Υπολογισμός του γινομένου $A^T \cdot A \cdot A^T$ με τη γλώσσα Julia

Και σύγκριση με την Python

Μέρη της παρουσίασης

1. Σύγκριση των δύο μεθόδων υπολογισμού:

   $(A^T \cdot A) \cdot A^T$ και $A^T \cdot (A \cdot A^T)$

   (*αξιοποιώντας κατάλληλα εργαλεία που προσφέρει η Julia)*

2. Υλοποίηση των δύο μεθόδων υπολογισμού και στην Python, ώστε να συγκριθούν οι χρόνοι εκτέλεσης των δύο γλωσσών.

ΜΕΡΟΣ 1

Σύγκριση των Μεθόδων Υπολογισμού

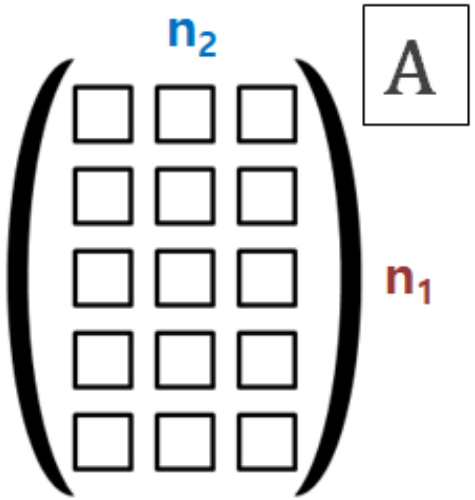
Σύγκριση σε θεωρητικό επίπεδο: Λόγοι

- Λόγος υπολογιστικής πολυπλοκότητας (M1/M2):

$$\frac{2n_1n_2^2}{2n_1^2n_2} = \frac{n_2}{n_1}$$

- Λόγος απαιτούμενων θέσεων μνήμης (M1/M2):

$$\frac{(n_1+n_2)\cdot n_2}{(n_1+n_2)\cdot n_1} = \frac{n_2}{n_1}$$



Σύγκριση σε θεωρητικό επίπεδο: Λόγοι

- Λόγος υπολογιστικής πολυπλοκότητας (M1/M2):

$$\frac{2n_1n_2^2}{2n_1^2n_2} = \frac{n_2}{n_1}$$

- Λόγος απαιτούμενων θέσεων μνήμης (M1/M2):

$$\frac{(n_1+n_2)\cdot n_2}{(n_1+n_2)\cdot n_1} = \frac{n_2}{n_1}$$

Συμπέρασμα: Περιμένουμε και ο χρόνος εκτέλεσης να εξαρτάται από το *σχήμα* του πίνακα A

Υλοποίηση σε Julia για μέτρηση του χρόνου εκτέλεσης

```
function M1(A)
B = (A'*A)*A'
return
end
```

```
function M2(A)
B = A'*(A*A')
return
end
```

Υλοποίηση σε Julia για μέτρηση του χρόνου εκτέλεσης

```
function M1(A)
B = (A'*A)*A'
return
end
```

```
function time_M1(A)
@btime M1($A)
end
```

```
function M2(A)
B = A'*(A*A')
return
end
```

```
function time_M2(A)
@btime M2($A)
end
```

Υλοποίηση σε Julia για μέτρηση του χρόνου εκτέλεσης

```
function M1(A)
B = (A'*A)*A'
return
end
```

```
using BenchmarkTools
function time_M1(A)
@btime M1($A)
end
```

```
function M2(A)
B = A'*(A*A')
return
end
```

```
using BenchmarkTools
function time_M2(A)
@btime M2($A)
end
```

Υλοποίηση σε Julia για μέτρηση του χρόνου εκτέλεσης

```
function M1(A)
B = (A'*A)*A'
return
end
```

```
using BenchmarkTools
function time_M1(A)
@btime M1($A)
end
```

```
function M2(A)
B = A'*(A*A')
return
end
```

```
using BenchmarkTools
function time_M2(A)
@btime M2($A)
end
```

496.231 ms (6 allocations: 763.40 MiB)

