ΕΡΩΤΗΜΑ 1

i,ii) Pentium (R) Dual-Core CPU E6600 @3.06GHz

Cache 2 επιπέδων με L1 Cache : 32KB και L2 Cache : 2048KB με πολιτική εγγραφής write back.

DDR2 Dual Channel Memory συνολικού μεγέθους 4GB

iii) Windows 7 Ultimate SP1 x64 και MATLAB R(2015a) (8.5.0.197613)

iv) Matlab Benchmark

================

Number of times each test is run___________________________: 3

I. Matrix calculation

Creation, transp., deformation of a 1200x1200 matrix (sec): 0.067446

1250x1250 normal distributed random matrix ^1000____ (sec): 0.11345

Sorting of 1,100,000 random values___________________ (sec): 0.067414

550x550 cross-product matrix (b = a' * a)___________ (sec): 0.02786

Linear regression over a 700x700 matrix (c = a \ b') (sec): 0.045408

Trimmed mean (2 extremes eliminated): 0.060089

II. Matrix functions

FFT over 900,000 random values_______________________ (sec): 0.038588

Eigenvalues of a 220x220 random matrix______________ (sec): 0.065184

Determinant of a 750x750 random matrix______________ (sec): 0.040486

Cholesky decomposition of a 1000x1000 matrix________ (sec): 0.057715

Inverse of a 500x500 random matrix__________________ (sec): 0.045756

Trimmed mean (2 extremes eliminated): 0.047986

III. Programmation

225,000 Fibonacci numbers calculation (vector calc)_ (sec): 0.041968

Creation of a 1500x1500 Hilbert matrix (matrix calc) (sec): 0.083566

Grand common divisors of 35,000 pairs (recursion)___ (sec): 0.018112

Creation of a 220x220 Toeplitz matrix (loops)_______ (sec): 0.00038868

Escoufier's method on a 22x22 matrix (mixed)________ (sec): 0.089615

Trimmed mean (2 extremes eliminated): 0.047882

Total time for all 15 tests_________________________ (sec): 0.80296

Overall mean (sum of I, II and III trimmed means/3)_ (sec): 0.051986

--- End of test ---

ΕΡΩΤΗΜΑ 2

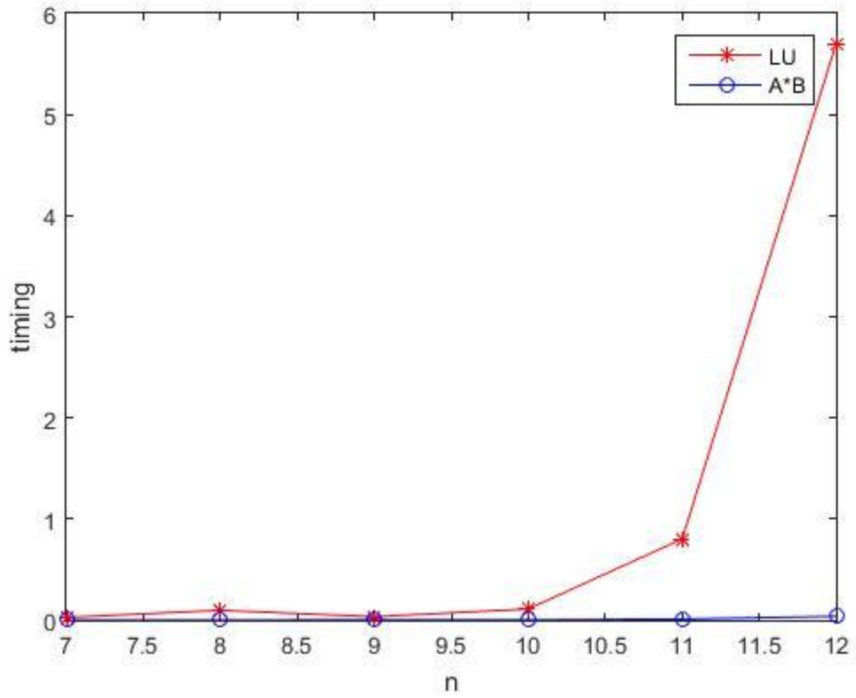
i)

