# GENERACIÓ DE FRACTALS EN GIMP

Bernat Borràs Civil

Miquel Torner Viñals

Grup 13

Quadrimestre de Tardor 2021

# Índex

# Resum


## Objectiu

L'objectiu és comparar la velocitat de renderització de fractals en un sistema operatiu basat en Linux envers un en Windows.

## Hipòtesis

Volem veure si el temps de renderització és menor en un sistema basat en Linux respecte un en Windows.

## Metodologia

De forma pseudoaleatòria es seleccionaran 10 resolucions diferents en les que hi generarem un total de 10 vegades un fractal IFS en cadascun dels sistemes operatius.

Amb l'ajuda d'un script obtindrem el temps per cadascuna de les execucions.

## Resultats

Rebutgem la hipòtesis nul·la de igualtats de mitjanes a favor de l'alternativa de $\mu$ t Ubuntu $<$ $\mu$ t Windows. Trobem l'interval de confiança del 95%, que és [-0.3158, -0.0639]. Finalment fem un model de predicció i el validem amb les premisses necessàries.

## Conclusió

Hem tingut limitacions a la hora de recollir les dades amb el nostre script, i aquest hauria d'estar integrat al GIMP per reduir el temps que l'script triga a processar.

# Introducció i objectius

Les dades que volem estudiar són el temps de renderització d'una imatge d'un fractal IFS (sistema de funcions iterades) (concretament d'un triangle de Sierpinski) emprant GIMP 2.10 en dos sistemes operatius, Ubuntu 21.10 LTS com a representant de Linux i Windows 10 21H1.

Havent seleccionat els dos sistemes operatius esmentats renderitzarem diverses imatges de fractals amb ambdós sistemes i intentarem rebutjar la hipòtesi nul·la d'igualtat de temps a favor d'una hipòtesi alternativa unilateral de que renderitzar una imatge amb GIMP en un sistema operatiu basat en Linux és més ràpid que renderitzar-la en un sistema operatiu Windows.

# Metodologia

Per tal d'obtenir una mostra aleatòria simple de la població de les resolucions a ser renderitzades, hem emprat un script[1] el qual ens ha generat 10 resolucions de forma pseudoaleatòria amb relacions d'aspecte entre 4:3 i 16:9 i resolució entre definició estàndard (360x480 píxels per la relació 4:3) i 4K (2160x3840 píxels per la relació 16:9).

Per tal de cronometrar el temps de renderització hem fet ús d'un segon script[2] el qual donat una sèrie de resolucions és capaç d'anar generant automàticament cadascuna de les resolucions donades i mesurar l'interval de temps entre l'ordre de renderització del fractal fins que aquest apareix en pantalla.

Tot plegat s'ha realitzat en un mateix ordinador amb un processador Intel_Core_i5-10210U a 2.1_GHz (gràfica integrada) i 16_GB de memòria RAM.

## Mostres aparellades

Els temps de renderització de fractals en Ubuntu i Windows són aparellats ja que, encara que les resolucions han estat seleccionades de forma pseudoaleatòria, aquestes s'han utilitzat en ambdós sistemes operatius per obtenir-ne el temps de renderització i així poder-los comparar.

## Prova d'hipòtesis

Pretenem comprovar si el temps de renderització és menor en Ubuntu respecte Android, per tant hem proposat la següent prova d'hipòtesis:

$$\begin{cases} H_0 : \mu_{t_{Ubuntu}} = \mu_{t_{Windows}} \\ H_1 : \mu_{t_{Ubuntu}} < \mu_{t_{Windows}} \end{cases}$$

En cas de rebutjar $H_0$ comprovaríem que la nostre suposició inicial es certa. La PH és unilateral ja que busquem comprovar si el temps de renderització de fractals IFS és menor en Ubuntu envers Windows, és a dir, volem assegurar que si es rebutja $H_0$ s'acceptarà que els temps no són iguals i per tant que $H_1$ sigui cert sota la nostra condició (el temps de renderització és menor en sistemes operatius basats en Linux envers Windows).

[1] Veure annex (random_pairs_generator.py)
[2] Veure annex (script_gimp.py)

## Estadístic i distribució

| | | | |
|---|---|---|---|
| $\mu$ | $H_0 : \mu_D = \mu_0$<br>$(\mu_D = \mu_1 - \mu_2)$ | $T = \frac{\overline{D} - \mu_0}{S_D/\sqrt{n}}$ | |
| D ~ N<br>m.a. aparellada | | $T \sim t_{n-1}$ | Rebutjar $H_0$ si<br>$\lvert T \rvert > t_{n-1,1-\alpha/2}$ |

## Interval de confiança

Aplicarem un interval de confiança del 95% sobre la diferència de mitjanes. La desviació tipus és desconeguda, i la n és pròxima a 100.

