



 [juanjoseparada](#) / [MCOC2021-P0](#)forked from [jaabell/MCOC2021-P0](#)

&lt; &gt; Code

 Pull requests Actions Projects Wiki Security Insights main ▾

...

[MCOC2021-P0](#) / README.md

juanjoseparada Update README.md

 History 2 contributors

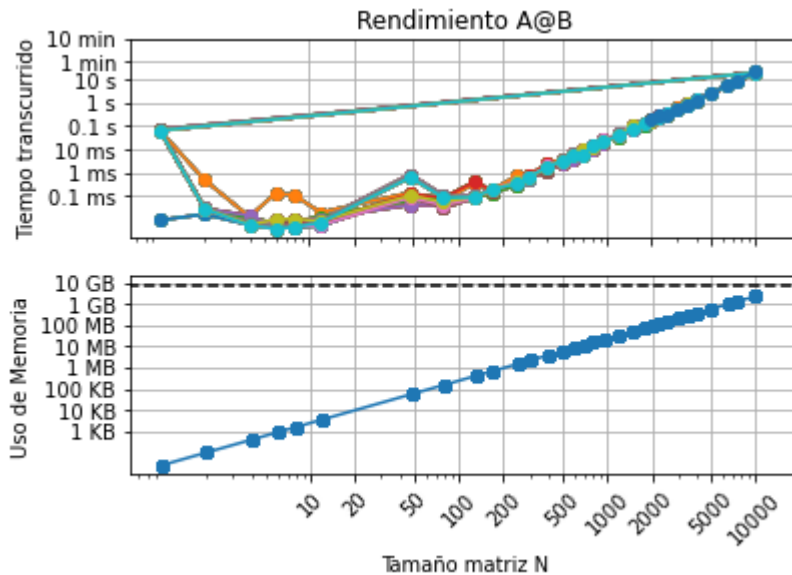
# MCOC2021-P0

## Mi computador principal

- Marca/modelo: VivoBook\_ASUSlaptop X571GT\_X571GT
- Tipo: Notebook
- Año adquisición: 2020
- Procesador:
  - Marca/Modelo: Intel Core i5-8300H
  - Velocidad Base: 2.30 GHz
  - Velocidad Máxima: 2.304 GHz
  - Numero de núcleos: 4
  - Humero de hilos: 2
  - Arquitectura: x86\_64
  - Set de instrucciones: Intel SSE4.1, Intel SSE4.2, Intel AVX2

- Tamaño de las cachés del procesador
  - L1d: 256KB
  - L1i: 256KB
  - L2: 1,0KB
  - L3: 8,0KB
- Memoria
  - Total: 8 GB
  - Tipo memoria: DDR3
  - Velocidad 2667 MHz
  - Numero de (SO)DIMM: 4
- Tarjeta Gráfica
  - Marca / Modelo: Nvidia GeForce GTX 1650
  - Memoria dedicada: 8118 MB
  - Resolución: 1920 x 1080
- Disco 1:
  - Marca: ASUS
  - Tipo: SSD
  - Tamaño: 476,93 GB
  - Particiones: 3
  - Sistema de archivos: EXT4
- Dirección MAC de la tarjeta wifi: 40-EC-99-3C-AB-95
- Dirección IP (Interna, del router): 192.168.0.52
- Dirección IP (Externa, del ISP): 190.160.0.14
- Proveedor internet: VTR Banda Ancha S.A.

Desempeño MATMUL



¿Cómo difiere del gráfico del profesor/ayudante?

Viendo los dos tipos de graficos, se puede ver una gran diferencia en los tiempos transcurridos al ejecutar el programa, esto se puede deber a la diferencia de computadores basicamente y la velocidad de estos para procesar los datos. Se pudo notar un tiempo de 0.1 ms mientras que en el mio se demoraba 0.1 s. Mi grafico tuvo un problema de ultimo segundo que pretendo no incluirlo en mis resultados, el cual unió el último tramo con el primero.

