Python vs C: Heapsort

Álex Herrero Bravo, Walter José Troiani Vargas i Lluc Clavera Comas
Desembre 2021 Q3

1. Resum

Objectiu: L'objectiu d'aquest estudi és comparar el temps d'execució de l'algorisme *Heapsort* en C i en Python. Sospitem (i volem comprovar) que l'algorisme *Heapsort* s'executa més ràpidament en C.

Mètode: Per assolir el nostre objectiu hem executat l'algorisme Heapsort 100 vegades (50 per cada llenguatge) amb els mateixos vectors generats de forma aleatòria (veure apartat 3 tema "Variables emprades" i "Tipus de mostra" per més informació). Al ordenar cada vector amb C i amb Python ens trobem en un cas de mostres aleatòries aparellades.

Resultats: Les dades recollides donen una molt forta evidència (p valor menor a 2.2e-16) que Python executa Heapsort més lentament, amb un interval de confiança del 95% de la mitjana geomètrica del rati (temps de python/ temps de C) de [35.99363, 40.62904]

Conclusió: Hem rebutjat que l'algorisme *Heapsort* ordeni els vectors en el mateix temps en C i en Python, i els resultats indiquen que C és més ràpid que Python en ordenar un vector utilitzant l'algorisme prèviament mencionat. Tot i això, els resultats s'han d'interpretar amb prudència, ja que no podem afirmar amb total seguretat que els resultats compleixin les premisses establertes.

2. Introducció i Objectius

En l'àmbit de la programació hi ha una immensitat de llenguatges diferents: interpretables, procedurals, declaratius, esotèrics… Depenent del llenguatge que s'empri hi poden haver diferències a l'hora de compilació i per això també diferències abismals quan ens referim a l'eficiència.

El nostre objectiu és comparar dos llenguatges de programació molt coneguts i molt usats, C i Python, i comprovar quins dels dos és més ràpid a l'hora d'executar l'algorisme de Heapsort el qual ordena els números d'un vector amb cost asimptòtic $\Theta(n \log n)$, tots dos programats seguint la mateixa pauta (vegeu codi per més informació aquí), executant-lo amb diferents vectors.

Hem escollit l'algorisme "Heapsort", ja que és un algorisme òptim (asimptòticament) respecte a algorismes d'ordenació basats en comparacions per ordenar i volíem utilitzar un algorisme conegut per comparar aquests dos llenguatges.

3. Métodes

Variables emprades

La variable Y serà el temps d'execució del programa mesurat en segons. La nostra variable X serà el llenguatge de programació emprat per endreçar el vector (Python o C). També recollirem una variable Z que serà la longitud (nombre d'elements) del vector.

Per determinar els temps d'execució, hem generat vectors a l'atzar (de mides fixes) i els ordenarem utilitzant l'algorisme heapsort en dos programes diferents, un escrit en C i l'altre escrit en Python.

Per assegurar-nos de diferències mínimes, s'executaran tots en el mateix ordinador en un entorn de sistema operatiu Linux Ubuntu 20.04.3 LTS x86_64, en un processador AMD RYZEN 7 4800H de 8 nuclis (16 threads), a una freqüència màxima de 4.2 GHz amb 3 nivells de cache (On l'últim, L3, té 8 MB).

S'ha de tenir en compte, en cas que es volgués replicar l'experiment, que estem usant la versió 3.8.10 de Python i la versió que usarem per gcc és la 9.3.0. Per compilar el programa en C en la versió C17 hem utilitzat la comanda **gcc -o heapsort.exe heapsort.c.** Per executar els programes hem utilitzat **./heapsort.exe < arrayX.txt** en cas de C i **python3 heapsort.py < arrayX.txt** en cas de Python, on arrayX.txt és el document de text on està guardat el vector que volem ordenar.
Per mesurar el temps hem iniciat un comptador just abans d'entrar a la funció heapsort i l'hem parat just després de sortir de la funció. Hem fet servir la llibreria time.h de C i la llibreria time de Python pels comptadors de temps.