| | | |
|---|---|---|
| $n \geq \approx 100$ | $Z \sim N(0,1)$ | $\mu \in (\overline{X} \pm z_{1-\alpha/2}\sqrt{\frac{S^2}{n}})$ |

# Resultats

## Descriptiva

Les dades obtingudes s'han representat en aquest gràfic:

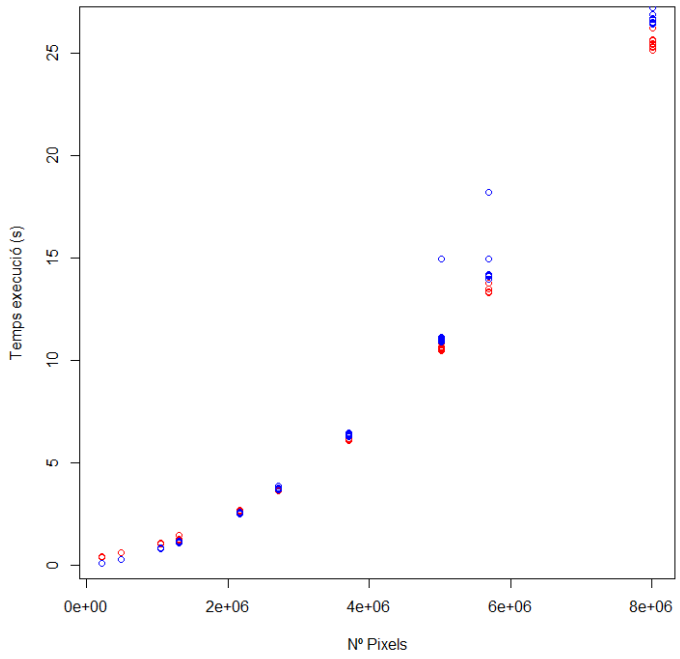


Figura 1. Temps d'execució de les proves (en vermell Ubuntu i en blau Windows)

Com es pot veure a la gràfica, hi ha punts en que s'alineen punts, això es dona ja que per cadascuna de les resolucions que s'han escollit s'han executat diverses proves tant per Ubuntu com per Windows.

Aquesta taula conté la descriptiva dels temps de renderització en segons per Ubuntu.

| Mínim | 1r quartil | Mediana | 3r quartil | Màxim | Mitjana | Desviació Estàndard |
|---|---|---|---|---|---|---|
| 0.383 | 1.024 | 3.172 | 10.532 | 26.213 | 6.551 | 7.665 |

Aquesta taula conté la descriptiva dels temps de renderització en segons per Windows.

| Mínim | 1r quartil | Mediana | 3r quartil | Màxim | Mitjana | Desviació Estàndard |
|---|---|---|---|---|---|---|
| 0.064 | 0.804 | 3.118 | 10.978 | 27.224 | 6.740 | 8.173 |

Aquesta taula conté la descriptiva de les diferències de temps de renderització d'Ubuntu envers Windows.

| Mínim | 1r quartil | Mediana | 3r quartil | Màxim | Mitjana | Desviació Estàndard |
|---|---|---|---|---|---|---|
| -0.346 | 0.246 | -0.081 | 0.371 | 4.713 | 0.190 | 0.766 |

També podem observar i comparar més clarament la descriptiva de Ubuntu i Windows en el següent gràfic:

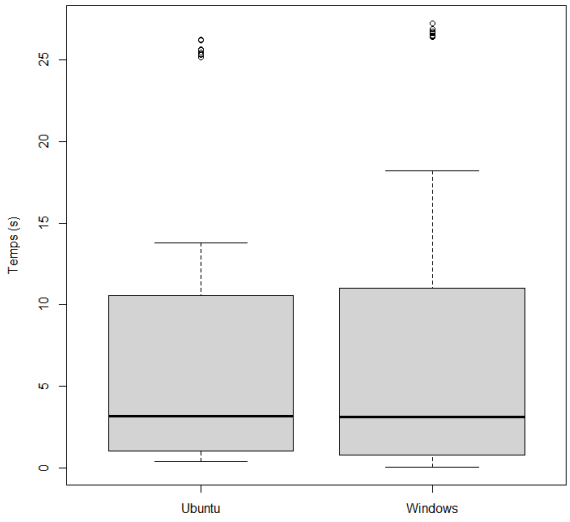


Figura 2. Boxplot del temps d'execució de les proves

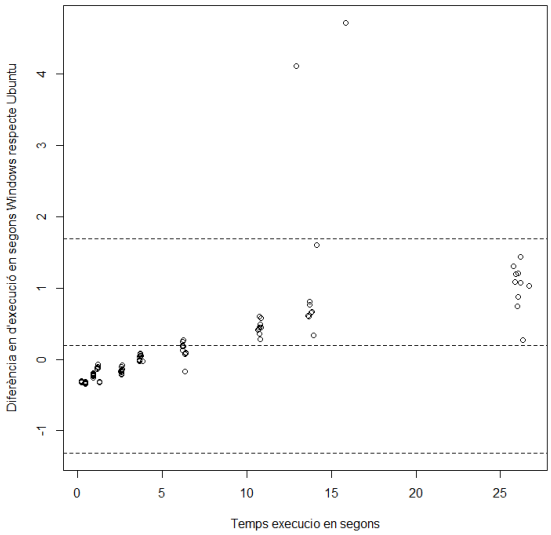


Figura 3. Gràfic Bland-Altman

En la figura 2 podem observar que tant en Ubuntu i en Windows la mediana i la mitjana són molt similars. Una diferència notòria entre aquests dos és que en Windows, la desviació i la variança és major que en Ubuntu. Es pot comprovar mirant la mida de les caixes, que en el cas de Windows sobresurt per dalt i per baix de la d'Ubuntu. També podem veure que els vigotis del gràfic de Windows s'estenen més que el gràfic d'Ubuntu.

També s'inclou, donat que les nostres dades són aparellades, un gràfic Bland-Altman (figura 3), el qual ens permet comparar les diferències entre cadascuna de les execucions en Windows respecte Ubuntu.

Hem obtingut un estadístic de -2.4796 i un pvalor de 0.1272. El punt crític obtingut és de 1.9842. Amb aquests valors podem rebutjar la hipòtesis nul·la de igualtats de mitjanes: $|T|>t_{n-1,\ 1-a}$, 2.4796>1.9842. Tenim suficients evidències de rebutjar la hipòtesis nul·la a favor de la hipòtesis alternativa.

També hem calculat un interval de confiança del 95% sobra la diferència de mitjanes. Els valors obtinguts de l'interval són [-0.3158, -0.0639].

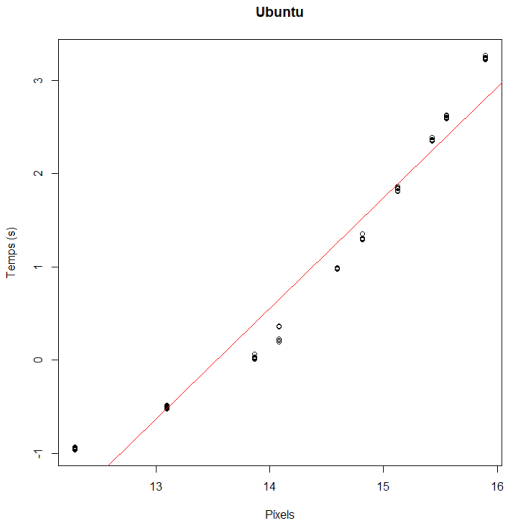


Figura 6. Recta de regressió Ubuntu amb transformació logarítmica

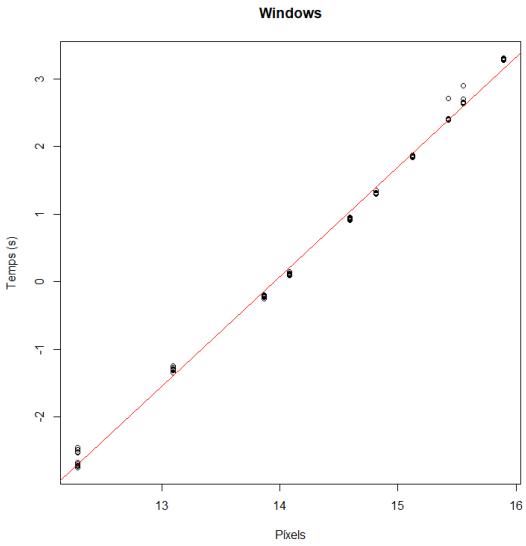


Figura 7. Recta de regressió Windows amb transformació logarítmica

Donada la naturalesa de les nostres dades (estem treballant amb imatges bidimensionals) hem aplicat una transformació logarítmica a les nostres dades per tal de poder complir la premissa de la linealitat a l'hora de validar el nostre estudi. En les figures 6 i 7 podem veure les rectes de regressió conjuntament amb les dades una vegada aplicats els logaritmes en el dos casos. Les rectes sí que contenen termes independents ja que tenim temps menors a 1, el que resulta tenir logaritmes negatius. El pendent ascendent de les rectes de regressió ens indiquen que hi ha una relació directament proporcionals entre el número de píxels, i el temps que triga a renderitzar-los. Com més píxels, més estona triga.

Per validar el nostre estudi hem fet una sèrie de comprovacions de les següents premisses: linealitat, homoscedasticitat, independència i normalitat.

En les figures 8 i 9 podem veure que els residus s'aproxima un patró lineal. La recta és força horitzontal, validant la premissa de linealitat. També veiem que la variància es manté constant en tot el gràfic, i podem validar la premissa d'homoscedasticitat.

