Proyecto: Álgebra lineal básica en Python

Prof. Leonardo Ignacio Martínez Sandoval
Ayud. Ayax Calderón Camacho
Ayud. Blanca Yazmín Radillo Murguía

20 de mayo de 2020

1. Motivación

Hoy en día, en muchas aplicaciones de la vida cotidiana y en la investigación matemática, se usa el álgebra lineal. Puedes encontrar algunos ejemplos en los otros proyectos de este documento.

Las cuentas que se tienen que hacer típicamente no se hacen a mano. En vez de esto, se usan herramientas computacionales que puedan hacer cuentas rápidamente. Toda la formación que hemos desarrollado en álgebra lineal sirve para saber exactamente qué pedirle a la computadora, y entender sus alcances y limitaciones.

Hay muchos lenguajes de programación con los cuales se puede hacer álgebra lineal. De hecho, en ocasiones ni siquiera es necesario usar un lenguaje de programación, pues hay muchas herramientas que se pueden usar directamente "en línea". Por ejemplo, una búsqueda en Google de "Gram-Schmidt online" da varios resultados de herramientas que se pueden usar directamente en el explorador para hacer el proceso de Gram-Schmidt en un conjunto arbitrario de vectores.

Una ventaja de usar un lenguaje de programación es que usualmente se pueden hacer operaciones mucho más grandes. Otra ventaja es que al programar podemos pasar las entradas y salidas de un problema a otro con facilidad.

De entre varios ambientes de programación en los que se puede hacer álgebra lineal (como Matlab, Wolfram Mathematica, R, etc), hemos decidido usar Python para este proyecto, pues es un lenguaje de programación versátil y de alta demanda en el ámbito laboral actual.

2. Instalación del software necesario

Lo primero que necesitarás para este proyecto será instalar Miniconda. Este es un ambiente para hacer ciencia de datos (y muchas otras cosas más) basado en Python. Es una versión miniatura de Anaconda, en la que se tienen que instalar menos cosas.

Para encontrar la versión de Miniconda para tu sistema operativo, visita

`https://docs.conda.io/en/latest/miniconda.html`

Descarga e instala la versión de Python 3.x más reciente, que al escribir estas notas es la 3.7.

Espera a que se instale Miniconda. Ya que esté instalada, para agregar los paquetes que necesitamos para usar Python en una libreta interactiva y hacer álgebra lineal numérica y simbólica, tendremos que hacer un paso más.

Si estás en Windows, abre el programa Anaconda Prompt que se instaló. Si estás en macOS o Linux, abre la terminal. Corre el siguiente comando, que instalará lo necesario. Necesitarás estar conectado a internet.

```
conda install -y scipy sympy notebook
```

Sigue las opciones por default. Esto descargará como 270MB y va a tardar algunos minutos. Estará listo cuando la terminal indique "Executing transaction: done".

3. La libreta de Jupyter

Para abrir la libreta de Jupyter, que es en donde escribiremos y correremos el código, hay que correr el comando

```
jupyter notebook
```

en Anaconda Prompt (Windows) o bien en la terminal (macOS y Linux).

Si todo sale bien, tras unos instantes se abrirá una ventana de tu navegador de internet como la de la Figura 1.

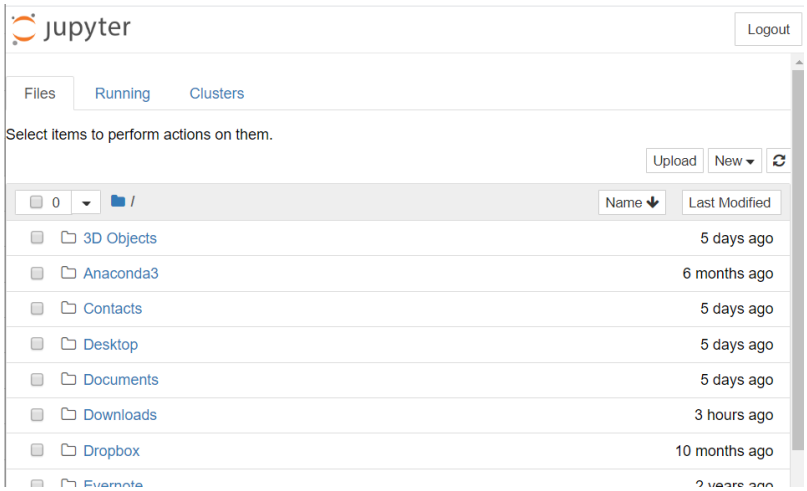

Figura 1: Ventana principal de Jupyter

Arriba a la derecha hay un botón que dice "New". Elígelo y elige la opción "Python 3". Al hacer esto, se abrirá una ventana nueva, con una libreta de trabajo como la de la Figura 2.

