F210 Programacion Aplicada a Finanzas

3 de Mayo de 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. (3 puntos) Vamos a programar el juego de la Torre de Hanoi usando objetos. La clase se llamará `Hanoi` y los atributos de clase serán:

   - Un entero `N` indicando la cantidad de discos a usar en el juego.
   - Una lista de listas llamada `estado`,que represente el **estado actual** del juego, conteniendo tres sublistas. Cada sublista representa una vara, donde de izquierda a derecha se puede leer el número identificatorio de cada disco de abajo hacia arriba. Es decir, el elemento 0 de cada sublista es el elemento que está más abajo en su respectiva vara. Una sublista vacía corresponde a una vara vacía. El número identificatorio de cada disco corresponde al radio de cada disco. La regla de que un disco de radio menor no puede estar debajo de un disco de radio mayor en una misma vara sigue vigente como siempre.
   - Una lista de listas llamada `estado_inicial`,con la misma estructura que `estado`, conteniendo la **situación inicial** del juego.
   - Una lista de listas llamada `estado_final`,con la misma estructura que `estado`, conteniendo el estado buscado **al finalizar** el juego.
   - Un entero `counter` que vaya contando la cantidad de moves que le lleva al algoritmo resolver el juego.
   - Una lista `solution` conteniendo la secuencia de movidas que va generando la instancia de Hanoi a medida que resuelve el juego. Una movida se representa por una tupla del tipo `(disco, origen, destino)` donde disco indica el numero de disco, origen la sublista de origen y destino la de destino. Por ende, solution sera una lista de tuplas.
   - Un `__init__` method que inicialice el juego para una situacion inicial y final arbitraria a definir por el usuario
   - Un metodo `validacion` que verifique que un estado dado del juego es válido (en particular lo usará para el estado inicial y el final)
   - Un método `play(self)` que resuelva el juego de Hanoi en base a la configuración propuesta.

   Puede incorporar más métodos si los necesita.

   Las diferencias entre el caso que han planteado en el TP y el caso que tendrán que resolver en esta ocasión son 3:

   (a) La cantidad de varas/sublistas ahora será mayor o igual que tres. Esto permite reducir de manera importante la cantidad de movidas para resolver un problema.

   (b) Las situaciones iniciales y finales son arbitrarias.

   (c) Se lo libera de la obligatoriedad de que las funciones sean recursivas.

En el TP ya descubrió que en el caso ideal de tener N discos con N+1 varas la solución óptima tendría 2N-1 movidas, que llamaremos *spread and collect*, que consiste en desparramar los discos en las distintas varas y luego volverlos a apilar. Se le pide que elabore una solución en el método `play` que explote esta idea.

Una vez armado el juego, se le pide que resuelva los siguientes casos particulares (el algoritmo al finalizar debe imprimir el **counter** y la **solution**):

(a) Caso 1: 15 discos con 3 varas. Situación inicial, discos multiplos de 3 en la vara 1, discos multiplos de 2 en la vara 1, el resto en la vara 3. Situación final: discos 1 a 5 en vara 1, 6 a 10 en vara 2 y 11 a 15 en vara 3.

(b) Caso 2: 15 discos con 5 varas. Situación inicial: todos los discos en vara 1, situacion final: [[14,13,1],[12,7],[11,8,6,2],[15,10],[9,5,4,3]]

2. (3 puntos) Construya una funcion `Fibonacci(N)` que encuentre los primeros N números de la secuencia de Fibonacci (con N $>$ 6 arbitrario a ingresar por el usuario) usando exclusivamente la siguiente informacion

$$\begin{aligned} F(n) &= F(n-1)+F(n-2) \qquad 2<n\leq N \\ F(4) &= 3 \\ F(7) &= 13 \end{aligned}$$

y el método de eliminación de Gauss visto en clase.

3. (4 puntos) **Método de Gauss Seidel.** Otro desarrollo importante de Gauss fue un método iterativo para resolver sistemas de ecuaciones no lineales. Suponga que usted tiene un conjunto de 2 ecuaciones con 2 incógnitas de la forma

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

Gauss-Seidel propusieron el siguiente algoritmo iterativo **teórico**:

(a) Proponer una semilla inicial $v^0 = \left(x^0, y^0\right)$

(b) Manteniendo constantes y^0 buscar una solución de la ecuacion $f\left(x, y^0\right) = 0$. Llamar al valor encontrado x^1.

(c) Manteniendo constantes x^1 encontrar una solución de la ecuacion $g\left(x^1, y\right) = 0$. Llamar al valor encontrado y^1.

(d) Una vez encontrado $v^1 = \left(x^1, y^1\right)$, compararlo contra v^0 usando la distancia euclidea $d\left(v^0, v^1\right) = \sqrt{\left(x^0 - x^1\right)^2 + \left(y^0 - y^1\right)^2}$. Si $d\left(x^0, x^1\right) < tol$ (donde tol es una tolerancia especificada por el usuario) , terminar el proceso y devolver v^1. Caso contrario, hacer $v^0 = v^1$ y repetir los pasos b-d hasta convergencia.

Note que en cada uno de los pasos b-c se resuelve una **ecuacion no lineal de una sola variable** (ya que la otra se toma como constante).