- **LU**= Είναι ένας τρόπος διάσπασης του μητρώου σε δυο μητρώα L και U, στην οποία το L είναι κάτω τριγωνικό και το U είναι άνω τριγωνικό μητρώο. Ο κάτω τριγωνικός L έχει όλο μονάδες στην διαγώνιό του και οι πολλαπλασιαστές lij βρίσκονται κάτω στη διαγώνιο του L . A=L*U (ενδογενής).
- **QR**= Είναι ένας τρόπος διάσπασης του μητρώου σε δυο μητρώα Q και R, στην οποία το R είναι άνω τριγωνικό μητρώο και το Q είναι ορθογώνιο μητρώο με ορθοκανονικές στήλες. A=Q*R (ενδογενής).
- **SVD**= Είναι ένας τρόπος διάσπασης του μητρώου σε τρία μητρώα U, S και V’, τα μητρώα U και V είναι ορθογώνια μητρώα και το S είναι μητρώο που έχει μη μηδενικά στοιχεία στην διαγώνιο και είναι σε φθίνουσα σειρά. A=U*S*V’ (ενδογενής).
- **DET**= Επιστρέφει την ορίζουσα του τετραγωνικού μητρώου. det(A) (m-function).
- **RANK**= Επιστρέφει την τάξη του μητρώου. Αν το μητρώο το μετατρέψουμε σε αναγμένων γραμμών κλιμακωτή μορφή τότε η τάξη του μητρώου είναι το πλήθος των οδηγών. rank(A) (m-function).
- **Polyval(p,x)=** Επιστρέφει την τιμή ενός πολυωνύμου βαθμού n υπολογιζόμενη στο x. Η είσοδος p είναι ένα διάνυσμα μήκους n + 1 τα στοιχεία της οποίας είναι οι συντελεστές κατά φθίνουσα σειρά, του πολυωνύμου, που πρέπει να αξιολογηθούν (m-function).

i)

```
for j=1:6
   i = j+6;
   n = pow2(i);
   A = (rand(n)); % nxn
   b = (rand(n,1)); %nx1
   tic % ksekinaei o xronos gia lu
   [L,U] = lu(A);
   timing(j) = toc
   %A apo8ikeuoume ton xrono pou vrikame
   tic   % ksekinaei o xronos gia A*B
   B=A*b;
   timing1(j) = toc

end
   figure
   n=7:1:12;
   plot(n,timing,'r-*')
   hold on;
   plot(n,timing1,'b-o');
   hold off;
   xlabel('n')
   ylabel('timing')
   legend('LU','A*B')
```

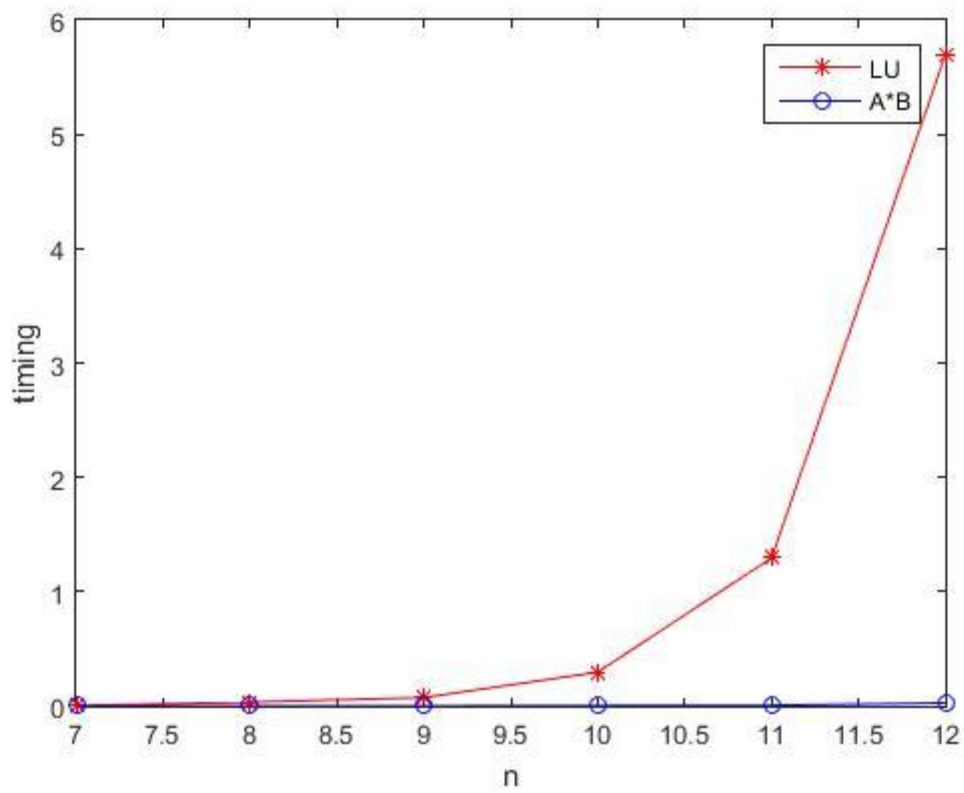
```
timing =

   0.0343    0.1310    0.0325    0.1845    1.1583    7.2038

timing1 =

   0.0007    0.0002    0.0010    0.0024    0.0099    0.0327
```