Perfecto, ahora tenemos una libreta nueva dentro de la cual podemos escribir código de Python. Hasta arriba puedes cambiar el título dando clic a "Untitled" y poniendo, por ejemplo, "Álgebra Lineal". Escribe en el cuadro de texto azul (celda) lo siguiente:

```
print("¡Hola mundo")
```

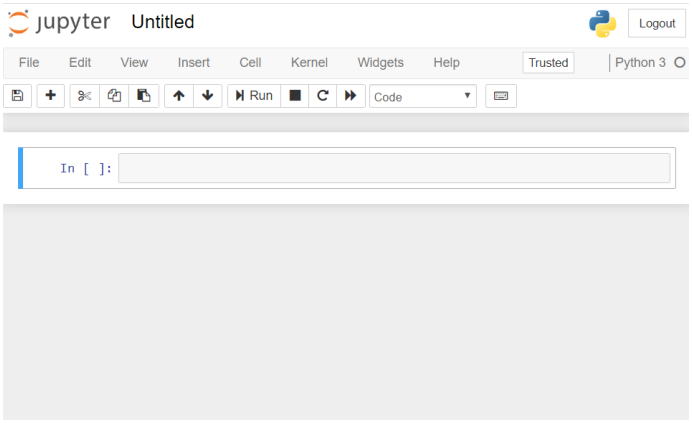

Figura 2: Nueva libreta de Jupyter

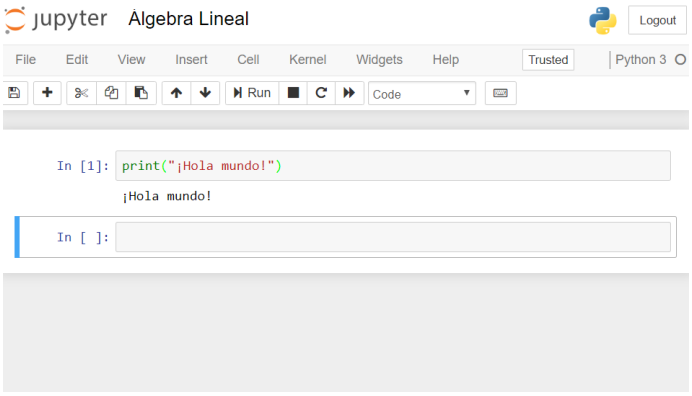

Figura 3: "Hola mundo" en Jupyter

Haz clic en el botón que dice "Run". Si es la primera vez que usas Python, ¡felicidades!, es el primer código que corres en este lenguaje. Puedes ver el resultado en la celda abajo de la que dice In [1]. La pantalla se debe ver como en la Figura 3.

Ahora escribe en la segunda celda las siguientes tres líneas:

```
x=3
y=4
print(x+y,x*y)
```

Ejecuta el código dando click en "Run". Observa que se ejecutan y muestran las operaciones $3+4$ y $3 \cdot 4$. Los símbolos $+$ y $*$ suman y multiplican números en Python, respectivamente. Mira la Figura 4.

Figura 4: Operaciones básicas en Python

4. NumPy, SciPy y funciones para hacer álgebra lineal

Todo esto está muy bien, pero queremos usar Python para hacer álgebra lineal, no sólamente operaciones básicas. Para ello necesitamos importar dos paquetes, NumPy y SciPy. Ellos tienen utilidades de que no están incluidas por default en Python.

