Álgebra Lineal I
Proyecto
Álgebra lineal básica en Python y Jupyter

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, R, etc), hemos decidido usar Anaconda para este proyecto, pues se basa sobre Python, que es un lenguaje de programación versátil y de alta demanda en el ámbito laboral actual.

En este proyecto veremos cómo instalar los programas necesarios, daremos una introducción a las libretas de Jupyter, compararemos métodos numéricos y simbólicos, y veremos cómo usar álgebra lineal para hacer conjeturas matemáticas.

2. Instalación de Miniconda

Para poder trabajar en tu computadora con Python, es necesario instalarlo. Lo primero que necesitarás para este proyecto será instalar Miniconda, que es un ambiente que instala Python y varias herramientas útiles. Miniconda es una versión miniatura de Anaconda, que es uno de los ambientes más importantes para hacer ciencia de datos hoy en día (y muchas cosas más). Elegimos Miniconda para que tengas que descargar e instalar menos cosas.

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

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

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

Ahí verás varias opciones. Busca tu sistema operativo. 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. Puedes usar todas las opciones por default.

3. Instalación de paquetes adicionales

Ya que instalaste Miniconda, ya tienes Python en tu computadora. Sin embargo, antes de empezar, vamos a agregar algunos paquetes adicionales, que precisamente son lo que necesitamos para hacer álgebra lineal y mostrarla en una libreta interactiva. Los paquetes adicionales que necesitamos son:

- `notebook` - Es para tener la libretas interactivas de Jupyter.
- `numpy` - Es para hacer álgebra lineal numérica (entre otras cosas).
- `sympy` - Es para tener álgebra lineal simbólica (entre otras cosas).

Hagamos la instalación. 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 notebook numpy sympy
```

Sigue las opciones por default. Esto descargará como 270MB y va a tardar algunos minutos. Estará listo cuando la terminal indique `Executing transaction: done`. Si te das cuenta, se están instalando más paquetes que sólo los que le pedimos a Conda. Esto sucede pues se están instalando las *dependencias* automáticamente, es decir, otros paquetes que son necesarios para que funcionen Jupyter, NumPy y SymPy.

4. La libreta de Jupyter y "Hola mundo"

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.

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!")
```


Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

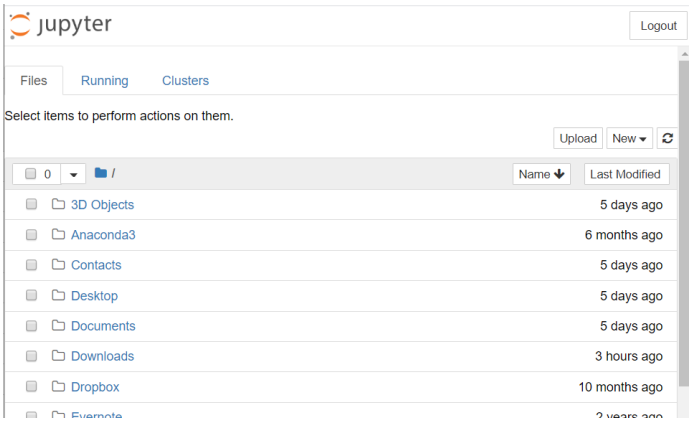

Figura 1: Ventana principal de Jupyter

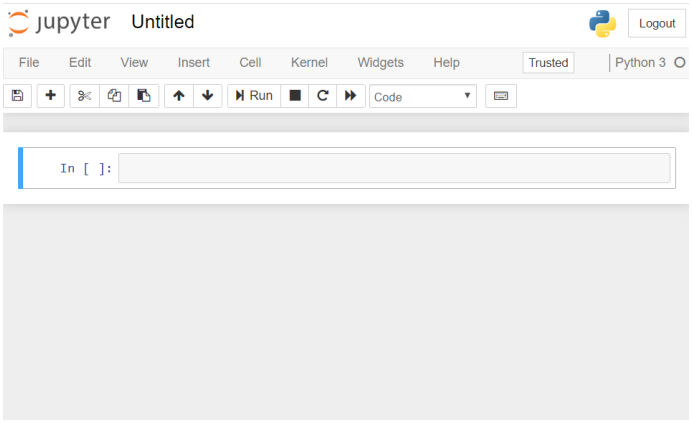

Figura 2: Nueva libreta de Jupyter

En Python es importante respetar mayúsculas, minúsculas y espacios. El comando `print` le dice a Python que quieres mostrar algo como resultado. Las comillas son importantes pues le dicen a Python que lo que está entre ellas es texto.