ii)

```
for j=1:6
   i = j+6;
   n = pow2(i);
   A = (rand(n));
   b = (rand(n,1));
   tic
   for t=1:500 %500 epanalipseis
      [L,U] = lu(A);
   end
   timing(j) = toc./500
   tic
   for t=1:500
   B = A*b;

end
   timing1(j)=toc./500
end
 figure
   n=7:1:12;
   plot(n,timing,'r-*')
   hold on;
   plot(n,timing1,'b-o');
   hold off;
   xlabel('n')
   ylabel('timing')
   legend('LU','A*B')
```

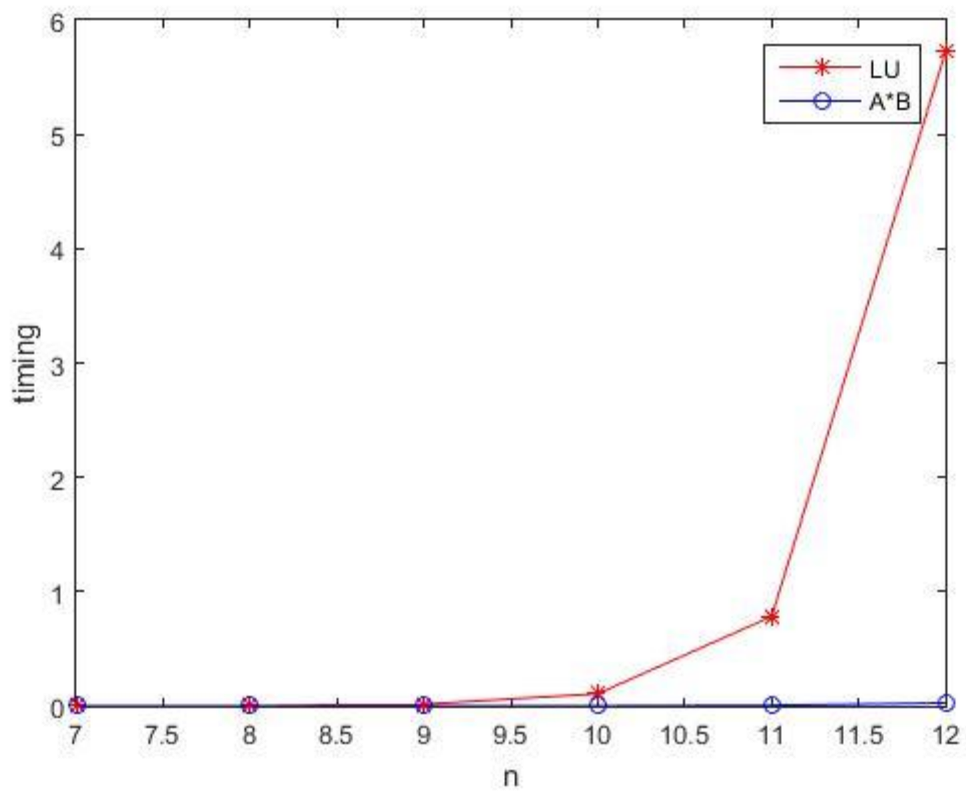
```
timing =

   0.0047   0.0343   0.0749   0.2963   1.2967   5.6973

timing1 =

   0.0000   0.0003   0.0016   0.0042   0.0065   0.0274
```