Para ello hay que correr el siguiente código, que puedes escribir en la tercer celda. Recuerda que cada vez que escribes en una celda, debes dar clic en "Run"para correr el código. Con el teclado también puedes correr la celda en la que estás escribiendo, con la combinación "Shift + Enter".

```
import numpy as np
import scipy
```

Luego, en la cuarta celda vamos a definir dos matrices como sigue:

```
A=np.array([[1,0,3],[0,1,-1]])
B=np.array([[4,1,0,-1],[5,-1,-8,0],[0,0,1,-2]])
print(A)
print(B)
```

Esto corresponde a definir las matrices

$$A = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & -1 \end{pmatrix}$$

y

$$B = \begin{pmatrix} 4 & 1 & 0 & -1 \\ 5 & -1 & -8 & 0 \\ 0 & 0 & 1 & -2 \end{pmatrix},$$

y luego mostrarlas. Hasta ahora tu libreta se verá como la de la Figura 5.

Estamos listos para hacer nuestra primer operación de álgebra lineal en Python: una multiplicación de matrices. Corre en la sexta celda el siguiente comando

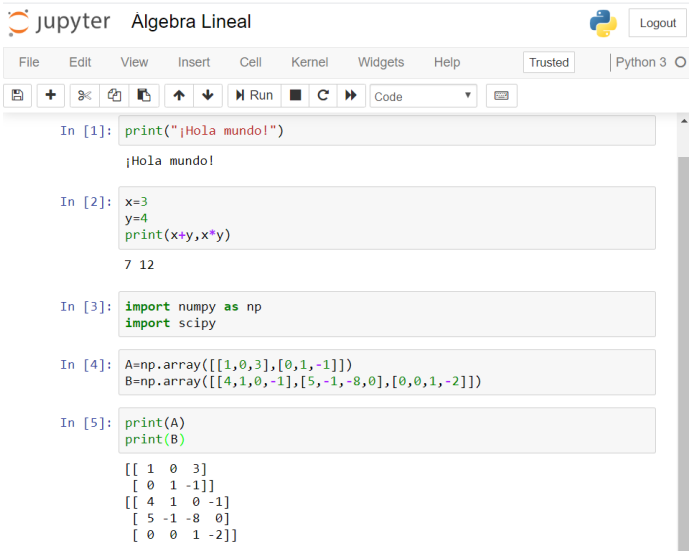

Figura 5: Definir matrices en Python

```
C = scipy.matmul(A,B)
print("La multiplicación de la matriz")
print(A)
print("y la matriz")
print(B)
print("es la matriz")
print(C)
```

La primer línea hace el producto. Las demás están ahí para mostrar la respuesta en una forma linda. ¡Felicidades! Haz hecho tu primer producto de matrices en Python. La última parte de tu libreta se debe de ver como en la Figura 6.

Dos cosas más. Si comienzas una línea de código con un símbolo de gato, es decir #, entonces Python ignora esa línea y no ejecuta nada. En la séptima celda de la libreta que estamos trabajando escribe tu nombre en el siguiente formato.

```
# Primer nombre Primer apellido
```

Ejecuta la celda y ve que no pasa nada. Finalmente, ve al menú "File". Observa que ahí puedes guardar lo que llevas. También, con "Download as..." puedes bajar la libreta en varios formatos. Guarda tu libreta en formato HTML como "Introduccion.html". Almacena este archivo en algún lugar que sepas donde fue, pues este es uno de los archivos que tienes que entregar en este proyecto. Mira la Figura 7 como referencia.

Repasa muy bien el flujo de ideas de esta sección. En realidad muchas de las cosas siguen exactamente el mismo procedimiento:

1. Definir las matrices que usarás.

Figura 6: Multiplicar matrices en Python

2. Aplicar una función de Python, de SciPy o de NumPy.
3. Mostrar la respuesta.

5. Problemas a resolver

1. Abre una nueva libreta de Jupyter. En la primer celda, importa los paquetes SciPy y NumPy como lo hicimos anteriormente. Ponle un título.

El siguiente código calcula y muestra la n-ésima potencia de una matriz M

```
print(np.linalg.matrix_power(M,n))
```

Definimos a la matriz

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & 2 & -1 & 2 \\ 3 & 1 & 3 & 1 \\ 0 & -1 & 0 & -1 \end{pmatrix}$$

.

2. En la celda 2, define la matriz A.
3. En la celda 3, encuentra A^{20} y muéstrala usando un comando "print". Necesitarás una linea para cada una de estas operaciones.

Vamos a ver ahora uno de los problemas de la librería NumPy y cómo hace las operaciones internamente. Definimos ahora a la matriz $B = \begin{pmatrix} 5 & 4 \\ 3 & -5 \end{pmatrix}$.