Διαφορετικά σχήματα του πίνακα

```
function P_1(n, m)
s = n*10^m
A = [Float64(i) for i in range(-s/2+1, s/2)]
for i = 0:m-1
        A = reshape(A, n*10^i, 10^(m-i))
        println(n*10^i, 'x', 10^(m-i))
        time_M1(A)
        time_M2(A)
        println("----------")
        A = reshape(A, 10^(i+1), n*10^(m-i-1))
        println(10^(i+1), 'x', n*10^(m-i-1))
        time_M1(A)
        time_M2(A)
        println("----------")
        end
end
```

Διαφορετικά σχήματα του πίνακα

```
function P_1(n, m)
s = n*10^m
A = [Float64(i) for i in range(-s/2+1, s/2)]
for i = 0:m-1
        A = reshape(A, n*10^i, 10^(m-i))
        println(n*10^i, 'x', 10^(m-i))
        time_M1(A)
        time_M2(A)
        println("----------")
        A = reshape(A, 10^(i+1), n*10^(m-i-1))
        println(10^(i+1), 'x', n*10^(m-i-1))
        time_M1(A)
        time_M2(A)
        println("----------")
        end
end
```

```
3×10000
496.231 ms (6 allocations: 763.40 MiB)
115.500 μs (5 allocations: 468.97 KiB)
———-
10×3000
31.927 ms (6 allocations: 69.12 MiB)
104.500 μs (5 allocations: 469.72 KiB)
———-
30×1000
2.847 ms (6 allocations: 8.09 MiB)
133.600 μs (5 allocations: 476.03 KiB)
———-
100×300
389.100 μs (6 allocations: 1.14 MiB)
200.900 μs (6 allocations: 547.02 KiB)
———-

κλπ.
```

n=3, m=4

Εκτέλεση για n = 5 και m = 4

Διαστάσεις:	5×10000	10×5000	50×1000	100×500
Μέθοδος 1	470.605 ms	101.106 ms	3.129 ms	980.000 μs
Μέθοδος 2	132.500 μs	131.100 μs	176.800 μs	253.600 μs
Λόγος	3551.7358	771.2128	17.6980	3.8644

500×100	1000×50	5000×10	10000×5
245.600 μs	198.400 μs	133.000 μs	140.300 μs
939.400 μs	3.380 ms	119.737 ms	555.251 ms
0.2614	0.0587	0.0011	0.0003

Τύπος των εγγραφών

```
function P_1(n, m, f=Float64)
s = n*10^m
A = [f(i) for i in range(-s/2+1, s/2)]
for i = 0:m-1
        A = reshape(A, n*10^i, 10^(m-i))
        println(n*10^i, 'x', 10^(m-i))
        time_M1(A)
        time_M2(A)
        println("----------")
        A = reshape(A, 10^(i+1), n*10^(m-i-1))
        println(10^(i+1), 'x', n*10^(m-i-1))
        time_M1(A)
        time_M2(A)
        println("----------")
        end
end
```