En la práctica, no se resuelven exactamente las ecuaciones $f\left(x, y^0\right) = 0$ o $g\left(x^1, y\right) = 0$ en los pasos b y c – algo que sería computacionalmente muy costoso – , y en cambio se resuelve el problema aproximante que se usa en Newton Raphson vía la recta tangente, dando lugar a las siguientes expresiones de actualizacion de $v^i = \left(x^i, y^i\right)$

$$\begin{aligned} x^{i+1} &= x^i - \alpha \frac{f\left(x^i, y^i\right)}{f'\left(x^i, y^i\right)}, \\ y^{i+1} &= y^i - \alpha \frac{g\left(x^{i+1}, y^i\right)}{g'\left(x^{i+1}, y^i\right)} \end{aligned}$$

donde α es un número positivo pequeño (ud trabaje con $\alpha = 0.1$) y $f'\left(x^i, y^i\right)$ la derivada de $f\left(x^i, y^i\right)$ respecto de x, tratando a y como si se tratara de una constante del problema, y $g'\left(x^{i+1}, y^i\right)$ la derivada de $g\left(x^{i+1}, y^i\right)$ con respecto a la variable y, tratando a x como si se tratara de una constante del problema.

La clase que vamos a construir va a tratar unicamente con funciones $f(x,y)$ y $g(x,y)$ de tipo polinomicas en dos variables, del tipo

$$\begin{aligned} p(x,y) &= \sum_{i=0}^{N}\sum_{j=0}^{M} a_{i,j}x^i y^j \\ &= a_{0,0} + a_{1,0}x + a_{0,1}y + a_{1,1}xy + ... + a_{NM}x^N y^M \end{aligned}$$

La forma de almacenar y representar las funciones f y g en la clase a construir será mediante dos matrices rectangulares F y G conteniendo los coeficientes $a_{i,j}$ de cada una. En el caso escrito arriba debería tratarse de una matriz de NxM con coeficientes

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,M} \\ a_{1,0} & a_{1,1} & & \\ \vdots & & \ddots & \\ a_{N,1} & & & a_{N,M} \end{bmatrix}$$

donde cada columna esta asociada a una potencia de y y cada fila a una potencia de x.Notese que cuando a una de las variables se la trata como una constante, el polinomio que queda (expresado en la otra variable) es univariado y puede tratar como un miembro de la clase `poly` que ud ya conoce.

Se le pide:

- Construir una clase `GS` que permita la resolución del problema iterativo de Gauss Seidel arriba formulado.

  Cada instancia de clase debe contener por atributos `x,y` con las componentes de v en cada momento del proceso, las matrices F y G que definen las funciones polinómicas a utilizar en la actualización de x y y respectivamente, y una lista `history` que debe inicializarse como una lista vacía, y que se empieza a llenar cada vez que se corra el método `solver` de la instancia, en donde se irá acumulando la secuencia de soluciones parciales $(x^0,y^0), (x^1,y^0), (x^1,y^1), (x^2,y^1)$ ... en forma de tuplas hasta que el solver converja.

  En cuanto a métodos; deberá contener un `__init__(self,v0,F,G) ,` que debe recibir un valor de inicializacion para $v^0 = (x^0,y^0)$ – que por default será 1,1 – y los objetos myarray asociados a F y G, de las dimensiones que corresponda según el problema.

  También debe tener metodos `f(self)y g(self)` que en base a los valores de F, G, `self.x` y `self.y` construyan y devuelvan los polinomios de tipo `poly` a usar en cada iteración (observe que al ir cambiando los valores de `self.x` y `self.y` los coeficientes de `f` y `g` deben actualizarse). Recuerde que `f` es el objeto poly que resulta cuando se supone x variable e y constante y `g` el que resulta de suponer y variable y x constante.

  Como los objetos `poly` son **callable** y tienen dentro un metodo para evaluar la derivada en un punto, con construir las instancias para `f` y `g` deberia bastarle para hacer las actualizaciones.

  Debe haber un método `solve(self,guess=(1,1),tol=0.0001)` donde se implemente el método Gauss Seidel a partir de una semilla inicial `guess` a introducir por el usuario y para una tolerancia arbitraria `tol`. El método debe ir actualizando el atributo history, y al finalizar devolver la solución final. A fin de evitar bucles infinitos, proponga una salida de emergencia si el numero de iteraciones supera las 100000.

  Finalmente, debe haber un método `plot_sol(self)` que grafique la secuencia de soluciones encontradas por el método $(x^0,y^0), (x^1,y^0), (x^1,y^1), (x^2,y^1)$ ... hasta su convergencia.

- Aplique lo desarrollado en el punto anterior para el problema

$$\begin{aligned} f(x,y) &= -1 + xy = 0 \\ g(x,y) &= -10 + 5x + 2y = 0 \end{aligned}$$

Si dibuja a mano la gráfica de $f(x,y) = 0$ $g(x,y) = 0$ notará que hay dos soluciones posibles.
Verifique que el algoritmo, independientemente de donde lo dispare, converge tarde o temprano al punto `(0.22540 4.43649).`
Verifique que si intercambia $f(x,y)$ con $g(x,y)$, esto es, si hace

$$\begin{aligned} f(x,y) &= -10 + 5x + 2y = 0 \\ g(x,y) &= -1 + xy = 0 \end{aligned}$$

y corre el problema de nuevo, encuentra la otra solución.Indique el nuevo valor encontrado.