El siguiente código calcula y muestra el determinante de la matriz M:

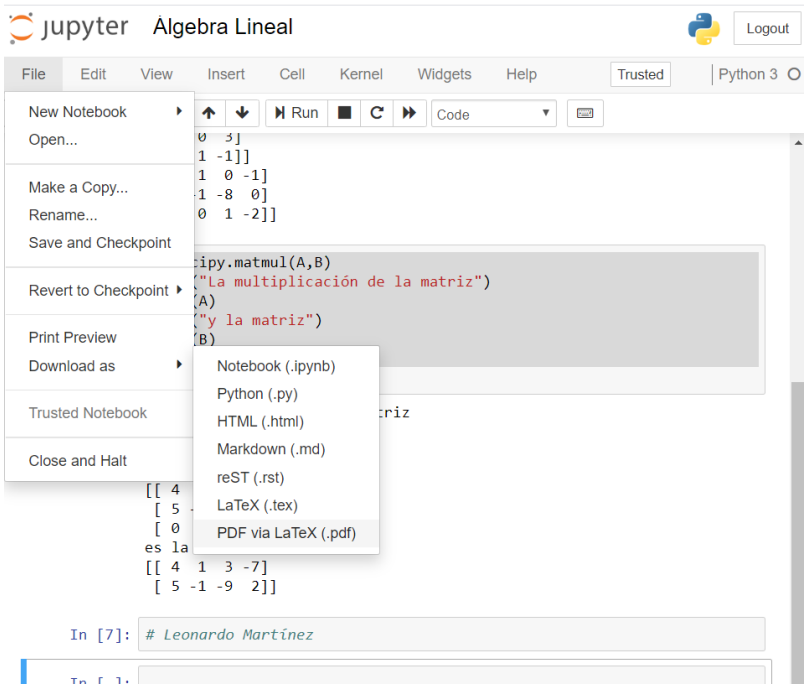

Figura 7: Comentarios y descargar libreta

```
print(np.linalg.det(M))
```

4. Observa que los vectores fila de la matriz A son todos de la forma (x, y, x, y), así que están en un subespacio vectorial de dimensión 2. ¿Por qué esto implica que la matriz no es invertible? ¿Cuál es el determinante de la matriz B? En la celda 4 calcula y muestra con Python el determinante de la matriz A y el determinante de la matriz B. ¿Qué observas? Responde estas preguntas como un comentario en la celda.

Lo que está sucediendo es que las librerías que estamos usando tienen ciertos métodos internos para hacer las cuentas que son propensos a tener *errores numéricos*. Esto sucede porque se usan *métodos numéricos*, en los que los números se les representa como decimales finitos, y entonces en el proceso se pueden perder dígitos. Hay un área importante de las matemáticas y la computación que se llama *análisis numérico*, que justo estudia cómo se comportan estos errores.

Hay una alternativa a los métodos numéricos, que se llaman *métodos simbólicos*. En este tipo de métodos la computadora usa un poco más de memoria y de tiempo para calcular valores exactos. Por ejemplo, para hacer la suma de fracción $\frac{1}{5} + \frac{1}{3}$, en un método número la computadora "hace":

$$\frac{1}{5} + \frac{1}{3} = 0{,}5 + 0{,}333\ldots = 0{,}8333\ldots,$$

mientras que en un método simbólico "hace":

$$\frac{1}{5} + \frac{1}{3} = \frac{3+5}{15} = \frac{8}{15}.$$

Los métodos numéricos son, en general, más rápidos pero más imprecisos. Los métodos simbólicos son, en general, más precisos pero más lentos.

El siguiente código importa la librería SymPy, que es con la que se pueden hacer cuentas de álgebra lineal simbólicas:

```
import sympy as sp
```

Para crear una matriz con SymPy, se usa el siguiente código, que es un poco distinto al que usa NumPy

```
B=sp.Matrix([[5,4],[3,-5]])
```

5. En la celda 5 de tu libreta importa la librería SymPy. En la celda 6 define a las matrices A y B, pero ahora como matrices de SymPy.