En les figures 10 i 11 veiem que la premissa de la independència no s'arriba a complir. Veiem que es segueix un patró on els residus es van reduint de mica en mica, i arriba a un punt on els residus van augmentat.

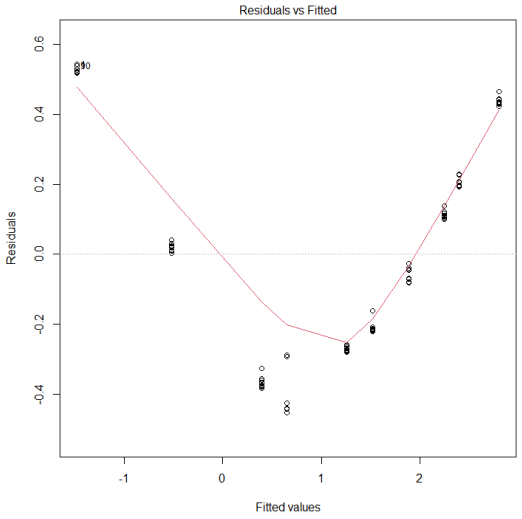


Figura 8. Residuals vs fitted per Ubuntu amb transformació logarítmica

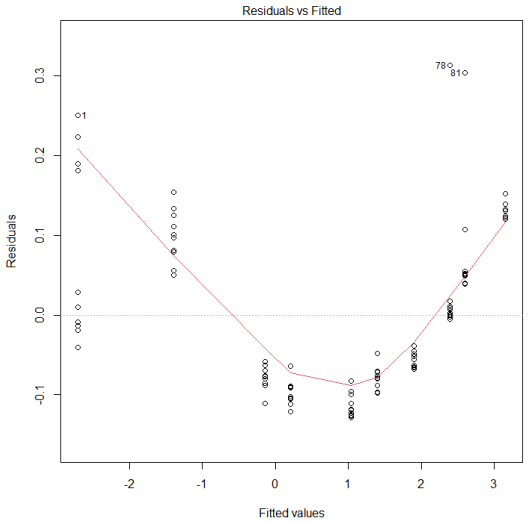


Figura 9. Residuals vs fitted per Windows amb transformació logarítmica

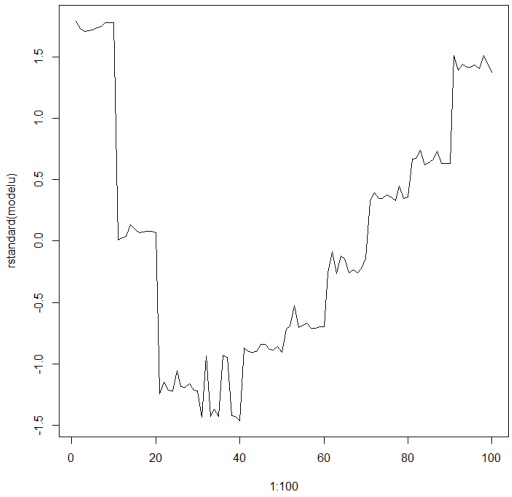


Figura 10. Ordre dels residus per Ubuntu amb transformació logarítmica

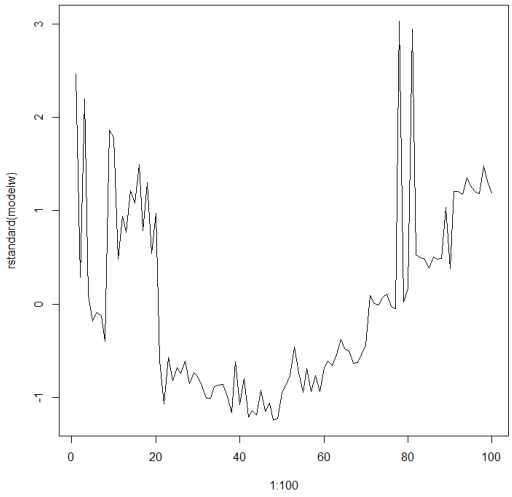


Figura 11. Ordre dels residus per Windows amb transformació logarítmica

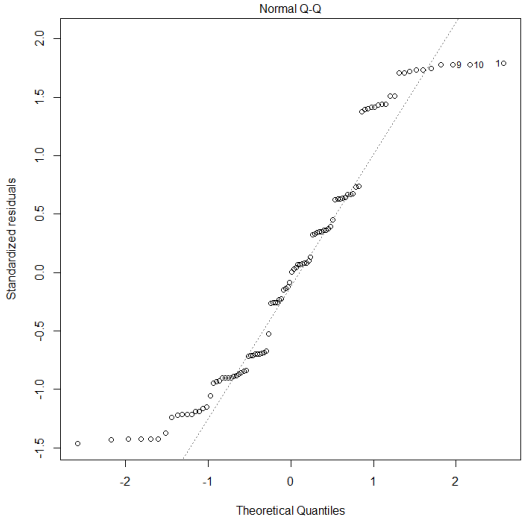


Figura 12. Normal QQ per Ubuntu amb transformació logarítmica

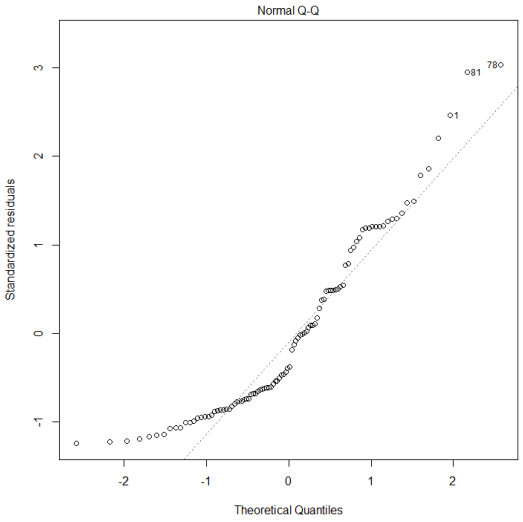


Figura 13. Normal QQ per Windows amb transformació logarítmica

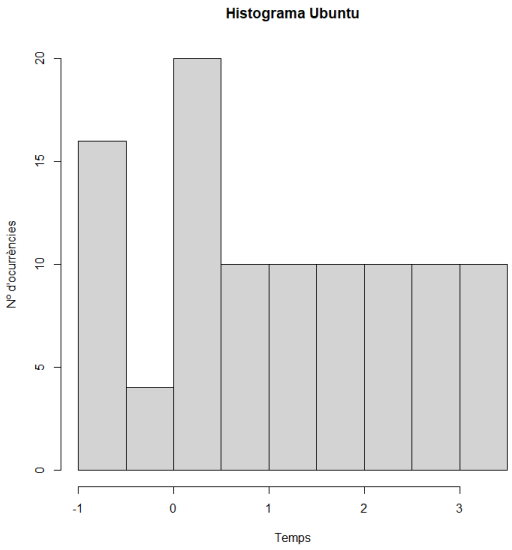


Figura 14. Histograma per Ubuntu amb transformació logarítmica

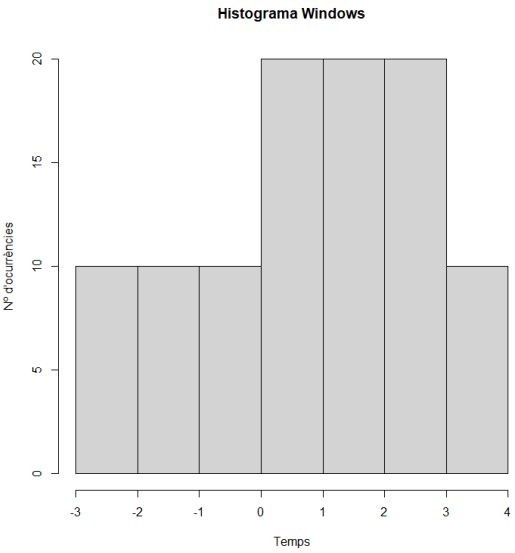


Figura 15. Histograma per Windows amb transformació logarítmica

Amb els gràfics de qqnorm (figures 12 i 13) i els histogrames (figures 14 i 15) podem validar la premissa de normalitat. En els gràfics de qqnorm podem veure que els residus es cenyeixen sobre la recta. En els histogrames de residus, tot i que no són tan fiables que

en els de qqnorm, podem veure que els residus fan una mica de forma de campana (clarament més a la de Windows que no pas a la d'Ubuntu).

Al tenir poques dades, és difícil avaluar les premisses. Tot i així no tenim suficients proves per rebutjar cap de les quatre premisses.

# Discussió

## Conclusions

En la primera part de l'estudi hem comprovat que tenim suficients evidències per rebutjar la hipòtesis nul·la d'igualtat de mitjanes a favor de la hipòtesis alternativa. També hem pogut trobar un interval de confiança del 95% per a les nostres dades.

En la segona part hem hagut d'aplicar logaritmes a les nostres dades per poder validar les quatre premisses, i així acceptar el nostre model de predicció.

## Limitacions

Tot i que hem utilitzat el mateix hardware per a realitzar les proves, hi ha factors que possiblement influeixen als resultats del nostre estudi, no hi ha dubte que un d'ells l'execució de l'script que mesura el temps de cada renderització. Aquest s'executa de forma concurrent fet que pot provocar una desviació en cadascun dels resultats obtinguts, doncs depèn de cada operatiu com gestionar els recursos que es donen a aquest script mentre s'executa la renderització.