Haz clic en el botón de Jupyter 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,x**y)
```

Ejecuta el código dando clic en "Run". Observa que se ejecutan y muestran las operaciones $3+4$, $3\cdot 4$ y 3^4. Los símbolos $+$, $*$ y $**$ suman, multiplican y elevan a potencia números en Python,

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

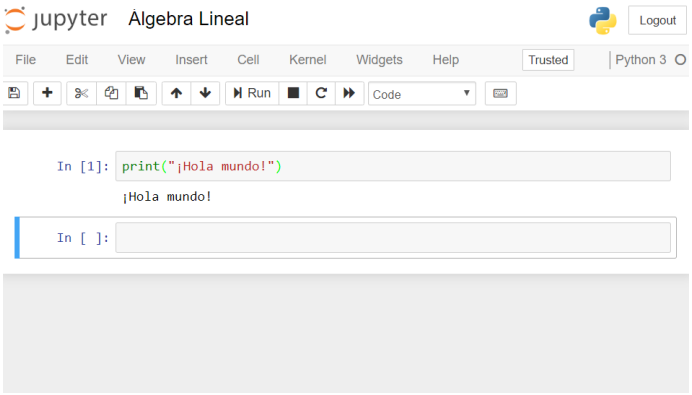

Figura 3: "Hola mundo" en Jupyter

respectivamente. Mira la Figura 4.

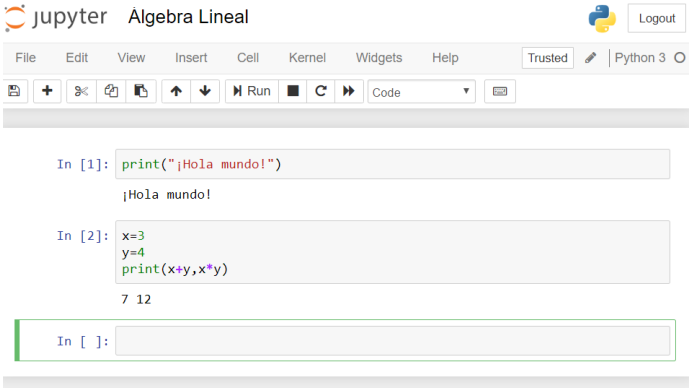

Figura 4: Operaciones básicas en Python

5. Álgebra Lineal en Python

Todo esto está muy bien, pero queremos usar Python para hacer álgebra lineal, no solamente operaciones básicas. Para ello necesitamos importar algún paquete que tenga las herramientas necesarias. En este proyecto veremos dos: NumPy y SymPy. Comencemos con el primero.

Para que estén disponibles las funciones del paquete NumPy 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
```


Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

Luego, en la cuarta celda vamos a definir dos matrices de NumPy 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.

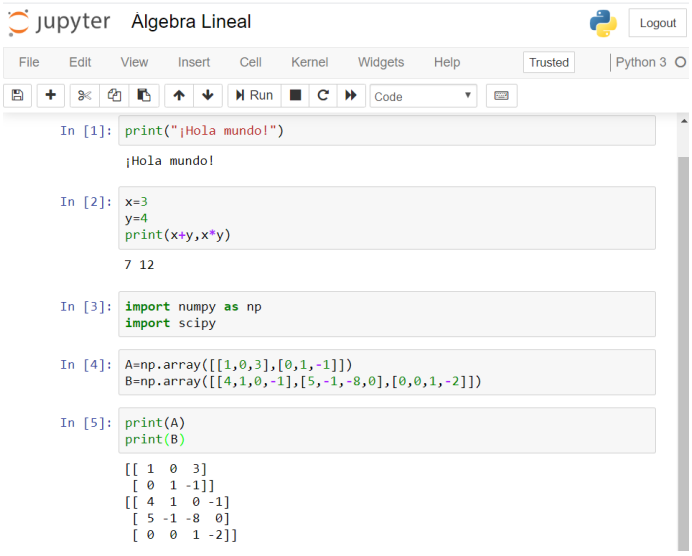

Figura 5: Definir matrices en Python

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

```
C = np.matmul(A,B)
print("La multiplicación de \n{}\n y \n{}\n es la matriz \n{}".format(A,B,C))
```

La primer línea hace el producto. La segunda lo muestra de una forma linda. Las llaves vacías `{}` y la parte de `format` le permite a Python leer variables para insertarlas en una cadena de texto. Los `\n` dan saltos de línea.

