uniandes

Laboratorio de Métodos Computacionales

Tarea 6 - Sistemas lineales y PCA

01-2018

uniandes

La solución a este taller debe subirse por SICUA antes de terminada la clase. Los archivos código fuente deben subirse en un único archivo `.zip` con el nombre `NombreApellido_hw6.zip`, por ejemplo yo debería subir el zip `JesusPrada_hw6.zip` (10 puntos). Recuerden que es un trabajo individual y debe ser realizado en un script de python (.py).

1. (15 points) **Comentarios**

   Por favor comenten todo su código. Específicamente, a cada variable importante o no evidente deben comentar su significado. A cada función deben comentar su propósito principal y el significado de sus parámetros. A cada iteración deben comentar su objetivo. Se acepta declarar variables con nombre evidente como `matrix` en lugar de comentar su significado.

Aclaraciones sobre python: Punteros

2. (5 points) **Variables simples**

   Cuando definimos una variable sencilla de tipo `int`, `float`, etc., su manejo es también trivial.

   Defina una variable llamada `var1` de tipo entero que contenga el número `1000`. Ahora defina una variable `var2` que tome el valor de la variable `var1`.

   Al asignarle a `var2` el valor de la variable `var1` hemos creado efectivamente una **COPIA INDEPENDIENTE** de la variable `var1` en `var2`. Esto significa que si modificamos el valor de la variable `var1`, el valor de la variable `var2` no debería verse afectado.

   A continuación modifique el valor de la variable `var2` a `2000` e imprima un mensaje que deje claro cuál es el valor de cada variable. CORRA EL PROGRAMA. Se esperaba ese resultado?

   Deje expresadas sus conclusiones en un mensaje que debe imprimir.

   En una analogía con un caso de la vida real. Supongamos que se le ha dañado el celular. Entonces, la manera más sencilla y práctica de **MODIFICARLO** es llevando el celular mismo directamente al técnico. En este caso, la manera más sencilla de **MODIFICAR** una variable es trabajar directamente con su **valor**. Así, el objeto en sí es equivalente al valor de la variable. Tal vez suene un poco extraña, pero esta analogía demostrará ser útil luego.

3. (5 points) **Punteros**

   Uno de los paradigmas más importantes para explotar a fondo las capacidades de la programación es la cuestión de los punteros. Los punteros están principalmente asociados a tipos de variables no elementales, especialmente contenedoras modificables como las listas.

Cuando definimos una variable de tipo `list`, en realidad la variable **NO** está asociada directamente al valor de la lista que hemos definido. En su lugar, la variable es un **PUNTERO**. Un **PUNTERO** es un elemento que **APUNTA** a la **DIRECCIÓN** en memoria donde se encuentra el contenido de la variable que estamos representando.

Esta situación es mucho más simple de lo que parece. Invocando otra vez el caso de la vida real, las variables más complicadas corresponderían a elementos de la vida real más difíciles de tratar. Supongamos que en un apagón le ha ocurrido algún imperfecto a la nevera. Ahora, la forma más sencilla de **MODIFICARLA** NO es llevarla al técnico. En lugar de eso, esperamos que el técnico **MODIFIQUE** la nevera a domicilio. Pero para esto, el técnico necesita saber la **DIRECCIÓN** de la casa.

Las contenedoras como las listas o arreglos de Numpy son tipos de dato complejos y dado que su tamaño es variable y pueden llegar a ser bastante grandes (como una nevera), la forma más sencilla es trabajar con este tipo de datos es manejando la dirección en memoria. En general, la variable que definimos para la lista o el arreglo **NO** está asociada a su valor. En lugar de guardar el valor de los elementos del arreglo, la variable está asociada a la dirección en memoria del arreglo. Esto quiere decir que al copiar el valor de una lista en otra variable, estamos copiando en realidad la dirección en memoria de los elementos, en lugar de los elementos en sí. Qué significa esto y por qué nos importa? Miremos:

Defina una variable de tipo lista llamada `lista1` con los elementos `1,2,3`. Defina una variable llamada `lista2`. Iguale `lista1` a `lista2`, claramente asignando `lista1` a `lista2`. Después modifique el primer valor de `lista1` con el valor `1000`.

Imprima un mensaje que deje claro cuáles son los elementos de `lista1` y de `lista2`. CORRA EL PROGRAMA.

Nota la diferencia? Al modificar `lista1` se modifica el valor de `lista2`? Por qué? Deje expresadas sus conclusiones en un mensaje que debe imprimir.