Les implementacions de l'algorisme heapsort que hem implementat nosaltres, estan fortament inspirades en la implementació del capítol 6 del CLRS ("Introduction to algorithms" By Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest i Clifford Stein).

CLRS: LinkCLRS
Link al programa escrit en C: LinkC
Link al programa escrit en Python: LinkPython
Link al script de R usat: LinkR

Per generar els arxius txt amb els vectors a ordenar hem utilitzat el codi següent:

Generador de vectors: Link Generador de números

Tipus de mostres

Per determinar els temps d'execució, hem generat 50 vectors V_i , on cada un té una mida determinada pel nombre d'elements seguint la formula V_i = 10000 + 40000*i ($\forall i \in \mathbb{N} : i = 0, 1, 2 \ldots 49$). Vam escollir generar 50 vectors per tenir suficients mostres i amb una mida que segueix la fórmula anterior escrita perquè a l'hora de tenir la mida

màxima (i = 49) no tenir conflictes amb restriccions de C i Python a l'hora de generar aquests vectors.

Cadascun d'aquests vectors han sigut creats amb el generador de números pseudoaleatoris que implementa el llenguatge C (funció rand() de la <stdlib.h>), després han sigut guardats en fitxers de text i finalment hem donat als nostres programes els vectors que hem generat prèviament per calcular el temps d'execució.

Les dades recollides són "mostres aparellades", ja que cada vector generat el provem dos cops: un per cada llenguatge. Aleshores d'una mateixa mostra tindrem 2 temps, un corresponent a C i l'altre a Python.

Llavors serà convenient i pràctic, per estalviar càlculs crear una variable diferència D, la qual representarà la diferència dels logaritmes entre els temps d'execucions de les 2 llengües. Hem decidit aplicar logaritmes, ja que, com més endavant veurem, semblen tenir una relació multiplicativa, no pas additiva constant.

$$D = ln(P) - ln(C)$$

Procediments i prova d'hipòtesi

En basarem en la variable D per tal de determinar resultats. Partim de la hipòtesi inicial que ambdós llenguatges tenen la mateixa mitjana, amb certa sospita, la qual és plasmada en la nostra hipòtesi alternativa, que en mitjana C trigarà menys, o sigui, la diferència (temps de Python - temps de C) serà positiva (prova de tipus unilateral).

$H_o: \mu_D = 0$

$H_A: \mu_D > 0$ (enfoc unilateral)

Partim de tres premisses per a realitzar el nostre estudi estadístic:

-La variable diferencia D, assumirem que estadísticament s'assembla a una normal
-Són mostres aparellades aleatòries
-La variable diferència té un efecte additiu constant.

Aleshores un estadístic adient donades aquestes condicions és: $t' = \frac{D - \mu_D}{S_D \sqrt{\frac{1}{N_D}}}$

On D és la diferència, S_D és la desviació tipus de la diferència i $N_d = 50$
Aquest estadístic seguirà una distribució semblant a una T-student amb N_d - 1 graus de llibertat, o sigui 49. $t' = t_{49}$

Usarem un interval de confiança del 95% de confiança (o sigui risc $\alpha = 5\%$) per donar un rang amb certa seguretat del valor real poblacional de quina podria ser la diferència de logaritmes.

$$\mu_{real} \in IC(\mu_D, 95\%) = X \pm t_{49,\,0.975} * \sqrt{\frac{S_D}{N_D}}$$ on X és la mitjana mostral

Si |t'| > $t_{49,\,0.95}$, llavors rebutgarem la hipòtesi nul·la

A més, utilitzarem la variable Z per fer un model lineal del temps que triga l'algoritme heapsort escrit en Python en funció de la mida del vector.