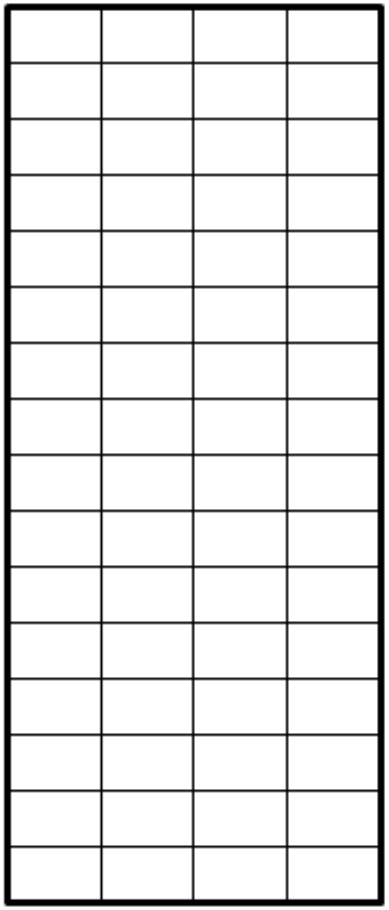
Τύπος των εγγραφών

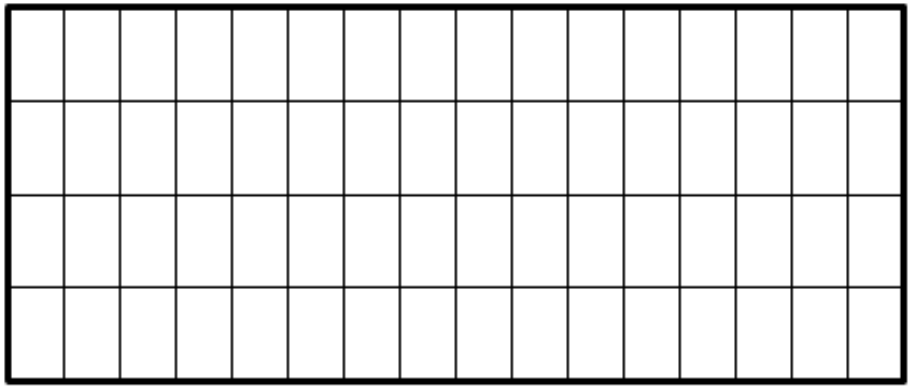
```
function P_1(n, m, f=Float64)
s = n*10^m
A = [f(i) for i in range(-s/2+1, s/2)]
for i = 0:m-1
        A = reshape(A, n*10^i, 10^(m-i))
        println(n*10^i, 'x', 10^(m-i))
        time_M1(A)
        time_M2(A)
        println("----------")
        A = reshape(A, 10^(i+1), n*10^(m-i-1))
        println(10^(i+1), 'x', n*10^(m-i-1))
        time_M1(A)
        time_M2(A)
        println("----------")
        end
end
```

f =

- Float16
- Float32
- Float64
- Int16
- Int32
- Int64

Πίνακες με πολλές γραμμές ή πολλές στήλες





Εκτέλεση για πίνακες με πολλές γραμμές ή πολλές στήλες

- Πίνακες τύπου Big Data:

  (παρουσιάζονται οι χρόνοι της Μεθόδου 1 μόνο)

Γραμμές:	10^2	10^3	10^4	10^5	10^6	10^7	10^8
3 στήλες	1.720 μs	11.500 μs	107.200 μs	946.600 μs	12.144 ms	123.828 ms	1.232 s
5 στήλες	2.211 μs	15.000 μs	141.500 μs	1.383 ms	21.338 ms	213.499 ms	2.251 s
10 στήλες	3.900 μs	31.200 μs	254.500 μs	4.606 ms	57.967 ms	594.739 ms	14.894 s

- Πίνακες τύπου High Dimensional Data:

  (παρουσιάζονται οι χρόνοι της Μεθόδου 2 μόνο)

Στήλες:	10^2	10^3	10^4	10^5	10^6	10^7	10^8
3 γραμμές	1.640 μs	10.800 μs	98.800 μs	972.000 μs	10.976 ms	108.077 ms	1.103 s
5 γραμμές	1.980 μs	14.100 μs	131.800 μs	1.354 ms	14.963 ms	147.417 ms	1.589 s
10 γραμμές	3.638 μs	29.000 μs	219.600 μs	3.074 ms	30.649 ms	299.361 ms	15.066 s

ΜΕΡΟΣ 2

Σύγκριση της Julia με την Python

Υλοποίηση των μεθόδων σε Python

```
def Method1(A):
    B = (A.T*A)*A.T
    return B
```

```
def Method2(A):
    B = A.T*(A*A.T)
    return B
```

Υλοποίηση των μεθόδων σε Python

```
def Method1(A):
    B = (A.T*A)*A.T
    return B
```

```
def Method2(A):
    B = A.T*(A*A.T)
    return B
```

```
import numpy
numpy.mat(διδιάστατη_λίστα)
numpy.array(διδιάστατη_λίστα)
```