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

¡Buen trabajo! Haz hecho tu primer producto de matrices en Python. La última parte de tu libreta se debe de parecer a la de la Figura 6.

Figura 6: Multiplicar matrices en Python

Dos cosas más: comentarios y cómo guardar el archivo. 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. A una línea que comienza con gato se le llama un *comentario*. Los comentarios sirven para escribir texto en el código que te recuerde qué hace.

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 una ubicación que recuerdes. Así podrás abrirlo cuando tengas que revisar los básicos de nuevo. 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 o vectores que usarás.
2. Aplicar una función de Python, de NumPy o de SymPy.
3. Mostrar la respuesta.

La parte que puede ser un poco más tediosa es la primera, pues a veces hay que poner como entrada matrices más o menos largas. Por ejemplo, si tienes que multiplicar dos matrices de 5×5, hay que meter 50 entradas en la computadora (25 por cada una de las matrices).

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

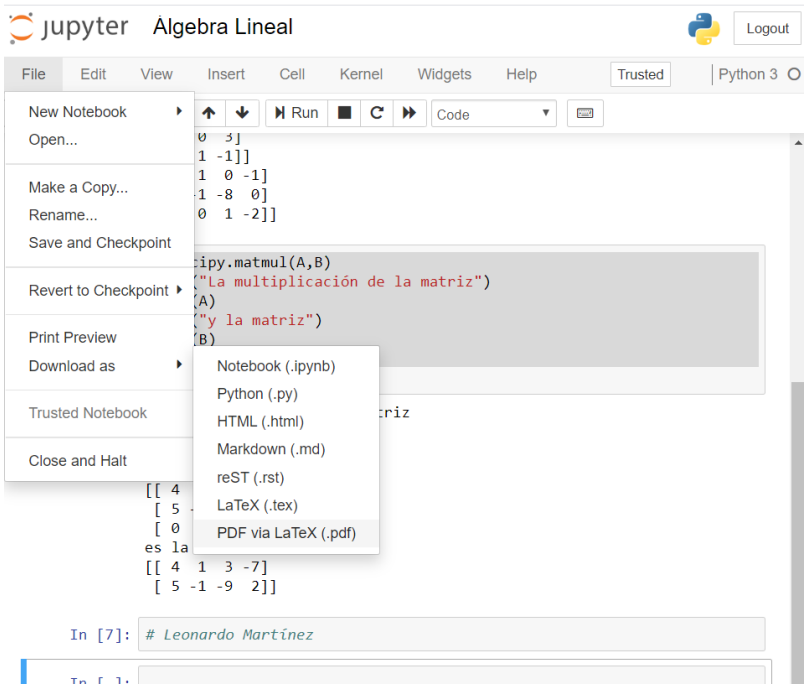

Figura 7: Comentarios y descargar libreta

Esto puede valer la pena pues ahorrará el trabajo de hacer la multiplicación de matrices a mano, que requiere de $5^3 = 125$ multiplicaciones de números (cinco por cada elección de una fila y columna) y varias sumas. Es más probable realizar errores multiplicando y sumando, que sólo copiando. Pero tal vez no valga la pena para multiplicaciones de matrices de 2×2 en donde hacerlas a mano es relativamente rápido. Con la práctica se va encontrando cuándo usar la computadora es demasiado y cuándo sí vale la pena.

6. Numpy y álgebra lineal numérica

En esta sección vamos a explorar algunas cosas más que puede hacer NumPy. Aquí comienzan los ejercicios numerados que se deben entregar y tendrás que empezar una nueva libreta. Entre cada ejercicio hay comentarios que introducen los conceptos y herramientas necesarias para el siguiente ejercicio.

1. Abre una nueva libreta de Jupyter. Ponle de título "Proyecto Python y Álgebra Lineal". En la primer celda [1], importa el paquete NumPy como lo hicimos anteriormente.