Per fer aquesta anàlisi partirem de 4 premisses:
-Linealitat: La relació entre el temps i la mida és lineal
-Homoscedasticitat: mateixa desviació en tots els errors
-Independència: Els errors són independents i un no aporta informació sobre l'altre
-Normalitat: Els errors segueixen una distribució normal

La fórmula del temps que triga Python (Y) en funció de la mida del vector (X, no confondre amb la X de la variable del llenguatge) serà la següent:

$$Y_i = B_0 + B_1 \cdot X_i + \varepsilon_i$$

Per calcular les variables B_0 i B_1 Usarem les fórmules següents:

$$B_1 = \frac{S_{xy}}{S^2_x} \qquad B_0 = Y - B_1 \cdot X$$

On S_{xy} és la covariància entre les 2 variables i S^2_x és la variància de la variable x (mida del vector).

Finalment, farem una prova d'hipòtesi per veure si el pendent és nul (no hi ha relació entre la mida i el temps) o si el pendent és positiu (a major mida, major serà el temps d'execució).

Estadístic que usarem:

$$T = \frac{b_1}{S^2 b1} \quad on \quad Sb1 = \frac{S^2}{S^2 b1} \quad i \; S^2 = \frac{(n-1)(S^2 y - b1 Sxy)}{n-2};$$

$T \sim t_{n-2}$

Si |t'| > $t_{48,\,0.95}$, llavors rebutgarem la hipòtesi nul·la

4. Resultats i Anàlisi estadística

La figura 1 representa una comparació de les dades aparellades mitjançant un gràfic boxplot, que ens informa d'on estan situades les mitjanes i on s'acumulen els quartils Q_{25} i Q_{75}.

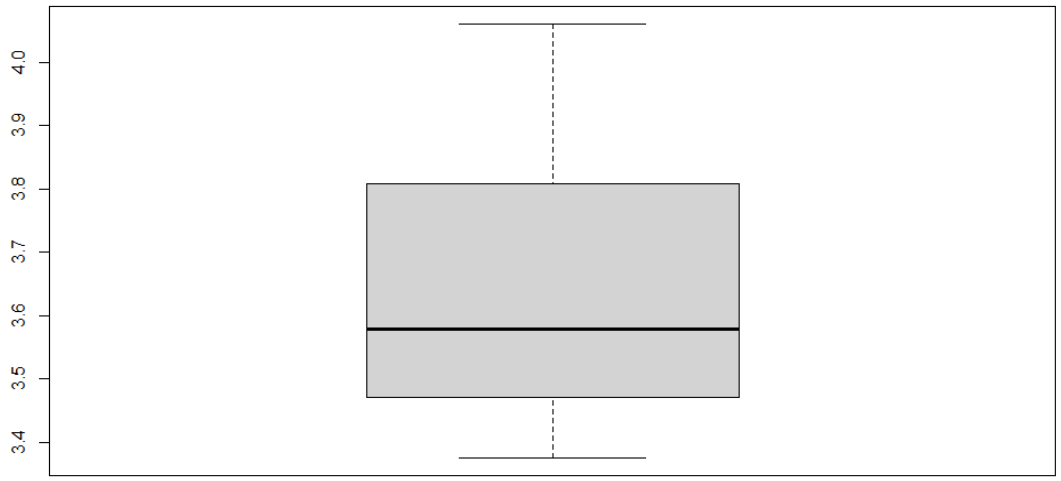

Figura 1: Box Plot de la diferencia dels logaritmes del temps d'execució

Aleshores extraiem la mitjana, mediana i quartils d'aquest gràfic en la següent taula:

Mínim	Quartil 25%	Mediana	Mitjana	Quartil 75%	Màxim
3.375	3.471	3.578	3.644	3.801	4.060

Taula A: Descriptiva de la diferencia de logaritmes

Els gràfics QQnorm i l'histograma ens ajudaran a determinar com és de normal la nostra variable diferència.

