

# HB. Tema 1 - Operaciones vectoriales y complejidad en tiempo y memoria

Xiomara Caceres

02/01/2025

## Índice

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introducción</b>                        | <b>1</b> |
| <b>2</b> | <b>Código Implementado</b>                 | <b>1</b> |
| 2.1      | Implementación Vectorizada . . . . .       | 1        |
| 2.2      | Implementación en bucles . . . . .         | 2        |
| <b>3</b> | <b>Comparación de tiempo y rendimiento</b> | <b>2</b> |
| 3.1      | Resultados gráficos . . . . .              | 3        |
| <b>4</b> | <b>Conclusiones</b>                        | <b>4</b> |

## 1 Introducción

El problema abordado consiste en comparar dos enfoques para la multiplicación de matrices. Se realizará de dos maneras distintas, con bucles anidados y operaciones vectorizadas, contrastando la eficiencia de ambos algoritmos, los tiempos y uso de memoria.

## 2 Código Implementado

A continuación, se presenta el código utilizado en Python:

### 2.1 Implementación Vectorizada

```
1 import numpy as np
2
3 def multiplicacionMatrizVectorizada(A):
4     matrix = np.dot(A,A)
5     return matrix
```

Listing 1: Multiplicación vectorizada con numpy

```

1 time1 = time.time()
2 tracemalloc.start()
3 A21 = multiplicacionMatrizVectorizada(A)
4 print("Matriz A**2 con funciones preexistentes\n", A21)
5 usedMemory1, _ = tracemalloc.get_traced_memory()
6 tracemalloc.stop()
7 time2 = time.time()

```

Listing 2: Calculo del tiempo y uso de memoria con la multiplicación vectorizada con numpy

## 2.2 Implementación en bucles

```

1 def multiplicacionMatrizBucles(A):
2     matrix = np.zeros((n,n))
3     for i in range(n):
4         for j in range(n):
5             for k in range(n):
6                 matrix[i,j] = matrix[i,j] + A[i,k]*A[k,j]
7     return matrix

```

Listing 3: Multiplicación con bucles

```

1 time3 = time.time()
2 tracemalloc.start()
3 A22 = multiplicacionMatrizBucles(A)
4 print("Matriz A**2 hard-coded\n", A22)
5 usedMemory2, _ = tracemalloc.get_traced_memory()
6 tracemalloc.stop()
7 time4 = time.time()

```

Listing 4: Calculo del tiempo y uso de memoria con la multiplicación con bucles

## 3 Comparación de tiempo y rendimiento

El tiempo de ejecución y el uso de memoria se midieron para matrices cuadradas de tamaños crecientes. Los resultados se graficaron y se observó una mejora significativa en la versión vectorizada. El código es el siguiente:

```

1 print("\nTiempo con funciones preexistentes: ", format(time2-time1))
2 print("\nTiempo con bucles hard-coded: ", format(time4-time3))
3
4 print("\nMemoria con funciones preexistentes: ", usedMemory1)
5 print("\nMemoria con bucles hard-coded: ", usedMemory2)

```

Listing 5: Calculo de tiempo y rendimiento

### 3.1 Resultados gráficos

Generamos los resultados de la siguiente forma:

```
1 def generarGrafica(n, tiempos_bucles, tiempos_numpy,
2   memorias_bucles, memorias_numpy):
3   plt.figure(figsize=(12, 6))
4
5   plt.subplot(1, 2, 1)
6   plt.plot(n, tiempos_bucles, label="Bucles", marker="o")
7   plt.plot(n, tiempos_numpy, label="Vectorizado", marker="o")
8   plt.title("Tiempo de Ejecución")
9   plt.xlabel("Tamaño de la Matriz ({t} x {t}).format(t=n))
10  plt.ylabel("Tiempo (s)")
11  plt.legend()
12
13  plt.subplot(1, 2, 2)
14  plt.plot(n, memorias_bucles, label="Bucles", marker="o")
15  plt.plot(n, memorias_numpy, label="Vectorizado", marker="o")
16  plt.title("Uso de Memoria")
17  plt.xlabel("Tamaño de la Matriz ({t} x {t}).format(t=n))
18  plt.ylabel("Memoria (KB)")
19  plt.legend()
20
21  plt.tight_layout()
22  plt.savefig("comparacion_matrices.pdf")
23  plt.show()
```

Listing 6: Cálculo de tiempo y rendimiento

La gráfica generada muestra la complejidad en el tiempo y memoria de ambos algoritmos:

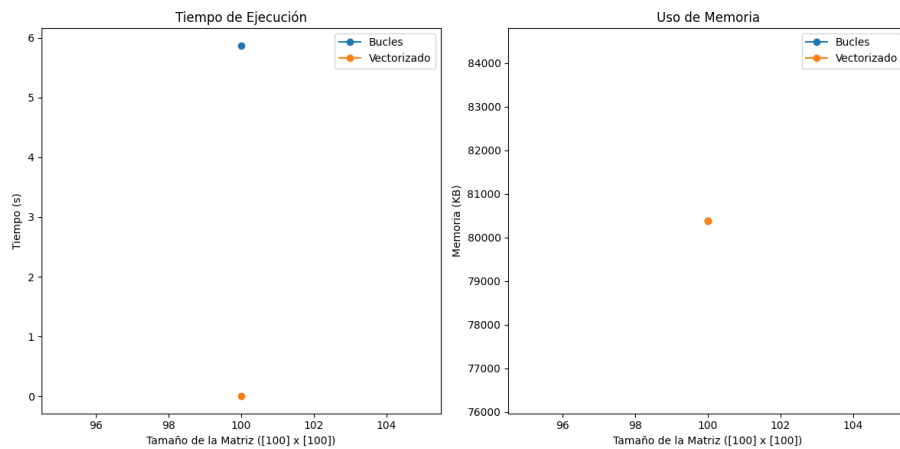


Figure 1: Comparación de tiempos de ejecución y uso de memoria

## 4 Conclusiones

El enfoque vectorizado con `numpy` supera ampliamente a la implementación con bucles en términos de tiempo de ejecución y uso de memoria. Esto refuerza la importancia de utilizar herramientas optimizadas para problemas computacionalmente intensivos.