[1] En los ejercicios hacemos referencias a celdas específicas "la primera", "la celda 2", etc. Quizás rápidamente te des cuenta de que la posición de la celda, de arriba a abajo, no siempre corresponde con el número que Jupyter le da. Jupyter le da el número de acuerdo al orden en el que se ejecutaron las celdas. En los ejercicios nos referimos a la posición de la celda de arriba a abajo. Si estuviste corriendo celdas en otro orden, o experimentando, siempre puedes hacer que Jupyter numere de arriba a abajo yendo a "Kernel → Restart and run all" y confirmando. Ten cuidado:

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

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 como una matriz de NumPy. Luego, calcula A^{10} y almacénala en una variable D. Finalmente, muestra a D usando un comando `print`. Necesitarás una línea para cada una de estas operaciones.

3. Intentemos llevar más allá la función para elevar a exponentes. En la celda 3 calcula A^{-1} usando un comando `np.linalg.matrix_power`. Obtendrás un error. ¿Por qué NumPy no quiere calcular A^{-1}?

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 de NumPy M:

```
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 si lo calculas a mano? Escribe tus respuestas como comentarios en la celda 4. Después, en nuevas lineas de la celda 4, calcula y muestra con Python el determinante de la matriz A y el determinante de la matriz B. ¿Qué observas? Responde esta pregunta como comentario en la celda.

Lo que está sucediendo es que NumPy tiene 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.

Si quieres saber qué más cosas puede hacer NumPy en términos de álgebera lineal, puedes visitar la documentación en la siguiente página:

`https://numpy.org/doc/stable/reference/routines.linalg.html`

esto borra las variables que se han almacenado. Aquí no deberías tener problema con esto, pero tómalo en cuenta si usas Jupyter en el futuro.

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

7. Sympy y álgebra lineal simbólica

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 numérico la computadora "hace":

$$\frac{1}{5}+\frac{1}{3}=0{,}2+0{,}333\ldots=0{,}5333\ldots.$$

Como no puede almacenar una infinidad de dígitos, se detiene en algún momento y se pierde algo de precisión.

Por otro lado, 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.

Veamos ahora cómo hacer álgebra lineal simbólica con Python. Para ello necesitamos un paquete que haga esto. Uno de ellos es SymPy. El siguiente código lo importa:

```
import sympy as sp
```

Dependiendo del paquete que uses, es posible que necesites definir tus objetos de maneras diferentes. Por ejemplo, para crear la matriz B 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 nuevas líneas define a las matrices A y B que consideramos anteriormente, pero ahora como matrices de SymPy. Define, adicionalmente, una matriz de SymPy llamada D que sea de 2×4, que tenga las entradas que tú quieras.

A continuación se muestran algunas cosas que puedes calcular con SymPy, ya que tienes matrices M y N de SymPy. Observa que son un poco distintas a la forma en la que se hace en NumPy.

```
M*N # La multiplicación de matrices $M$ y $N$
M**n # La matriz $M$ elevada a la potencia $n$
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
M.T # La matriz transpuesta
```

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

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


Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

Para entender más a profundidad cómo funcionan los métodos anteriores, y ver otros, puedes entrar opcionalmente al tutorial que SymPy tiene en su pagina oficial. Está disponible en el enlace

`https://docs.sympy.org/latest/tutorial/matrices.html`

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

7. Ya sabemos que la matriz A no es invertible. ¿Qué rango tendrá? Una forma de determinar esto es mediante su forma escalonada reducida. En la celda 7 encuentra la forma escalonada reducida de A y muéstrala. ¿De qué rango es A? A partir de esta información y lo que hemos visto, ¿cuál es la dimensión del kernel de la transformación de $\mathbb{R}^4$ a sí mismo con regla de correspondencia $x \mapsto Ax$? Responde como comentario en esa misma celda de Python.

8. Para terminar esta sección, en la celda 9 calcula el rango de la matriz DA^tD y la dimensión del kernel de su transformación asociada.

8. Implementando la regla de Cramer

Los paquetes que instalamos no siempre tienen todas las herramientas que queremos. En el curso hablamos de cómo usar las fórmulas de Cramer para resolver sistemas de ecuaciones. Veamos cómo se pueden combinar varias funciones básicas de SymPy para construir las fórmulas de Cramer en Python. A esto se le llama una *implementación*.

Considera el siguiente sistema de ecuaciones lineales:

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

Recuerda que para escribirlo en la forma matricial $CX = b$, se toma C una matriz en $M_5(\mathbb{R})$, X un vector columna de variables y b un vector en $\mathbb{R}^5$. Recuerda también que, de acuerdo a las fórmulas de Cramer, si la matriz C es invertible entonces la única solución está dada por

$$x_i = \frac{\det C_i}{\det C}, \text{para todo } i = 1, 2, \ldots, n$$

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

El siguiente código usa los métodos `col_del` y `col_insert` de matrices de SymPy para poner el vector b en la primer columna de la matriz M y almacenar el resultado en una matriz N. **Ojo:** la primer columna es la de índice 0, por eso el cero en la primer línea.

Se asume que ya se definió una matriz b de SymPy de $1 \times n$ y una matriz M de SymPy de $n \times n$.