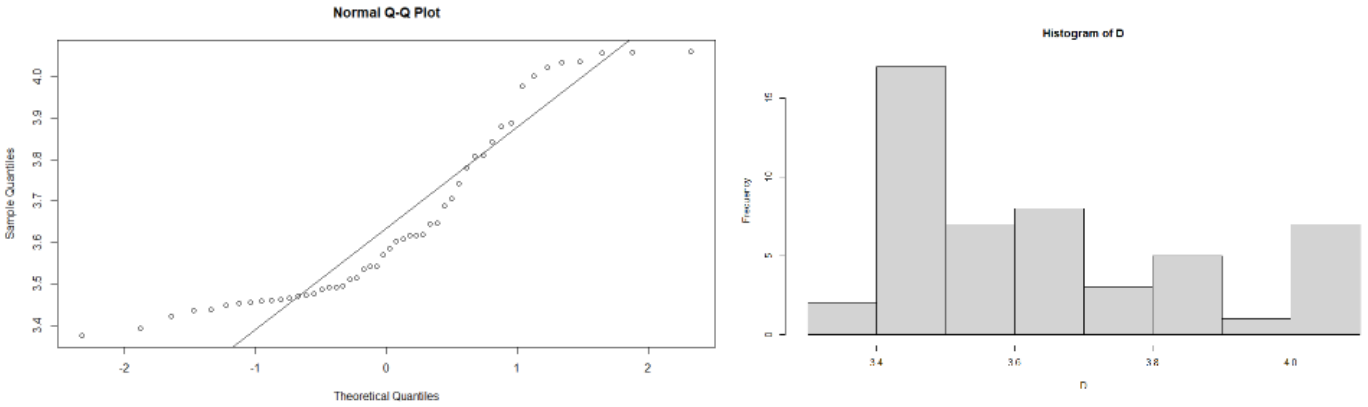

Figura 2: QQnorm i histograma de la diferència de logaritmes

Com es pot observar en el gràfic QQnorm i l'histograma, no queda del tot clar que es compleixi la premissa de normalitat, tot i així, per fer l'estudi estadístic considerarem que sí que es compleix la premissa de normalitat. Per aquest motiu els resultats que surtin d'aquest estudi s'hauran d'interpretar amb cautela, ja que no estem segurs de què les premisses necessàries es compleixin.

A més a més, entre els temps de Python i C hem reproduït un gràfic Bland-Altman, per saber no només usant correlació (pel fet que aquesta només ens diu si estan relacionades) l'interval d'acord entre aquests 2 i si hi ha un efecte additiu constant.

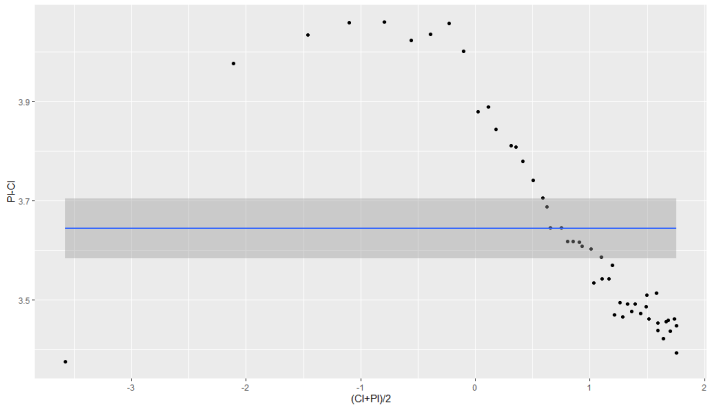

Figura 3: Gràfic Bland-Altman de la diferència de logaritmes

L'interval d'acord és $mitjana \pm 1.96 * S_D$, o sigui [3.226179, 4.061646], el qual és suficient per abastar tots els punts de la diferència, ja que el menor de tots és 3.375 i 4.060 el més gran. Aleshores increïblement no tenim cap punt fora de l'interval. Podem concloure llavors que hi ha acord entre les dues dades, però com hi ha una òbvia tendència decreixent, no podem afirmar que es compleixi l'efecte additiu, com en el cas de la normalitat, considerem que l'efecte esmentat sí que es compleix, encara que això provoqui que els resultats s'hagin d'interpretar amb precaució.