4. (5 points) **Cómo hacer copias de variables asociadas a punteros**

   Hemos confirmado que la línea de código `puntero2 = puntero1` NO crea una copia del objeto asociado a cada variable. En lugar de esto, se copian las direcciones en memoria. Entonces, no tenemos dos copias de los objetos, sino dos copias de la dirección a un mismo objeto. Esto puede ser problemático especialmente cuando uno quiere hacer operaciones sobre una lista o un arreglo, sin cambiar el arreglo original, como vieron en el caso de la eliminación gaussiana en donde la matriz de entrada es modificada iterativamente.

   Para hacer copias reales, o clones de los objetos asociados a un puntero, es necesario tomar otro camino. En el caso de las listas, la acción de *slicing* en realidad crea copias reales de las sub-listas. Esto quiere decir que para copiar una lista simplemente podemos escribir:

```
lista2 = lista1[:]
```

Otra manera de hacer una copia verdadera es exigir la creación de un objeto nuevo a partir de los elementos de otro objeto. En este caso, el objeto es de tipo `list` y, dado un elemento iterable `iter`, al llamar el constructor de la clase `list` sobre el iterable, lo que pasará es que se creará una copia elemento a elemento que serán organizados en una lista. En este caso podemos clonar una lista de la siguiente manera:

```
lista2 = list(lista1)
```

Defina una variable de tipo lista llamada `lista1` con los elementos `1,2,3`. Defina una variable llamada `lista2`. Haga una copia de `lista1` y guárdela en la variable `lista2`. Después modifique el primer valor de `lista1` con el valor `1000`.

Imprima un mensaje que deje claro cuáles son los elementos de `lista1` y de `lista2`. CORRA EL PROGRAMA.

Nota la diferencia? Se solucionó el problema? Deje expresadas sus conclusiones en un mensaje que debe imprimir.

Ahora, la tarea de verdad!

Trabajando con órbitas de planetas OTRA VEZ!

5. (5 points) **Cargar los datos**

   Cree una función llamada `cargar(filename)` que tome como parámetro un string llamado `filename` y retorne el arreglo de numpy producido con `np.loadtxt()`.

6. (30 points) **Matriz de covarianza**

   Cree una función llamada `covarianza(matriz)` que tome como parámetro un arreglo de numpy llamado `matriz` consistente de `N` valores (filas) de `M` dimensiones (columnas) y retorne la matriz de covarianza asociada a este cojunto de valores. La matriz de covarianza debe ser `MxM`. No se le olvide tener cuidado de no modificar el parámetro `matriz`.

7. (25 points) **PCA**

   Cree una función llamada `PCA(cov)` que tome como parámetro un arreglo de numpy llamado `cov`. Este parámetro será una matriz de covarianza de tamaño `MxM`. La función debe diagonalizar la matriz de covarianza y devolver el autovector correspondiente al mínimo autovalor.

8. (10 points) **Aplicación a órbitas de planetas**

   Dadas las posiciones `X,Y,Z` de un planeta a intervalos de un día en el archivo `planeta.txt`, al sacar la matriz de covarianza, esto nos dará una idea de la dispersión de los datos en cada variable y de la correlación entre variables. Los planetas describen órbitas elípticas, lo cual es consecuencia de la fuerza radial entre el planeta y el sol. Las elipses viven en planos, lo cual

quiere decir que bajo alguna transformación de coordenadas (rotación), habrá una variable, llámese `Z'` que permanecerá constante y en cero. Esto quiere decir que la dispersión asociada a `Z'` será 0. Al diagonalizar la matriz de covarianza, estamos efectivamente aplicando una rotación sobre la cual la matriz queda en forma diagonal. Los autovectores dictarán las direcciones principales de covarianza.

Todo esto quiere decir, que si diagonalizamos la matriz de covarianza asociada a las coordenadas de un planeta, encontraremos que la dirección del mínimo autovalor será la dirección perpendicular a la elipse, es decir, la dirección con varianza aproximadamente nula.

Al realizar PCA sobre las órbitas de los planetas podemos encontrar la dirección del plano elíptico donde se desarrolla el movimiento planetario.

Para los planetas Mercurio, y Venus, utilice las funciones anteriormente definidas para obtener el vector asociado al plano elíptico de cada planeta.

Para esto:

- Cargue los datos
- Obtenga la matriz de covarianza
- Aplique PCA sobre la matriz de covarianza

Deje explícito por medio de mensajes cuál es el vector del plano elíptico de cada planeta.

9. (10 points) **Gráficas**

   Haga una gráfica de las coordenadas de los planetas (una por cada una) y grafique sobre la misma figura el vector del plano elíptico.

   Para graficar en 3 dimensiones:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
xx,yy,zz = el vector del plano
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x,y,z, c='r', marker='.')
ax.plot([0,xx],[0,yy],[0,zz])
```

   Guarde cada figura en un archivo llamado `planet.png`