1}''(x_N) &= a_{N-1}\ 6x_N + b_{N-1}\ 2 = 0 \end{aligned}$$

A modo de ejemplo, si hubiera solamente tres puntos para interpolar, tendriamos solo dos polinomios cubicos , N = 2, y el sistema deberia quedarles de 8x8 con esta forma:

$$\begin{bmatrix} x_0^3 & x_0^2 & x_0 & 1 & 0 & 0 & 0 & 0 \\ x_1^3 & x_1^2 & x_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1^3 & x_1^2 & x_1 & 1 \\ 0 & 0 & 0 & 0 & x_2^3 & x_2^2 & x_2 & 1 \\ 3x_1^2 & 2x_1 & 1 & 0 & -3x_1^2 & -2x_1 & -1 & 0 \\ 6x_1 & 2 & 0 & 0 & -6x_1 & -2 & 0 & 0 \\ 6x_0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6x_2 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_1 \\ y_2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Y si hubiera 4 puntos para interpolar, tendriamos tres polinomios cubicos N =3, y queda un sistema de 12x12 con esta forma:

$$\begin{bmatrix} x_0^3 & x_0^2 & x_0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_1^3 & x_1^2 & x_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1^3 & x_1^2 & x_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_2^3 & x_2^2 & x_2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_2^3 & x_2^2 & x_2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 \\ 3x_1^2 & 2x_1 & 1 & 0 & -3x_1^2 & -2x_1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3x_2^2 & 2x_2 & 1 & 0 & -3x_3^2 & -2x_3 & -1 & 0 \\ 6x_1 & 2 & 0 & 0 & -6x_1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6x_2 & 2 & 0 & 0 & -6x_2 & -2 & 0 & 0 \\ 6x_0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6x_3 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_1 \\ y_2 \\ y_2 \\ y_3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Se pide:

- Construir la clase `cub_interp` , donde en el init method se plantee y resuelva el sistema de ecuaciones expresado arriba mediante eliminacion Gaussiana, a partir de los datos de entrada, y con el vector de coeficientes resultante se construya la lista de interpolys cúbicos asociados.
- Hacer que `cub_interp` sea **callable** y para un valor de x devuelva $C(x) = \sum_{i=0}^{N-1} C_i(x)$.
- Dotar a `cub_interp` de un método `interplot(self,xmin,xmax,n=100)` de igual funcionalidad al desarrollado para la clase `linterp`.
- Construir la instancia `cub_interp` asociada a la siguiente lista de puntos [(1,2.718) , (9,1.118), (5,1.221), (3,1.396), (7,1.154)] . Grafique `interplot` para el rango $[0, 10]$ y el rango $[3, 9]$.