Càlculs

Fent els càlculs adients (de les fórmules del apartat 3, secció “Procediment i proves d’hipòtesis”, usant l’script de R) ens han sortit els resultats següents:

-mitja D ($\mathbb{D}$): 3.643913

-Desviació estàndard de D (S_D): 0.2131291

-estadístic T: 120.8955
-punt critic: 1.67655
-pValor: < 2.2e-16

-IC(μ_D, 95%) = [3.583342, 3.704483]

-IC(μ_D, 99.9%) = [3.538406, 3.749149]

A pesar que només és necessari per càlculs auxiliars i podria ser omès mostrem el valor de l'interval amb 99.9% de confiança, el qual també mostra que el temps d'execució de Python és força major.

Utilitzant la funció exponencial sobre els intervals de confiança podem obtenir un interval per la ràtio entre el temps d’execució dels dos llenguatges:

IC(95%) = [35.99363, 40.62904]

Aquest interval ens indica que, amb un 95% de confiança, C és entre 35.99 i 40.63 vegades més ràpid que Python en executar l’algorisme *Heapsort*.

IC(99.9%) = [34.41201, 42.49640]

Si provem a fer un t-test en R en surt el següent output (considerant que es unilateral):

```
        Paired t-test

data:  log(Pt) and log(Ct)
t = 120.9, df = 49, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 3.59338     Inf
sample estimates:
mean of the differences
               3.643913
```

Figura 4: Sortida de R del t-test

5. Regressió lineal:

Discussió de les premisses:

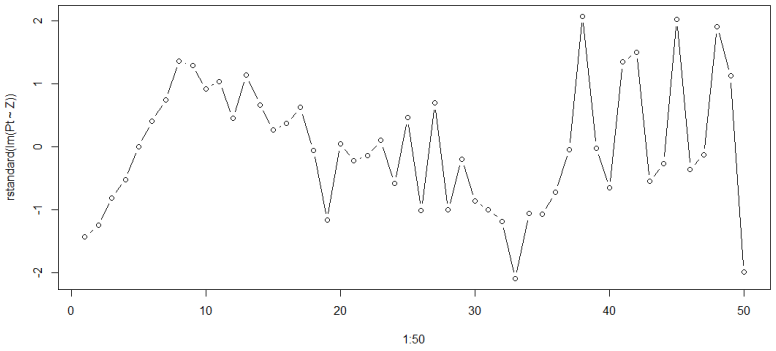

Figura 4: Residus estandaritzats

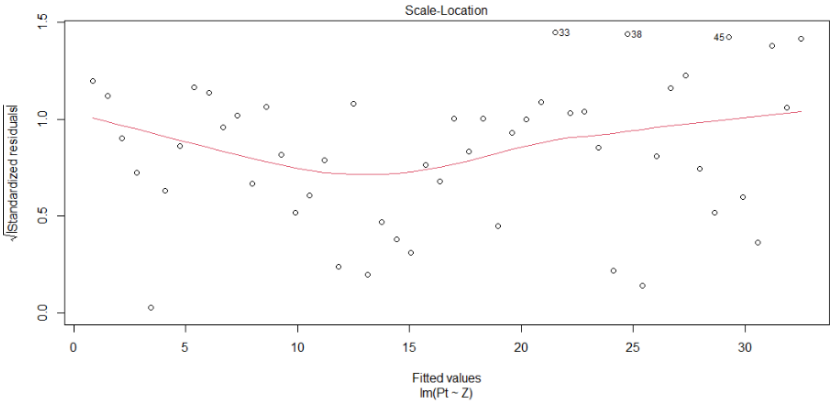

Figura 5: taula de comparacio Residuals Vs FittedValues

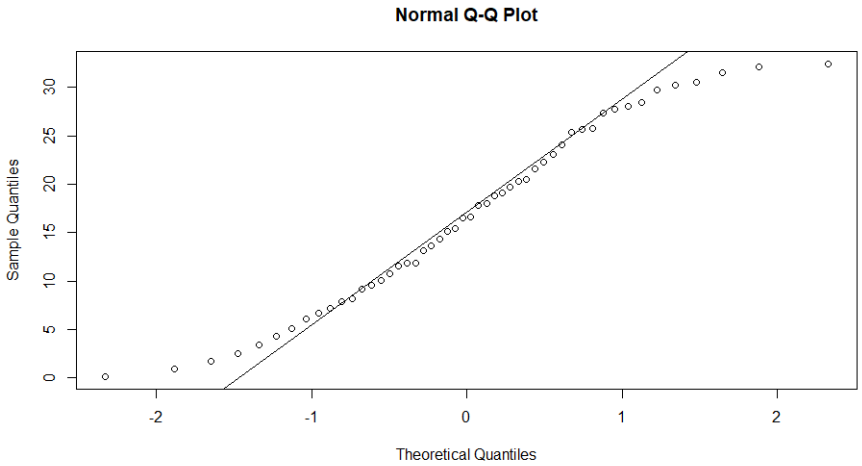

Figura 6: taula QQnorm pels residus

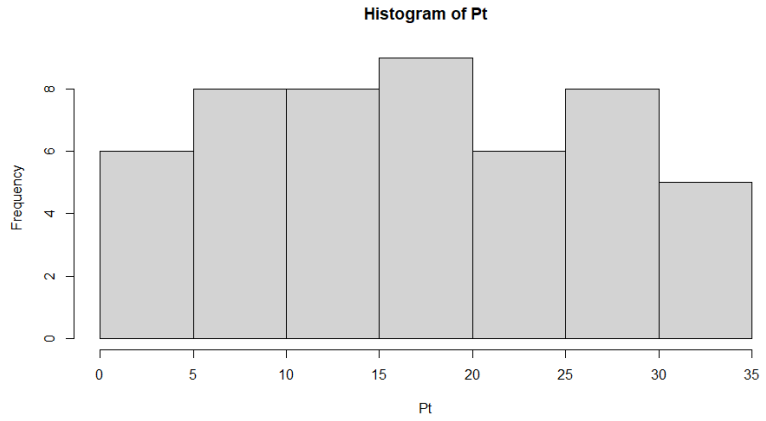

Figura 6: taula QQnorm pels residus

D'aquests gràfics podem extreure conclusions respecte a les 4 premisses que havíem mencionat inicialment per a poder fer l'anàlisi.

-Normalitat: A pesar que tant per quartils com per histograma s'assembla molt més a una distribució normal, aquesta no és del tot fiable com per donar per vàlida la premissa de normalitat, però els resultats seran molt més precisos que no pas a l'anterior prova d'hipòtesi.

-Linealitat: Si ens fixem en la figura 5, podem observar com els residus gairebé sempre estan a la mateixa distància de la línia roja, aleshores podríem dir que es verifica.

-Homoscedasticitat: Com que la variabilitat dels residus a la figura 5, es manté pràcticament constant aleshores es verifica aquesta propietat.

-Independència: Considerant la figura 4 dels residus estandarditzats, no hi ha manera de trobar un patró que segueixin aquests residus, ja que a pesar que tots fan aquesta forma de serra, en alguns punts l'altura de la serra és molt més alta que en altres i a vegades s'interromp aquest possible patró. Aleshores podríem dir que es verifica la independència dels residus.