¿A qué se pueden deber las diferencias en cada corrida?

Se puede ver una diferencia en cada corrida debido a varias causas, de las cuales la principal que se me ocurre es el uso de memoria en diferentes actividades del computador, que al sobrepasar un nivel seguirá al siguiente, lo cual puede entorpecer el proceso y con esto generar varación en los tiempos de estos.

El gráfico de uso de memoria es lineal con el tamaño de matriz, pero el de tiempo transcurrido no lo es ¿porqué puede ser?

En el grafico del tiempo transcurrido el comportamiento no es lineal, esto se debe a que al aumentar el tamaño de la matriz, suben la cantidad de operaciones que se estan realizando de forma exponencial, por lo cual el tiempo también sube de forma exponencial. Por otra parte el segundo grafico tiene un comportamiento lineal, ya que al aumentar las dimensiones de las matrices utilizadas también sube la cantidad de memoria necesaria, por lo que se comporta linealmente.

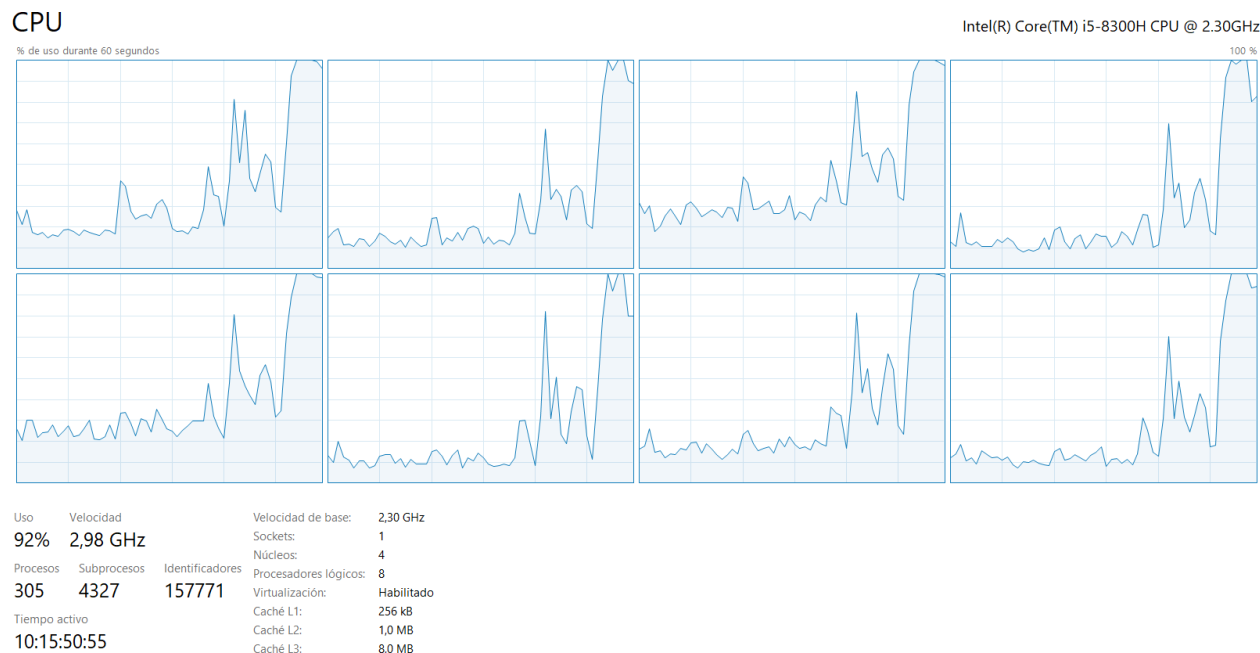
¿Qué versión de python está usando?

R: 3.8.3

¿Qué versión de numpy está usando?

R: 1.18.5

Durante la ejecución de su código ¿se utiliza más de un procesador? Muestre una imagen (screenshot) de su uso de procesador durante alguna corrida para confirmar.