Υλοποίηση των μεθόδων σε Python

```
def Method1(A):
    B = (A.T*A)*A.T
    return B
```

```
def Method2(A):
    B = A.T*(A*A.T)
    return B
```

```
import time
def time_M1(A):
    times = [0.0]*100
    for i in range(100):
        tic = time.perf_counter()
        Method1(A)
        toc = time.perf_counter()
        times[i] = toc - tic
    return min(times)
```

```
import time
def time_M2 (A):
    times = [0.0]*100
    for i in range(100):
        tic = time.perf_counter()
        Method2(A)
        toc = time.perf_counter()
        times[i] = toc - tic
    return min(times)
```

Σύγκριση για διάφορα σχήματα του πίνακα Α

- Χρόνοι Python:

	3x10000	10x3000	30x1000	100x300	300x100	1000x30	3000x10	10000x3
Μέθοδος 1	0.6612451	0.0409254	0.0051908	0.0006549	0.0002634	0.0001772	0.0001431	0.0001036
Μέθοδος 2	0.0001181	0.0001501	0.0001878	0.0002617	0.0006631	0.0050689	0.0397772	0.6029755

- Λόγοι Python/Julia:

	3x10000	10x3000	30x1000	100x300	300x100	1000x30	3000x10	10000x3
Μέθοδος 1	1.346981	1.344859	1.820694	1.717545	1.448447	1.217869	1.46319	0.975059
Μέθοδος 2	1.185148	1.640437	1.567613	1.398717	1.832527	1.641483	1.204384	1.071554

Σύγκριση για τετραγωνικούς πίνακες

- Χρόνοι Python:

	100x100	316x316	1000x1000	3160x3160	10000x10000	31600x31600
Μέθοδος 1	0.0001947	0.0013469	0.0201069	0.4145732	10.9509571	MemoryError
Μέθοδος 2	0.0001979	0.001344	0.0196797	0.407547	10.7451892	MemoryError

- Λόγοι Python/Julia:

	100x100	316x316	1000x1000	3160x3160	10000x10000	31600x31600
Μέθοδος 1	1.67268	1.920029	1.381253	1.1150526	1.048440124	-
Μέθοδος 2	1.725371	1.947826	1.3243405	1.0755319	1.030417069	-

Σύγκριση για πίνακες τύπου Big Data

- Χρόνοι Μεθόδου 1 στην Python:

100x3	1000x3	10000x3	100000x3	1000000x3	10000000x3	100000000x3
0.000010	0.000018	0.0001673	0.0016162	0.0182335	0.1696696	4.514029

- Λόγοι Python/Julia (v2):

100x3	1000x3	10000x3	100000x3	1000000x3	10000000x3	100000000x3
5.7558	1.6087	1.5606	1.7074	1.5014	1.3702	3.6640

Σύγκριση για πίνακες τύπου High Dimensional Data

- Χρόνοι Μεθόδου 2 στην Python:

5x100	5x1000	5x10000	5x100000	5x1000000	5x10000000	5x100000000
0.000010	0.000024	0.0002588	0.0026256	0.0323429	0.3210824	6.2524447

- Λόγοι Python/Julia:

5x100	5x1000	5x10000	5x100000	5x1000000	5x10000000	5x100000000
5.25252483	1.67375871	1.96358120	1.93914328	2.16152510	2.17805545	3.93482989

Συμπέρασμα

Julia:

+ Πιο γρήγορη
+ Απλή σύνταξη για τη δημιουργία πινάκων
+ Απλή σύνταξη για τη δημιουργία λίστας (με for)
– Χρήση end στο τέλος βρόχων και συναρτήσεων
+ Άμεση πρόσβαση σε απλά εργαλεία για τη βελτιστοποίηση του κώδικα
+ Διευκόλυνση ακριβούς προσδιορισμού του τύπου των δεδομένων (Float32, Float64 κλπ.)

Python:

– Πιο αργή
– Κλήση συνάρτησης της βιβλιοθήκης numpy για τη δημιουργία πίνακα
+ Απλή σύνταξη για τη δημιουργία λίστας (με for)
+ Χρήση εσοχών για βρόχους και συναρτήσεις
– Τα εργαλεία μέτρησης του απαιτούμενου χρόνου και της μνήμης είναι διασκορπισμένα σε διάφορες βιβλιοθήκες
– Χρήση πιο ασαφών τύπων δεδομένων (π.χ. float)