

Escuela Politécnica Nacional

Nombre: Marco Marcillo

Tema: [Taller 18] Eliminación gaussiana vs Gauss-Jordan

Compare la complejidad computacional del método de eliminación gaussiana vs Gauss-Jordan.

```
In [7]: #-*. coding: utf-8 -*-
"""
Pythón
25 / 06 / 2025
@author: z_tjona
"""

# ----- logging -----
import logging
from sys import stdout
from time import time as time
import numpy as np
import matplotlib.pyplot as plt
import time

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s [%(levelname)s] %(message)s',
    stream=stdout,
    datefmt='%m-%d %H:%M:%S',
)
logging.info(f'datetime: {time.time()}')

# =====
def eliminacion_gaussiana(A: np.ndarray) -> np.ndarray:
    """
    Eliminación gaussiana
    """
    start_time = time.time()

    if not isinstance(A, np.ndarray):
        raise ValueError("No es un array (A)")
    assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1)."
    n = A.shape[0]

    for o in range(0, n - 1):
        p = None
        for pi in range(1, n):
            if A[o, pi] == 0:
                continue
            if p is None or abs(A[pi, o]) < abs(A[p, o]):
                p = pi
        if p is None:
            raise ValueError("No existe solución única.")
        if p != o:
            A[[o, p], :] = A[[p, o], :]
            for j in range(o + 1, n):
                A[j, o] = A[j, o] / A[o, o]
                A[j, i:] = A[j, i:] - m * A[o, i:]

        if A[o, o] == 0:
            raise ValueError("No existe solución única.")

    solution = np.zeros(n)
    solution[n - 1] = A[n - 1, n] / A[n - 1, n - 1]

    for i in range(n - 2, -1, -1):
        suma = sum(A[i, j] * solution[j] for j in range(i + 1, n))
        solution[i] = (A[i, n] - suma) / A[i, i]

    end_time = time.time()
    tiempo = end_time - start_time

    return solution, tiempo

# =====
def gauss_jordan(Ab: np.ndarray) -> np.ndarray:
    """
    Gauss-Jordan
    """
    start_time = time.time()

    if not isinstance(Ab, np.ndarray):
        raise ValueError("No es un array (Ab)")
    assert Ab.shape[0] == Ab.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1)."
    n = Ab.shape[0]

    for i in range(0, n):
        p = None
        for pi in range(1, n):
            if Ab[i, pi] == 0:
                continue
            if p is None or abs(Ab[pi, i]) < abs(Ab[p, i]):
                p = pi
        if p is None:
            raise ValueError("No existe solución única.")
        if p != i:
            Ab[[i, p], :] = Ab[[p, i], :]
            for j in range(n):
                if j != p:
                    Ab[j, i] = Ab[j, i] / Ab[p, i]
                    Ab[j, i:] = Ab[j, i:] - m * Ab[p, i:]

        if Ab[p, p] == 0:
            raise ValueError("No existe solución única.")

    solution = np.zeros(n)
    for i in range(n - 1, -1, -1):
        solution[i] = Ab[i, -1] / Ab[i, i]

    end_time = time.time()
    tiempo = end_time - start_time

    return solution, tiempo

# =====
def matriz_aumentada(A: np.ndarray, b: np.ndarray) -> np.ndarray:
    """
    Matriz aumentada
    """
    if not isinstance(A, np.ndarray):
        raise ValueError("No es un array (A)")
    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)
    assert A.shape[0] == b.shape[0], "Las dimensiones de A y b no coinciden."
    return np.hstack((A, b.reshape(-1, 1)))

# =====
# PRUEBA Y GRAFICA COMPARATIVA
# =====
# n de 2 a 100
dimensiones = list(range(2, 200))

tiempos_gauss = []
tiempos_gj = []

for n in dimensiones:
    logging.info(f"Probando con dimensión n = {n}")
    A = np.random.rand(n, n) * 100
    b = np.random.rand(n, 1) * 100

    Ab = matriz_aumentada(A, b.flatten())

    # Eliminación gaussiana
    t_gauss = eliminacion_gaussiana(Ab.copy())
    tiempos_gauss.append(t_gauss)

    # Gauss-Jordan
    t_gj = gauss_jordan(Ab.copy())
    tiempos_gj.append(t_gj)

# ----- Grafica -----
plt.figure(figsize=(12, 7))
plt.plot(dimensiones, tiempos_gauss, marker='o', label="Eliminación Gaussiana")
plt.plot(dimensiones, tiempos_gj, marker='x', label="Gauss-Jordan")
plt.xlabel("Número de variables (n)", fontsize=14)
plt.ylabel("Tiempo de ejecución (segundos)", fontsize=14)
plt.title("Comparación de métodos para resolver sistemas lineales", fontsize=16)
plt.legend(fontsize=12)
plt.grid(True)
plt.show()
```

Comparación de métodos para resolver sistemas lineales