```
N=M.col_insert(0,b)
N.col_del(1)
N
```


Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

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

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

9. Define en tu libreta de Jupyter a las matrices

$$C, C_1, C_2, C_3, C_4, C_5$$

que necesitamos para usar las fórmulas de Cramer. Deben ser matrices de SymPy. En vez de usar subíndices, llámalas simplemente $C1, C2, C3, C4, C5$ en Python. Haz esto en la celda 9. Puedes definirlas manualmente, o sólo definir manualmente a C y usar los métodos `col_del` y `col_insert` para definir a las demás.

10. En la celda 10 encuentra los determinantes de cada una, y haz las cuentas correspondientes para obtener la solución al sistema de ecuaciones usando las fórmulas de Cramer. 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. Define la matriz de SymPy $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$ con los números concretos que encontraste y verifica que $CX = b$ realizando la multiplicación en Python.

9. Python y conjeturas matemáticas

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.

Veamos un ejemplo. La *sucesión de los números de Tribonacci*, se define como sigue. Comienza con $0, 0, 1$ y luego cada término es la suma de los tres anteriores. Sus primeros términos son:

$$0, 0, 1, 1, 2, 4, 7, 13, 24, \ldots.$$

Si quieres ver más términos de la sucesión de Tribonacci puedes usar Python. El siguiente código muestra los primeros 15 términos:

```
# Define términos iniciales de Tribonacci
a,b,c=0,0,1

# Se van a mostrar 15 términos
for j in range(15):
    print("El término {} de la sucesión de Tribonacci es {}".format(j,a))
    # Ahora cambiamos a,b,c de acuerdo a la recursión de Tribonacci
    a,b,c=b,c,a+b+c
```

Si quieres usarlo tú, es importante que respetes los espacios iniciales de las últimas líneas, y que tengan la misma cantidad de espacio ambos. Lo que estamos haciendo aquí es un *ciclo*, que 15 veces está cambiando las variables a, b, c de acuerdo a las reglas de Tribonacci.

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721

Los ciclos son útiles para hacer operaciones repetitivas sin tener que escribirlas una por una. Observa que la variable j la podemos usar dentro del ciclo.

Por otro lado, considera ahora la matriz

$$T = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Resulta que las potencias de la matriz T están conectadas con la sucesión de números de Tribonacci, y ahora vamos a explorar esto.

11. En la celda 11 de tu libreta define a T como una matriz de SymPy. Luego, escribe un ciclo que muestre las matrices T, T^2, T^3, T^4, T^5, T^6, T^7 y T^8.

12. Cuando tengas todas ellas, ¿puedes identificar a los números de la sucesión Tribonacci en las entradas estas matrices? ¿dónde? Enuncia una conjetura en la celda 12 de tu libreta, usando un comentario para que Python no lo lea como código. Tu conjetura debe por lo menos decir qué es lo que sucede con las entradas en el último renglón de la matriz. Si quieres, puedes demostrar la conjetura que enunciaste y decir qué sucede con las entradas restantes, pero no es necesario para la evaluación de este proyecto. En esa misma celda agrega tu nombre completo y número de cuenta como comentario.

Finalmente, descarga tu archivo en formato HTML y nómbralo "ProyectoPython.html". Este archivo es el que tendrás que entregar para la evaluación de este proyecto.

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE104721