# Práctica Dirigida 2

# Análisis y Modelamiento Numérico I


**Integrantes:**

- Chowdhury Gomez, Junal Johir 20200092K
- Carlos Ramon, Anthony Aldair 20211104E


16 de abril de 2024

# Ejercicio 7

Programe la eliminación de Gauss Jordan y muestre una base para el espacio columna de cualquier matriz A, Por ejemplo la matriz del problema 3.

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

## Solución

### Código en Python

```
#Ejercicio 7
import numpy as np
def gauus_jordan(A):
    A = A.astype(float)
    m, n = A.shape
    print("Matriz original:")
    print(A)
    print("\nProceso de eliminacion de Gauss-Jordan:\n")
    for i in range(min(m, n)):
        pivot_row = i
        for k in range(i+1, m):
            if abs(A[k, i]) > abs(A[pivot_row, i]):
                pivot_row = k
        if pivot_row != i:
            A[[i, pivot_row]] = A[[pivot_row, i]]
        for j in range(i+1, m):
            factor = A[j, i] / A[i, i]
            A[j, i:] -= factor * A[i, i:]
    for i in range(min(m, n)-1, -1, -1):
        for j in range(i):
            factor = A[j, i] / A[i, i]
            A[j, i:] -= factor * A[i, i:]

    print("Matriz escalonada reducida:")
    print(A)
    print()
A = np.array([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])
gauus_jordan(A.copy())
```

**Output del Código**

```
Matriz original:
[[ 2. -1.  0.]
 [-1.  2. -1.]
 [ 0. -1.  2.]]

Proceso de eliminación de Gauss-Jordan:

Matriz escalonada reducida:
[[2.         0.         0.        ]
 [0.         1.5        0.        ]
 [0.         0.         1.33333333]]
```

Figura 1: Output del Código en Python.

Este código imprimirá solo la matriz original y la matriz escalonada reducida después de completar el proceso de eliminación de Gauss-Jordan.

# Ejercicio 14

Descargue la data de `https://www.mathstat.dal.ca/ iron/math3210/hw4data` y ajuste la mejor parábola $y = ax^2 + bx + c$ que se acerque a esos datos, usando el siguiente procedimiento:

1. Cargue la data en Python usando
   `import scipy.io as sio`
   `data = sio.loadmat('hw4data')`
2. Plantee un sistema $Ax = y$, donde $y$ es la data, $x = [a, b, c]^T$ y $A$ es una matriz no cuadrada.
3. Como no es posible resolver $Ax = y$ directamente, resuelva el siguiente problema $A^T Ax = A^T b$ con el método de eliminación gaussiana.
4. Grafique el ajuste cuadrático y la data en un solo gráfico.

## Solución

### Código en Python

```python
#Ejercicio 14
#a)
import scipy.io as sio
data = sio.loadmat('hw4data')
y = data['y']

#b)
import numpy as np
n = len(y)
x = np.arange(1, n + 1)  # Valores de x: 1, 2, ..., n
A = np.column_stack((x**2, x, np.ones(n)))  # Columnas  x^2, x, y 1
#print(A)
# Ahora tenemos el sistema Ax = y

#c)
# Calcular A^T y A y A^T y
A_transpose_A = np.dot(A.T, A)
A_transpose_y = np.dot(A.T, y)
# Resolver el sistema A^T A x = A^T y usando eliminacion gaussiana
x = np.linalg.solve(A_transpose_A, A_transpose_y)
print(x)

#d)
import matplotlib.pyplot as plt
# Generar puntos para graficar la parabola ajustada
x_vals = np.linspace(1, n, 100)
y_vals = x[0] * x_vals**2 + x[1] * x_vals + x[2]
# Graficar los datos y la parabola ajustada
plt.plot(x_vals, y_vals, label='Ajuste cuadratico')
plt.scatter(np.arange(1, n + 1), y, color='red', label='Datos')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Ajuste cuadratico a los datos')
plt.grid(True)
plt.show()
```

**Output del Código**

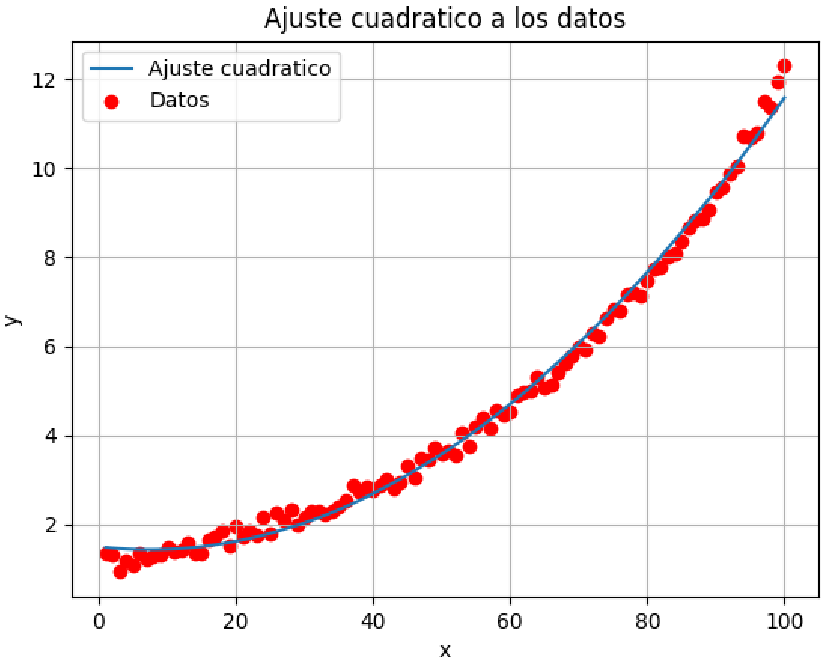


Figura 2: Ajuste cuadratico a los datos.

# Ejercicio 19

Asumiendo que se conoce una factorización LU de una matriz, diseñe un algoritmo para invertir tal matriz. Aplíquelo a la matriz del problema 3.

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

## Solución

Sabiendo que la factorización LU de A es:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -0{,}5 & 1 & 0 \\ 0 & -2/3 & 1 \end{bmatrix}, U = \begin{bmatrix} 2 & -1 & 0 \\ 0 & 1{,}5 & -1 \\ 0 & 0 & 4/3 \end{bmatrix}$$

**Código en Python**

```python
#Ejercicio 19
import numpy as np

def invertir_matriz_LU(L, U):
    # Obtenemos las dimensiones de la matriz
    n = len(L)
    # Creamos una matriz identidad del mismo tamano que A
    A_inv = np.eye(n)
    # Resolvemos el sistema de ecuaciones LUx = b
    for i in range(n):
        # Resolvemos Ly = b
        y = np.linalg.solve(L, A_inv[:, i])

        # Resolvemos Ux = y
        x = np.linalg.solve(U, y)

        # La solucion x es la i-esima columna de la matriz inversa
        A_inv[:, i] = x
    return A_inv

# Matriz A
A = np.array([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])
# Factorizacion LU conocida de A
L = np.array([[1, 0, 0], [-0.5, 1, 0], [0, -2/3, 1]])
U = np.array([[2, -1, 0], [0, 1.5, -1], [0, 0, 4/3]])
# Invertir la matriz utilizando su factorizacion LU
A_inv = invertir_matriz_LU(L, U)
print("Inversa de la matriz A:")
print(A_inv)
```

**Output del Código**

```
Inversa de la matriz A:
[[0.75 0.5  0.25]
 [0.5  1.   0.5 ]
 [0.25 0.5  0.75]]
```

Figura 3: Output del Código en Python.

# Ejercicio 21

Un fabricante de bombillas gana \$0.3 por cada bombilla que sale de la fábrica, pero pierde \$0.4 por cada una que sale defectuosa. Un día en el que fabricó 2100 bombillas obtuvo un beneficio de \$484.4. Determine el número de bombillas buenas y defectuosas según el siguiente requerimiento:

1. Modele el problema.
2. Determine la norma matricial de $A$ y $A^{-1}$.
3. Determine el número de condicionamiento de $A$.
4. Indique si está bien o mal condicionado.

## Solución

**a) Modelo del problema:**
Denotemos:

- $x$ como el número de bombillas buenas fabricadas.
- $y$ como el número de bombillas defectuosas fabricadas.

El fabricante gana \$0.3 por cada bombilla buena y pierde \$0.4 por cada bombilla defectuosa. Entonces, podemos escribir el sistema de ecuaciones para el beneficio total como:

$$\begin{cases} 0{,}3x - 0{,}4y = 484{,}4 \\ x + y = 2100 \end{cases}$$

**b) Norma matricial de $A$ y $A^{-1}$:**
El sistema en forma matricial $Ax = b$, donde:

$$A = \begin{bmatrix} 0{,}3 & -0{,}4 \\ 1 & 1 \end{bmatrix}, \quad x = \begin{bmatrix} x \\ y \end{bmatrix}, \quad b = \begin{bmatrix} 484{,}4 \\ 2100 \end{bmatrix}$$

La norma matricial de $A$ es la norma de Frobenius, que se calcula como la raíz cuadrada de la suma de los cuadrados de los elementos de la matriz:

$$\|A\| = \sqrt{0{,}3^2 + (-0{,}4)^2 + 1^2 + 1^2} = \sqrt{2{,}25} = 1{,}5$$

La inversa de $A$ se puede calcular como:

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} 1 & 0{,}4 \\ -1 & 0{,}3 \end{bmatrix} = \begin{bmatrix} 3/7 & -4/7 \\ 10/7 & 10/7 \end{bmatrix}$$

**c) Número de condición de $A$:**
El número de condición de $A$ se calcula como el producto de las normas de $A$ y $A^{-1}$:

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\| = 1{,}5x10/7 = 2{,}1428$$

**d) Condición de $A$ = 2.1428:**
Por lo tanto, podemos concluir que el sistema es relativamente estable y que el resultado proporcionado por el modelo no es muy sensible a pequeñas variaciones en los datos de entrada.

Sin embargo, es importante tener en cuenta que valores de condición mayores a 1 indican que el sistema es menos estable y más sensible a pequeñas perturbaciones. En este caso, el valor de 2.1428 sugiere que hay una cierta sensibilidad, pero no es alarmante.