-----DESEMPEÑO DE INV-----

Primero que nada cabe recalcar lo que sucedió en el proceso, donde al trabajar con el caso 1 se notó que los tipos de datos half y longdouble no pudieron ser ejecutados debido a que la librería numpy no los tiene incluidos. Por otro lado con la librería Scipy el único que actuó diferente fue el tipo longdouble, que al principio no corrió debido a que Python trabaja hasta los 64 bits y le estaba pidiendo 128, con lo cual se cambió el "float128" por "longdouble" al momento de importarlo y pudo correr el programa y entregar buenos rendimientos.

También se pudo notar una gran diferencia entre el uso de distintas librerías, donde scipy actuó de forma mucho más rápida que numpy. Y hablando más en específico de la librería scipy se pudo notar que la función `overwrite_a(True)` se ejecutó de mejor manera que todos los otros procesos en los diferentes scripts.

## PREGUNTAS

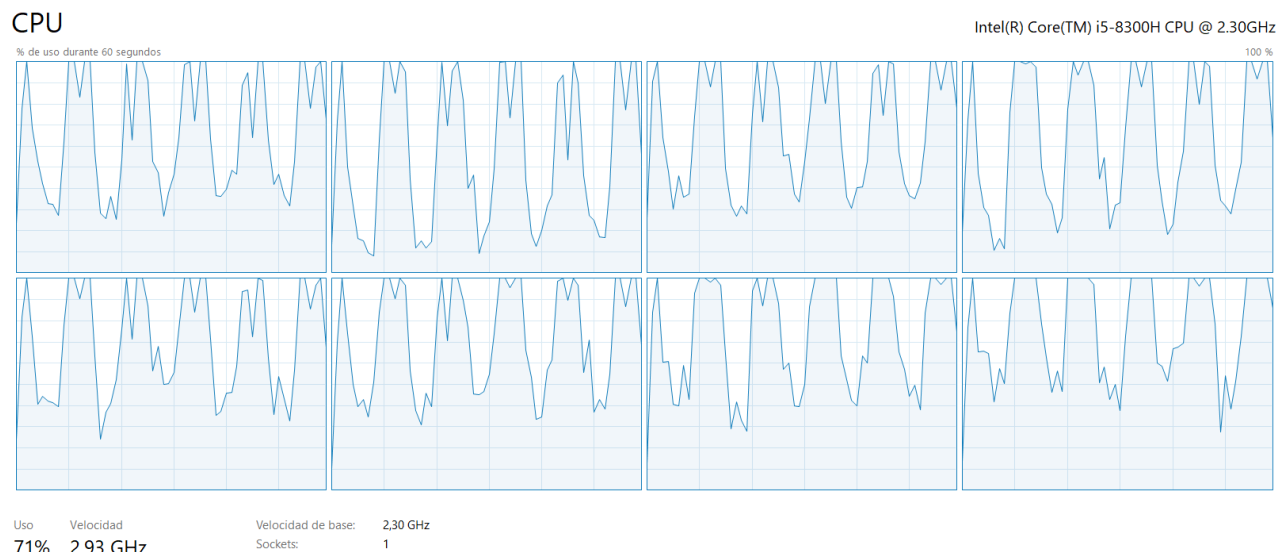
¿Qué algoritmos de inversión cree que utiliza cada método (ver wiki)? Justifique

Librería Numpy: es ejecutado con `numpy.linalg.solve(MAtrizNxN, l)` donde se utiliza la factorización LU (Lapack) mediante solve.

Librería Scipy:ejecutado con `scipy.linalg.inv()`, también utiliza Lapack pero también es optimizado por Atlas y Blas. Debido a esto se demorará menos tiempo en entregar resultados, ya que procesa rápidamente funciones del algebra lineal como las que ocupamos ahora.