## Consideracions

Una de les coses que s'haurien de millor d'aquest treball és l'obtenció de les dades, concretament augmentar-ne la variabilitat, doncs mentre el que s'ha fet és realitzar la mateixa prova diverses vegades per cada resolució, s'hauria d'haver fet una sola prova per cada resolució i augmentar la diversitat d'aquestes. D'aquesta forma s'hauria aconseguit una millor mostra.

A part tenint en compte que Gimp és un programa de codi obert, es podria editar aquest perquè el mateix programa fos l'encarregat d'indicar el temps que ha passat entre l'inici i el final de la renderització, fet que ajudaria a obtenir dades més fiables, és a dir amb menys desviació respecte la seva mitjana.

# Annex 1: random_pairs_generator.py

```python
import random

# Introduce here the number of elements you want to generate
n = 10
# Introduce the limits ratio
min = 4/3
max = 16/9
# Introduce the min and max amount of pixels for y axis
y_min = 360
y_max = 2160
x_min = y_min*min
x_max = y_max*max
# Indicate the minium diference between each y value, note that it
# must be less than (y_max-ymin)/n
diff = 150

def aproves(y_list, y):
    for i in y_list:
        if(abs(y-i) < diff): return False
    return True

def main():
    s = '['
    i = 0
    y_list = [0]
    while i < n:
        x = random.randint(x_min, x_max)
        y = random.randint(y_min, y_max)
        if (x/y >= min and x/y <= max):
            if ( aproves(y_list, y) ):
                y_list.append(y)
                s = s + '[' + str(x) + ', ' + str(y) + '], '
                i += 1

    s = s[:len(s)-2] + ']'
    print(s)

if __name__ == "__main__":
    main()
```

# Annex 2: script_gimp.py

```
# To run this script you'll have to install the following dependencies:
# $ pip3 install pyautogui
#
# Note: it may also ask you to install:
# $ sudo apt-get install python3-tk python3-dev
# $ sudo apt-get install scrot
#
# This program is intendeed to work on a FullHD screen
#

import time
import os
import pyautogui

# Put the ratios (in pixels it must be a ratio between 4/3 and 16/9)
ratios = [[539, 403], [809, 601], [1330, 792], [1378, 950], [1797, 1209],
 [1910, 1420], [2267, 1638], [2763, 1817], [2888, 1971], [3775, 2124]]

# Indicate the number of generatios you want for each ratio
n = 10

def open_gimp():
    time.sleep(1)
    pyautogui.press('win')
    time.sleep(0.3)
    pyautogui.click()
    time.sleep(0.1)
    pyautogui.write('gimp', interval=0.25)
    time.sleep(3)
    pyautogui.press('enter')
    time.sleep(3)

def changes():
    pix = pyautogui.pixel()

def main():
    print("Starting...")
    #Pixel that we'll take to check if the processing has finished
    temp_x = 560
    temp_y = 388
    ratios.sort(key=lambda y: y[0]*y[1])
    #ratios = ratios[8:]
    print(ratios)
    pyautogui.moveTo(temp_x, temp_y, duration=1)
    time.sleep(5)
    open_gimp()
    pyautogui.click()
    path = os.getcwd()
    with open(path+'/'+'resultats.txt', 'a') as f:
        for [x, y] in ratios:
            n_pixels = x*y
```

```
f.write("Result for ratio (" + str(x) + " " + str(y) +
 " ("+ str(n_pixels) +")) : ")
for i in range (0, n):
    ### Creating new file with size x y
    pyautogui.hotkey('ctrl', 'n')
    time.sleep(0.2)
    print('New file with size', x, y)
    pyautogui.press('tab')
    time.sleep(0.2)
    pyautogui.write(str(y), interval=0.2)
    pyautogui.hotkey('shift', 'tab')
    pyautogui.write(str(x), interval=0.2)
    pyautogui.press('enter')
    ### Fit image
    time.sleep(0.2)
    pyautogui.hotkey('shift', 'ctrl', 'j')
    time.sleep(0.2)
    ### Obtain a pixels to later compare its content
    pixels = (pyautogui.pixel(temp_x, temp_y))
    pixels_new = pixels
    ### Generate a IFS Fractal
    time.sleep(0.3)
    pyautogui.hotkey('alt', 'r')
    time.sleep(0.2)
    pyautogui.press('r')
    time.sleep(0.2)
    pyautogui.press('f')
    time.sleep(0.2)
    pyautogui.press('i')
    time.sleep(1)
    pyautogui.hotkey('alt', 'r')
    time.sleep(1)
    pyautogui.press('enter')
    time.sleep(1)
    pyautogui.press(['right'])
    time.sleep(1)
    pyautogui.press('enter')
    t_before = time.perf_counter()
    while(pixels == pixels_new):
        #print(pixels)
        pixels = (pyautogui.pixel(temp_x, temp_y))
    t_total = time.perf_counter() - t_before
    print("Time to generate:", t_total)
    time.sleep(2)
    f.write(str(t_total))
    if(i != n-1): f.write(', ')
    time.sleep(2)
    pyautogui.hotkey('ctrl', 'w')
```

```
                time.sleep(1)
                pyautogui.hotkey('alt', 'd')
                time.sleep(0.5)
            f.write('\n')
        pyautogui.hotkey('alt'+'f4')


if __name__ == "__main__":
    main()
```

# Annex 3: script R

```
tu = c(0.3928104650003661, 0.38613866200103075, 0.38298730200040154, 0.3836100400003488, 0.38446171800023876,
0.3864857769985974 3, 0.3880880670003535, 0.39126112300073146, 0.3913192039981368, 0.39143634799984284,
0.5941463680028392, 0.5978194109993638, 0.6000958990007348, 0.6174266479974904, 0.6114764659978391,
0.6052742159990885, 0.6063774850008485, 0.6076591560013185, 0.6072347629997239, 0.605932140002551,
1.0092323099997884, 1.0377492130028259, 1.018105574999936, 1.0168260419995931, 1.0700137429994356,
1.0265347250024206, 1.0257282899983693, 1.0344504360000428, 1.0186306119976507, 1.0152296449996356,
1.2308972630016797, 1.4365047570026945, 1.2328430479974486, 1.2540256879983644, 1.23260023200055,
1.4384445140021853, 1.4309636320031132, 1.233868880000955, 1.2330196219991194, 1.2194298460017308,
2.6728919979977945, 2.6466648089990485, 2.6431999880005606, 2.6444571530009853, 2.694105665999814,
2.6978086060007627, 2.6593154429974675, 2.6565679920022376, 2.683303400001023, 2.642734171000484,
3.646107745000336, 3.6738023340003565, 3.8685401610018744, 3.6627948220011604, 3.6828842840004654,
3.698112420999678, 3.6536567869989085, 3.6514857850015687, 3.6651766520008096, 3.6637670769996475,
6.095307376999699, 6.433294025999203, 6.088375280000037, 6.35739769599968, 6.313441080998018, 6.096856126001512,
6.148383091000142, 6.103424116001406, 6.1642854059973615, 6.329313353002362, 10.43963070600148,
10.670867131000705, 10.511210079002922, 10.524127229000442, 10.613739050000731, 10.556520229998569,
10.470421940997767, 10.85152145200118, 10.521694461000152, 10.556238698001835, 13.467594155998086,
13.50144059000013, 13.780776005001826, 13.289006597002299, 13.36134642700199, 13.468484715001978,
13.759095815003093, 13.340606084002502, 13.32315439300146, 13.332875502997922, 26.213422912998794,
25.28450629500003, 25.650899018000928, 25.452737540999806, 25.430792176000978, 25.593701723999402,
25.337706698999682, 26.195807776002766, 25.640305346001696, 25.1273522770025)
```

```
tw = c(0.08591600000000099, 0.06884939999999773, 0.08364780000000138, 0.06753369999999848, 0.06565009999999916,
0.06630610000000559, 0.06601419999999791, 0.06425370000000896, 0.08083370000002787, 0.08016530000000444,
0.25995779999999513, 0.27242499999999836, 0.2677836999999954, 0.28024970000001304, 0.2764307000000201,
0.28842030000001273, 0.2682694000000083, 0.2827100999999743, 0.2615483000000154, 0.2734345000000076,
0.8147506000000249, 0.776667400000008, 0.8179140999999959, 0.7965328999999883, 0.8087623000000121,
0.8029727999999636, 0.8140838999999573, 0.7939167000000111, 0.803889799999979, 0.800447800000029,
1.1277521000000092, 1.1119950000000358, 1.111627799999951, 1.126531200000045, 1.128309399999921,
1.1289717999999311, 1.1137796000000435, 1.0938085000000228, 1.1575381000000107, 1.1038780999999744,
2.585786999999982, 2.476835900000083, 2.4963325000001078, 2.482979099999966, 2.5527525999999625,
2.4936506000000236, 2.514798500000097, 2.470065399999953, 2.4743545999999697, 2.5436187000000245,
3.684525200000053, 3.719213800000034, 3.836824399999955, 3.7310571000000436, 3.650232499999902,
3.7484018000000106, 3.6528604999999743, 3.718401999999969, 3.653249399999936, 3.751383300000043,
6.2927603000000545, 6.264846000000034, 6.339046600000074, 6.448770700000068, 6.383027500000026, 6.364449899999954,
6.277007400000002, 6.283744200000001, 6.342126400000097, 6.407319799999868, 11.039521599999944,
10.946878499999912, 10.93084419999991, 11.014599699999962, 11.056521199999906, 10.91438879999987,
10.880907799999932, 14.962613200000078, 10.96586400000001, 11.135259899999937, 18.180663500000037,
14.166438700000072, 14.114150999999993, 14.096150200000011, 13.957862500000147, 14.125394799999867,
14.092164899999943, 14.101538000000119, 14.929516199999853, 13.946655899999996, 26.479466499999944,
26.48343970000019, 26.38855109999986, 26.8889979999999, 26.639894800000093, 26.471626300000025,
26.424158499999976, 27.22346440000001, 26.713380799999868, 26.433829699999933)
```

```
px = c(217217, 217217, 217217, 217217, 217217, 217217, 217217, 217217, 217217, 217217, 486209, 486209, 486209, 486209,
486209, 486209, 486209, 486209, 486209, 486209, 1053360, 1053360, 1053360, 1053360, 1053360, 1053360, 1053360,
1053360, 1053360, 1053360, 1309100, 1309100, 1309100, 1309100, 1309100, 1309100, 1309100, 1309100, 1309100, 2172573,
2172573, 2172573, 2172573, 2172573, 2172573, 2172573, 2172573, 2172573, 2172573, 2712200, 2712200, 2712200, 2712200,
2712200, 2712200, 2712200, 2712200, 2712200, 2712200, 3713346, 3713346, 3713346, 3713346, 3713346, 3713346, 3713346,
3713346, 3713346, 3713346, 5020371, 5020371, 5020371, 5020371, 5020371, 5020371, 5020371, 5020371, 5020371,
5692248, 5692248, 5692248, 5692248, 5692248, 5692248, 5692248, 5692248, 5692248, 8018100, 8018100, 8018100,
8018100, 8018100, 8018100, 8018100, 8018100, 8018100, 8018100)
```

```
logtu = log(tu)
```

```
logtw = log(tw)

logpx = log(px)


summary(tu)

summary(tw)


tD = tu - tw

sD = sd(tD)

sDu = sd(tu); sDw = sd(tw)

covu = cov(logtu, px); covu

covw = cov(logtw, px); covw

coru = cor(logtu, px); coru

corw = cor(logtw, px); corw

meanD = mean(tD)

length(tD)


#EstadÃstic

T = (meanD-0)/(sD/sqrt(n)); T


#Distribució

distr = qt(0.95, n - 1); distr


#pvalor

pvalor = pt(T, 0.95)


#IC 95%

ic<-c(meanD - qnorm(0.95)*sqrt(sD*sD/n), meanD + qnorm(0.95)*sqrt(sD*sD/n))



#GrÃ fic dades

plot(px, tu, type="p", col="red", xlab = "N? Pixels", ylab = "Temps execuci? (s)")

points(px, tw, col="blue") # DADES


modelu <- lm(logtu ~ logpx)

modelw <- lm(logtw ~ logpx)
```

```
#Boxplot

boxplot(tu, tw, names=c("Ubuntu", "Windows"), ylab="Temps (s)");


BlandAltmanLeh::bland.altman.plot(tw, tu, ylab="Difer?ncia en d'execuci? en segons Windows respecte Ubuntu", xlab="Temps execucio en segons")


#recta regressiÃ³ Ubuntu

plot(logpx, logtu, main="Ubuntu", ylab = "Temps (s)", xlab="P?xels");

abline(modelu, col = "red");


#recta regressiÃ³ Windows

plot(logpx, logtw, main="Windows", ylab = "Temps (s)", xlab="P?xels");

abline(modelw, col = "red");


#Normal Q-Q Ubuntu

qqnorm(logtu, logpx, main="Normal Q-Q Ubuntu", ylab = "Quantils", xlab="Quantils te?rics", ylim=c(0, 10))

qqline(logtu, col="red")


#Normal Q-Q Windows

qqnorm(logtw, logpx, main="Normal Q-Q Windows", ylab = "Quantils", xlab="Quantils te?rics", ylim=c(0, 10))

qqline(logtw, col="red")


hist(logtu,main="Histograma Ubuntu",xlab="Temps",ylab="N? d'ocurr?ncies")

hist(logtw,main="Histograma Windows",xlab="Temps",ylab="N? d'ocurr?ncies")


plot(fitted(modelu), resid(modelu), ylim=c(-1, 1)); abline(0,0, lty = 5)

plot(fitted(modelw), resid(modelw), ylim=c(-1, 1)); abline(0,0, lty = 5)


plot(modelu, c(2,1)) # Ubuntu residual vs fitted and QQnorm

plot(modelw, c(2,1)) # Windows residual vs fitted and QQnorm


plot(1:100, rstandard(modelu), type="l"); abline(0,0, lty = 5); points(rstandard(modelu)) # Ubuntu Ordre dels residus

plot(1:100, rstandard(modelw), type="l"); abline(0,0, lty = 5); points(rstandard(modelw)) # Windows Ordre dels residus
```