A continuación se muestran algunas cosas que puedes calcular con SymPy, ya que tienes una matriz M de SymPy:

```
M.det() # Determinante de M
M.rref() # Forma escalonada reducida de M
M.eigenvals() # Eigenvalores de M, con multiplicidad
M.nullspace() # Da una base del núcleo de M
```

Si quieres calcular y mostrar el resultado, puedes encerrar alguna de estas funciones en un comando print, por ejemplo,

```
print(M.det()) # Muestra e determinante de M
```

6. En la celda 7 calcula de nuevo los determinantes de A y B, y muéstralos. Observa que ahora sí se obtienen los valores exactos.

7. En la celda 8 encuentra la forma escalonada reducida de A y muéstrala.

Hagamos ahora una implementación sencilla de las fórmulas de Cramer. Considera el siguiente sistema de ecuaciones lineales:

$$\begin{cases} -2x_1 + 6x_2 + 6x_3 + 8x_4 - 4x_5 = 6 \\ 7x_1 - 2x_2 + 6x_3 + 4x_4 + 3x_5 = 14 \\ -4x_1 + 7x_2 + 5x_3 + 5x_4 + 6x_5 = 36 \\ 5x_1 + 3x_2 - 4x_3 + 8x_4 + 2x_5 = 40 \\ 6x_1 + 8x_2 + 6x_3 + x_4 + 9x_5 = 79 \end{cases}$$

Escríbelo en la forma $CX = b$, donde C es una matriz en $M_5(\mathbb{R})$, X es el vector columna de variables y b es un vector en $\mathbb{R}^5$. Recuerda que, de acuerdo a las fórmulas de Cramer, la solución está dada por

$$x_i = \frac{\det C_i}{\det C},$$

en donde C_i es la matriz obtenida de C al reemplazar la i-ésima columna por el vector b.

En Python los números se dividen usando el símbolo /. Por ejemplo, el siguiente código muestra la división del determinante de una matriz X entre el de una matriz Y.

```
print(X.det()/Y.det())
```

8. En la celda 9 encuentra la solución al sistema usando las fórmulas de Cramer. Para ello, define a las matrices
$$C, C_1, C_2, C_3, C_4, C_5$$
como matrices de SymPy (en vez de usar subíndices, llámalas simplemente $C1, C2, \ldots$ en Python). Encuentra los determinantes de cada una, y haz las cuentas correspondientes. Al final lo que tienes que mostrar es el valor de las cinco variables x_1, x_2, x_3, x_4, x_5 que son solución al sistema.

Python no sólo ayuda a que hagamos operaciones de álgebra lineal. También nos puede ayudar a conjeturar resultados matemáticos, a partir de lo cual después podemos idear una demostración. Por ejemplo, consideremos la sucesión de los números de Tribonacci, que comienza con $0, 0, 1$ y después cada término es la suma de los tres anteriores. Sus primeros términos son

$$0, 0, 1, 1, 2, 4, 7, 13, \ldots.$$

Puedes mostrar los primeros 15 términos de la sucesión de Tribonacci en Python mediante el siguiente código:

```
a,b,c=0,0,1
for j in range(15):
    print(a)
    a,b,c=b,c,a+b+c
```

Sí quieres usarlo tú, es importante que respetes los espacios iniciales de las últimas dos líneas, y que tengan la misma cantidad de espacio.

Considera ahora la matriz
$$T = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Cuando tienes una matriz M simbólica (de SymPy), para elevarla a una potencia se usa un código un poco distinto:

```
print(M**n) #Muestra la matriz M elevada a la n
```

9. En la celda 10 de tu libreta define a T como una matriz de SymPy y muestra las matrices T, T^2, T^3, T^4, T^5, T^6. ¿Puedes encontrar a los números de Tribonacci en estas matrices? ¿Dónde? Enuncia una conjetura en la celda 11 de tu libreta, usando un comentario para que Python no lo lea como código.

Si quieres, puedes demostrar la conjetura que enunciaste, pero no es necesario para la evaluación de este proyecto.

10. Finalmente, en una última celda (la 12), agrega tu primer nombre, tu primer apellido y tu número de cuenta como comentario. Descarga tu archivo tanto en formato HTML como en formato Notebook. Nombra a tus archivos "ProyectoPython.html"y "ProyectoPython.ipynb". Estos dos archivos y el archivo "Introduccion.pdf" son los documentos que tendrás que entregar.