¿Como incide el paralelismo y la estructura de caché de su procesador en el desempeño en cada caso? Justifique su comentario en base al uso de procesadores y memoria observado durante las corridas.

En este caso se pudo ver que el paralelismo en el proceso de ejecución actuó de buena manera, ya que se puede ver en la foto presente a continuación como estos se distribuyen el trabajo entre sí logrando tener un funcionamiento mas eficiente al momento de ejecutar el código



≡ 160 lines (92 sloc) | 8.72 KB

0.09.29.39

Caché L3:	8,0 MB
-----------	--------

-----P0E4 Desempeño EIGH y SOLVE-----

Comentarios:

Se pudo notar una subida en los tiempo en las matrices menores a 10. Esto ocurrió a ultimo momento, ya que se encontró el error pero no alcanzaba el tiempo para cambiarlo. Esto ocurre ya que al pasar el archivo de texto a una lista se llama a la función "pop()" la cual en las matrices pequeñas al eliminar el ultimo termino elimina un "elevado a..." lo cual hace que el exponente desaparezca haciendo que el tiempo transcurrido suba a esos valores. Ademas el computador demoro mas de lo esperado en los procesos, teniendo que dejar como tamaño maximo de matriz  $N=3000$ .

Preguntas:

2- La variabilidad del tiempo fue algo que se pudo notar hasta sin ver los graficos y resultados, ya que al momento de ejecutar el metodo Solve fue mucho mas rapido que al ocupar Eigh. Pero al evaluar el caso A, se pudo notar una demora mayor en el caso 1 y en el caso 6, ya que el `overwrite_b` actuó de forma menos eficiente. En el caso B, se pudo notar como el desempeño de Eigh fue notablemente mas lento, notando ademas una gran diferencia entre el caso 4 y 5.

3- En el caso A fue notablemente mas rapido el caso 2 "`assum_a='pos'`" En el caso B fue mas rapido el caso 3 con el `overwrite_a= True`, como se esperaba despues de la entrega anterior

4- Sí, pero también hay que asumir que el computador ocupará el paralelismo, con lo cual no afectaría tanto en el desempeño.

5- Sí, de hecho al ver el rendimiento de estos se pudo ver que actuaban todos al mismo tiempo. (Paralelismo)

6- Se pudo ver un comportamiento similar a la entrega anterior. Pero al correr Eigh se utilizó mas

----- P0E5 Matrices dispersas y  
complejidad computacional-----

Se utilizó la siguiente función para crear la matriz del tipo llena, logrando reutilizar la función ocupada en las entregas anteriores

```
def laplaciana(N, dtype):
    A = zeros((N,N) , dtype=dtype)
    for i in range(N):
        A[i,i] = 2
        for j in range(max(0,i-2),i):
            if abs(i-j) == 1:
                A[i,j] = -1
                A[j,i] = -1
```

```
return(A)
```

y para la la matriz dispersa se ocupó el siguiente codigo con la función "eye" ya que para la matriz del tipo llena no lo pudo soportar por el tipo de dato utilizado

```
return 2*sparse.eye(N,dtype=dtype)-sparse.eye(N,N,1,dtype=dtype)-
sparse.eye(N,N,-1,dtype=dtype)
```

Para esta entrega se ocupó nuevamente la operación MATMUL, donde se buscó poder ver la diferencias de tiempos transcurrido en operaciones de tipo Solve realizadas entre matrices laplacianas llenas y dispersas. Después de realizar lo anterior se pudo notar una gran diferencia en los tiempos transcurridos al ocupar estos diferentes tipos de matrices, donde la matriz dispersa necesito muchisimo menos tiempo que la matriz llena. Esto hizo sentido inmediato ya que la matriz del tipo dispersa contiene menos numeros que la matriz llena, debido a que todos los números ceros son eliminados de esta, con lo cual ocupará mucho menos datos y necesitará realizar menos operaciones. Esto se pude notar facilmente en los graficos obtenidos de tiempo vs tamaño de matriz y mostrados a continuación:

