Estudio empírico del orden de la multiplicación de matrices

Ricardo Alberich.

Contents

1 Generación de la muestra

La muestra se ha obtenido con el código siguiente que no se ejecuta. Por reproducibilidad se ha guardado en el objeto de R `dades1000` y no ejecutamos el código

```
library(future.apply)
set.seed(2020)# fijo la semilla para poder reproducir los datos
#lanzo la muestra
data1000=future_sapply(seq(10,5000,100),FUN=function(n){
random_matrix_n=function(n){
M1=matrix(runif(n^2),ncol=n)
M2=matrix(runif(n^2),ncol=n)
system.time(M1%*%M2, gcFirst = TRUE)}
c(n,sum(replicate(1,random_matrix_n(n),simplify="array")))
}
)
#guardamos el objeto en el working directory actual
save(data1000,file="data1000.Robj")
```

2 Carga de datos

Cargamos los datos, los transformamos en un `data frame` de dos variables `n` y `seconds`

```
#cargamos el objeto del working directory actual
load("data1000.Robj")
class(data1000)# clase
```

```
## [1] "matrix"
```

```r
str(data1000)# estructura
```

```
##  num [1:2, 1:50] 10 0 110 0.001 210 ...
```

```r
head(data1000)# primeros datos
```

```
##      [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]    [,8]    [,9]
## [1,]   10 110.000 210.000 310.000 410.000 510.000 610.000 710.000 810.000
## [2,]    0   0.001   0.003   0.008   0.013   0.026   0.045   0.068   0.094
##        [,10]    [,11]    [,12]    [,13]    [,14]    [,15]    [,16]    [,17]
## [1,] 910.000 1010.000 1110.000 1210.000 1310.000 1410.000 1510.000 1610.000
## [2,]   0.292    0.193    0.288    0.294    0.588    0.445    0.504    0.581
##        [,18]    [,19]    [,20]    [,21]    [,22]   [,23]    [,24]    [,25]
## [1,] 1710.00 1810.000 1910.000 2010.000 2110.000 2210.00 2310.000 2410.000
## [2,]    0.87    1.182    1.047    1.098    1.276    1.54    1.815    2.089
##         [,26]    [,27]    [,28]    [,29]    [,30]    [,31]    [,32] [,33]
## [1,] 2510.000 2610.000 2710.000 2810.000 2910.000 3010.000 3110.000  3210
## [2,]    2.307    2.823    2.956    3.361    3.606    3.994    4.457     5
##         [,34]    [,35]    [,36]   [,37]    [,38]    [,39]    [,40]    [,41]
## [1,] 3310.000 3410.000 3510.000 3610.00 3710.000 3810.000 3910.000 4010.000
## [2,]    5.529    6.207    7.027    7.59    8.092    9.276    9.934   11.086
##         [,42]    [,43]    [,44]    [,45]    [,46]    [,47]    [,48]    [,49]
## [1,] 4110.000 4210.000 4310.000 4410.000 4510.000 4610.000 4710.000 4810.000
## [2,]   10.301   11.292   12.913   13.575   15.122   15.777   18.295   22.363
##         [,50]
## [1,] 4910.000
## [2,]   20.598
```

```r
data=as.data.frame(matrix(unlist(data1000),ncol=2,byrow=TRUE))
head(data)
```

```
##    V1    V2
## 1  10 0.000
## 2 110 0.001
## 3 210 0.003
## 4 310 0.008
## 5 410 0.013
## 6 510 0.026
```

```r
#ponemos nombres
names(data)=c("n","seconds")
#eliminamos las filas con seconds=0
data=data[data$seconds!=0,]
head(data)
```

```
##     n seconds
## 2 110   0.001
## 3 210   0.003
## 4 310   0.008
## 5 410   0.013
## 6 510   0.026
## 7 610   0.045
```

3 Estudio del ajuste de tres modelos de curvas de orden

Vamos a estudiar si los modelos de orden del algoritmo de multiplicación de matrices se ajustan a un modelo lineal a uno exponencial o a uno potencial. Recordemos que sabemos que la multiplicación de matrices cuadradas de orden es $O(n^3)$.

Os pongo sólo el código simple, vosotros ampliad-lo y poned nombres a los gráficos

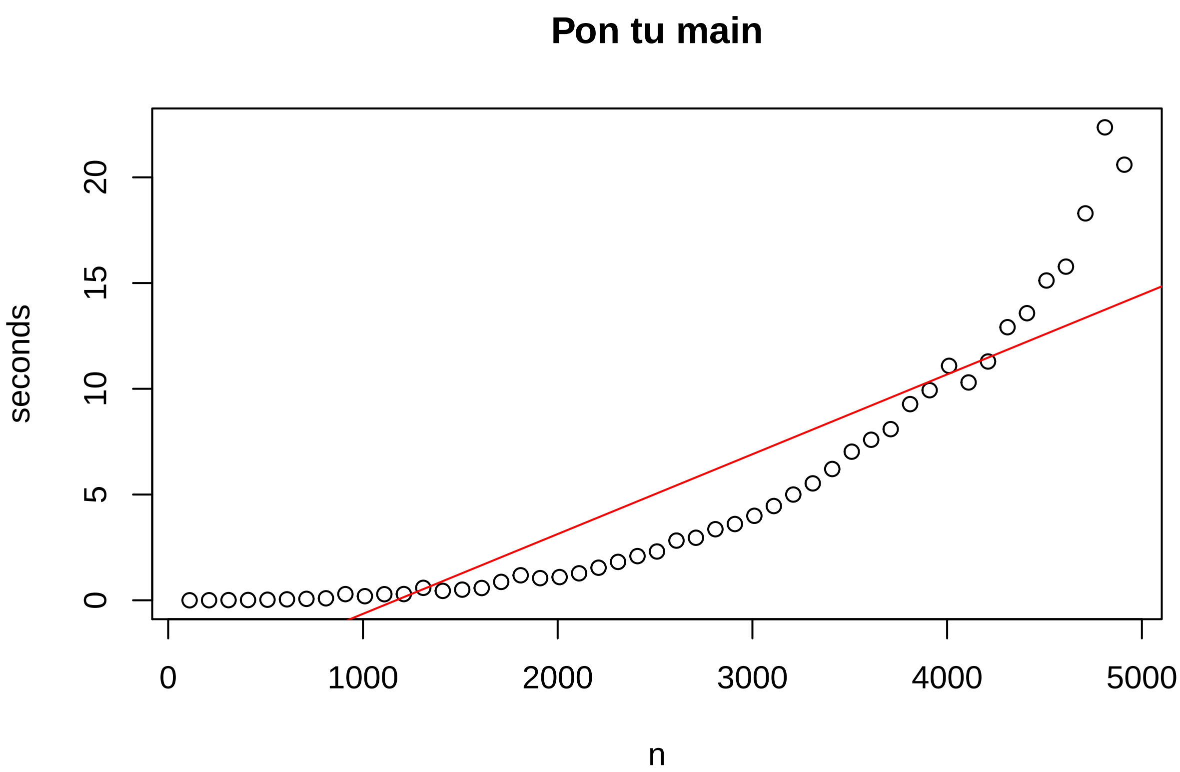
3.1 Regresión lineal

Pon el modelo

```
lm_x_y=lm(seconds~n,data=data)
summary(lm_x_y)
```

```
##
## Call:
## lm(formula = seconds ~ n, data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.962 -2.272 -0.689  1.453  8.623
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.4163099  0.7888949  -5.598 1.09e-06 ***
## n            0.0037746  0.0002738  13.785  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.711 on 47 degrees of freedom
## Multiple R-squared:  0.8017, Adjusted R-squared:  0.7975
## F-statistic:   190 on 1 and 47 DF,  p-value: < 2.2e-16
```

```
plot(data,main="Pon tu main")
abline(lm_x_y,col="red")
```



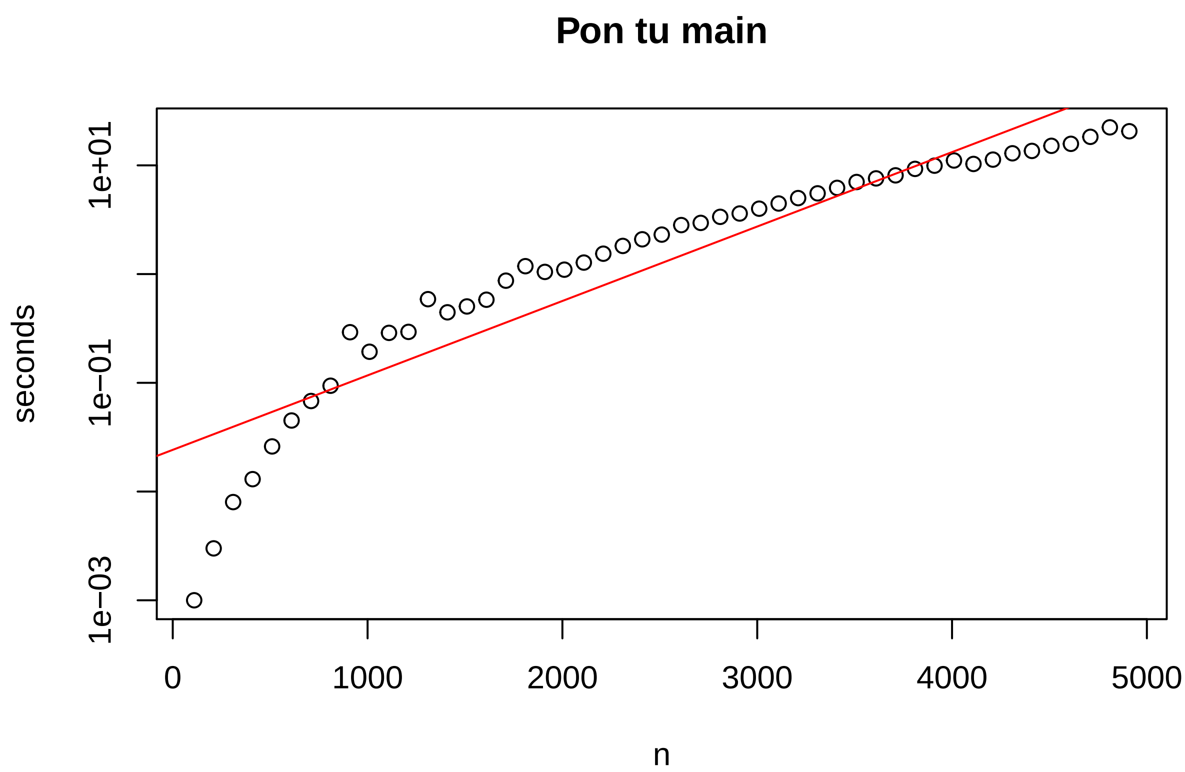
3.2 Regresión exponencial (semilog)

Pon el modelo

```
lm_x_logy=lm(log10(seconds)~n,data=data)
summary(lm_x_logy)
```

```
##
## Call:
## lm(formula = log10(seconds) ~ n, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.45878 -0.21276  0.09324  0.28050  0.48911
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.617e+00  1.131e-01  -14.29   <2e-16 ***
## n            6.846e-04  3.926e-05   17.43   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3887 on 47 degrees of freedom
## Multiple R-squared:  0.8661, Adjusted R-squared:  0.8632
## F-statistic:   304 on 1 and 47 DF,  p-value: < 2.2e-16
```

```
plot(data,main="Pon tu main",log="y")
abline(lm_x_logy,col="red")
```

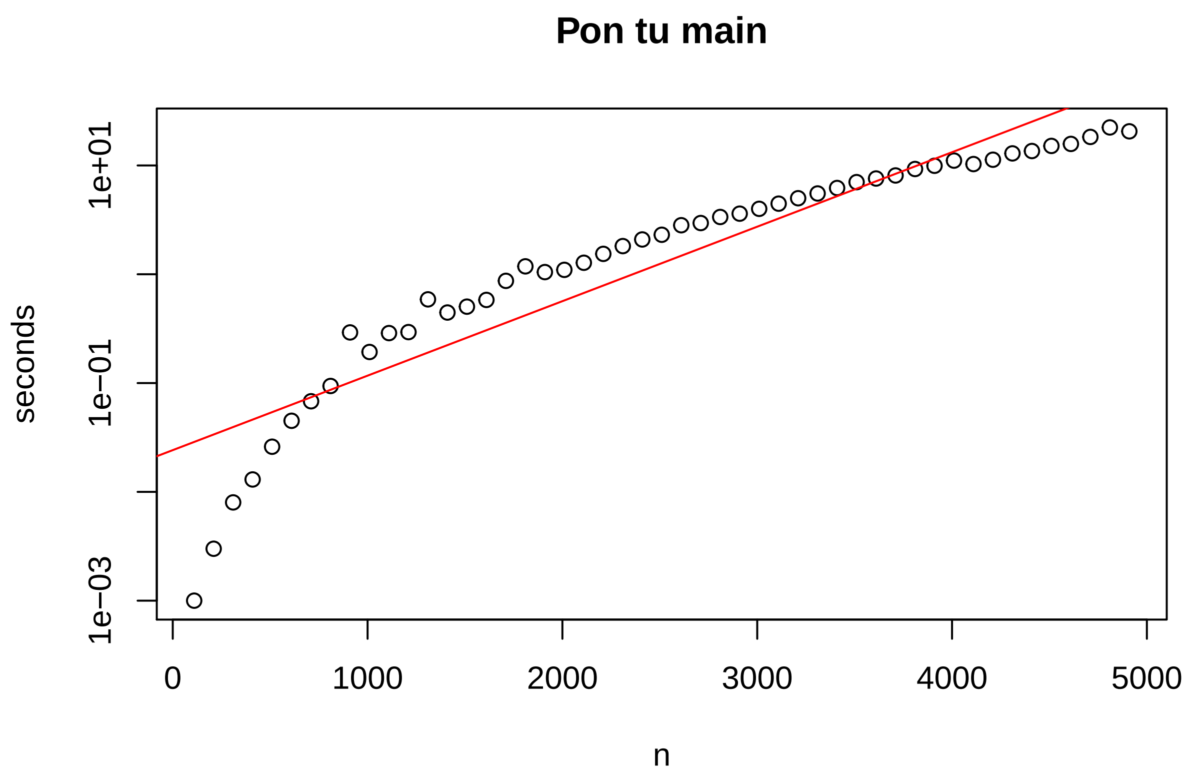


3.3 Regresión potencial (loglog)

```
lm_x_logy=lm(log10(seconds)~n,data=data)
summary(lm_x_logy)
```

```
##
## Call:
## lm(formula = log10(seconds) ~ n, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.45878 -0.21276  0.09324  0.28050  0.48911
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.617e+00  1.131e-01  -14.29   <2e-16 ***
## n            6.846e-04  3.926e-05   17.43   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3887 on 47 degrees of freedom
## Multiple R-squared:  0.8661, Adjusted R-squared:  0.8632
## F-statistic:   304 on 1 and 47 DF,  p-value: < 2.2e-16
```

```
plot(data,main="Pon tu main",log="y")
abline(lm_x_logy,col="red")
```

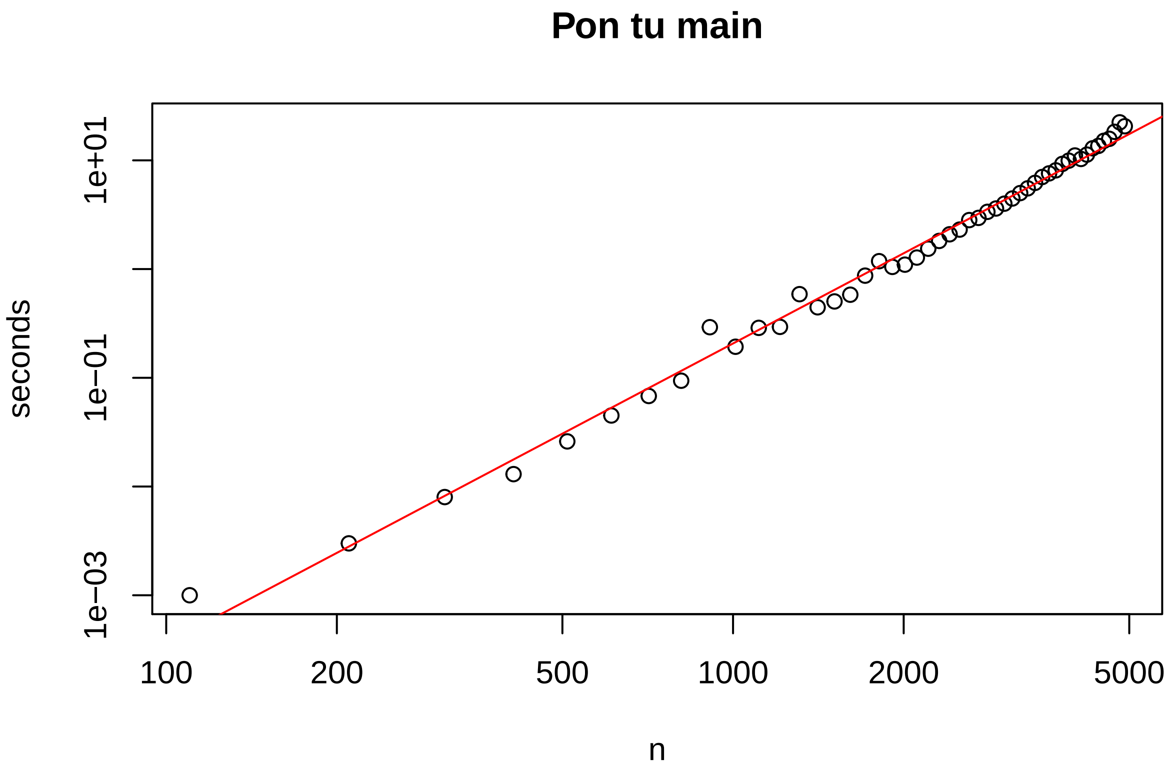


Pon el modelo

```
lm_logx_logy=lm(log10(seconds)~log10(n),data=data)
summary(lm_logx_logy)
```

```
##
## Call:
## lm(formula = log10(seconds) ~ log10(n), data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.13477 -0.07030 -0.01262  0.04673  0.32590
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.95146    0.11487  -77.92   <2e-16 ***
## log10(n)     2.75575    0.03474   79.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09146 on 47 degrees of freedom
## Multiple R-squared:  0.9926, Adjusted R-squared:  0.9924
## F-statistic:  6293 on 1 and 47 DF,  p-value: < 2.2e-16
```

```
plot(data,main="Pon tu main",log="xy")
abline(lm_logx_logy,col="red")
```



4 Estudio del ajuste de tres modelos de curvas de orden

4.1 Selección del mejor modelo

En principio el modelo con mejor R^2 es el log-log que equivale a una curva potencial

$$second \approx \beta \cdot n^{\alpha}$$

.

Como hemos visto en el manual (tema2 MOOC), si $log_{10}(seconds) = a \cdot log_{10}(n) + b$ Entonces $\beta = 10^b$ y $\alpha = a$.

Así que

```
b=lm_logx_logy$coefficients[1]
a=lm_logx_logy$coefficients[2]
b
```

```
## (Intercept)
##   -8.951463
```

```
a
```

```
## log10(n)
##  2.75575
```

```
beta=10^b
beta
```

```
##  (Intercept)
## 1.118245e-09
```

```
alpha=a
alpha
```

```
## log10(n)
##  2.75575
```

5 Estudio del ajuste de tres modelos de curvas de orden

Así que el modelo potencial es

$$secons \approx 1.1182449 \times 10^{-9} \cdot n^{`r\alpha}$$

.

Ahora podemos hacer el dibujo de la curva y los datos en las unidades originales

```
plot(data,main="Simulación del orden de complejidad\n de la multiplicación de matrices")
curve(beta*x^alpha,add=TRUE,col="blue")
```