Càlculs

Fent els càlculs adients per b_0 i b_1 (resultats de les fórmules de la pàgina 4, usant l'script en R) ens han sortit els resultats següents:

$S_x^2 =$ 3.4e+11
$S_y^2 = 88.94222$
$S_x = 583095.2$
$S_{xy} = 5491238$
$r_{xy} = 0.9985662$
$R^2 = 0.9971344$

b_1 = 1.61507e-05
b_0 = 0.6910594
Sb_1^2 = 1.561744e-14
Sb_0^2 = 0.02051039
S^2 = 0.2601866

Per tant, la fórmula que hem trobat per aproximar el temps d'execució de l'algorisme en funció de la mida del vector és la següent:

$$Y_i = 0.6910594 + (1.61507e - 05) \cdot X_i + \varepsilon_i$$

On $\varepsilon_i \sim N(0, \sigma)$ i l'estimador de σ és $S = \sqrt{S^2} = 0.5100849$

L'interval de confiança del 95% per els estimadors b1 i b0 són:

$IC(B_1, 95\%)$ = [1.589943e-05, 1.640197e-05]

$IC(B_0, 95\%)$ = [0.4031074, 0.9790115]

Per provar que el pendent és realment creixent hem calculat el valor de l'estadístic T amb la fórmula mencionada prèviament. Aquest valor ens ha donat: 129.2369, per una altra banda, el punt crític ens ha donat 1.677224 i el p-valor ens ha donat menys de 2.2e-16

6.Conclusions i discussió

Després d'observar l'anàlisi feta a l'apartat 5 és fàcil adonar-se, tant per criteri de punt crític com el del p-valor que la hipòtesi nul·la és enormement improbable. Aleshores, a no ser que sigui una anomalia estadística extremadament gran, es descarta completament pensar que les mitjanes de temps d'execució de C i Python són iguals.

També es pot veure que el temps d'execució de Python ha de ser major, tal com havíem cregut en la nostra hipòtesi alternativa ($\mu_D > 0$), això ho podem comprovar fàcilment mirant l'interval de confiança (IC(μ_D, 95%) = [3.583342, 3.704483]) on es veu que a part de ser positiu, el temps que triga Python és força major a C.

A més, també hem confirmat un efecte lineal entre la mida del vector i el temps d'execució de Python (encara que en el cas de C també es podria apreciar l'efecte lineal). Aquest efecte és un efecte creixent; això últim ho hem comprovat fent una prova d'hipòtesi sobre el pendent.

Igualment, els resultats s'han d'interpretar amb prudència, ja que la premissa de normalitat no es compleix del tot, ni tampoc ho fa la de l'efecte additiu constant, perquè es veu certa tendència decreixent en el gràfic Bland-Altman fins i tot després de fer una transformació logarítmica.

Un cop acabat aquest projecte hem notat que podríem haver millorat certs aspectes sobre l'estudi. La següent és una llista d'aspectes a millorar:

-Podríem haver fet un estudi previ de la *caché* perquè a l'hora d'executar els algorismes en C/Python per tal que aquesta s'hagués comportat de la mateixa manera en ambdues execucions.

-Els programes es van executar mentre hi havia altres programes en ús (navegador, explorador d'arxius, Rstudio…) per la qual cosa no sabem fins a quin punt el *Kernel* va entrar en ús alentint les execucions

-Les execucions NO es van executar en el màxim rendiment del portàtil, ja que aquest no estava endollat al corrent i s'alenteix la freqüència del processador en aquest tipus de dispositius quan això passa. Això pot haver influenciat en el temps d'execució (per motius interns del *Kernel*) i haver donat a Python, probablement pel fet que és interpretat, desavantatge. Aleshores per un altre intent miraríem d'endollar el portàtil o bé executar-ho en un sobretaula més potent i estable.

Finalment, i com a conclusió d'aquest estudi, és bastant probable que C també tingui un temps d'execució menor a l'hora d'executar altres algorismes, i, per tant, és recomanable utilitzar C abans que Python quan ens interessi tenir temps d'execució més ràpids.