iii)

```
for j=1:6
   i = j+6;
   n = pow2(i);
   A = (rand(n));
   B = (rand(n,1));
   f = @ () lu(A);
   timing(j) = timeit(f)
   f1 = @ () A*b;
   timing1(j)= timeit(f1)
end
 figure
   n=7:1:12;
   plot(n,timing,'r-*')
   hold on;
   plot(n,timing1,'b-o');
   hold off;
   xlabel('n')
   ylabel('timing')
   legend('LU','A*B')
```

```
timing =

   0.0005    0.0018    0.0165    0.1079    0.7836    5.7205

timing1 =

   0.0000    0.0000    0.0003    0.0017    0.0064    0.0271
```



Κανονικα θα υπολογίζαμε τα Gflops/s:

Για την πράξη A*b : (2n^2 - n)/ time (όπου time ο χρόνος που έκανε η matlab για να υπολογίσει την πράξη και n το μέγεθος του μητρώου για κάθε περίπτωση.)

Για την παραγοντοποίση LU : (2/3)*(n^3)/time (όπου time ο χρόνος που έκανε η matlab για να υπολογίσει την πράξη και n το μέγεθος του μητρώου για κάθε περίπτωση.)

Για τα ακόλουθα ερωτήματα χρειάστηκε να πάρω το n = 2.^[6:11], συμβουλευόμενος τις οδηγίες που δόθηκαν στην καρτέλα συζητήσεις του μαθήματος στο e-class.

ΕΡΩΤΗΜΑ 3

A)

```
for j = 1:1:6
   i = j+5;
   n = pow2(i);
   A = (rand(n));
   B = (rand(n));

   x1 = rand; % tixaios ari8mos gia tis diagwnious
   x2 = rand;
   x3 = rand;
   x4 = rand;
   x5 = rand;
   x6 = rand;

   v1 = repmat(x1,1,n);
   v2 = repmat(x2,1,n-1);
   v3 = repmat(x3,1,n-1);
   v4 = repmat(x4,1,n);
```

```
   v5 = repmat(x5,1,n-1);
   v6 = repmat(x6,1,n-1);

   C = gallery('tridiag',v2,v1,v3)
   D = full(C);
   E = gallery('tridiag',v5,v4,v6)
   G = full(E);

   f = @ () mtimes(A,B); % tixaio mitrwo
   timing0(j) = timeit(f)

   h = @ () mtimes(D,G);
   timing1(j) = timeit(h)

   L = triu(A);
   L1 = triu(B);
   g = @ () mtimes(L,L1); % anw trigwniko
   timing2(j) = timeit(g)

end
 figure
   m = 6:1:11;
   plot(m,timing0,'r-o')
   hold on;
   plot(m,timing1,'b--x');
   plot(m,timing2,'g:*');
   hold off;
   xlabel('m')
   ylabel('timing')
   legend('A*B','D*G','L*L1')

 figure
   m = 6:1:11;
   q = 12:2:22;
   elem = pow2(q);
   gf0 = ((elem*6)./timing0);
   gf1 = ((elem*6)./timing1);
   gf2 = ((elem*6)./timing2);
   plot(timing0,gf0,'r-o')
   hold on;
   plot(timing1,gf1,'b--x');
   plot(timing2,gf2,'g:*');
   hold off;
   ylabel('gf')
   xlabel('timing')
   legend('A*B','D*G','L*L1')
```

timing0 =

   0.0003   0.0020   0.0067   0.0363   0.3011   2.3990   (Χρόνοι για την εκτέλεση mtimes μεταξύ τυχαίων τετραγωνικών μητρώων)

timing1 =

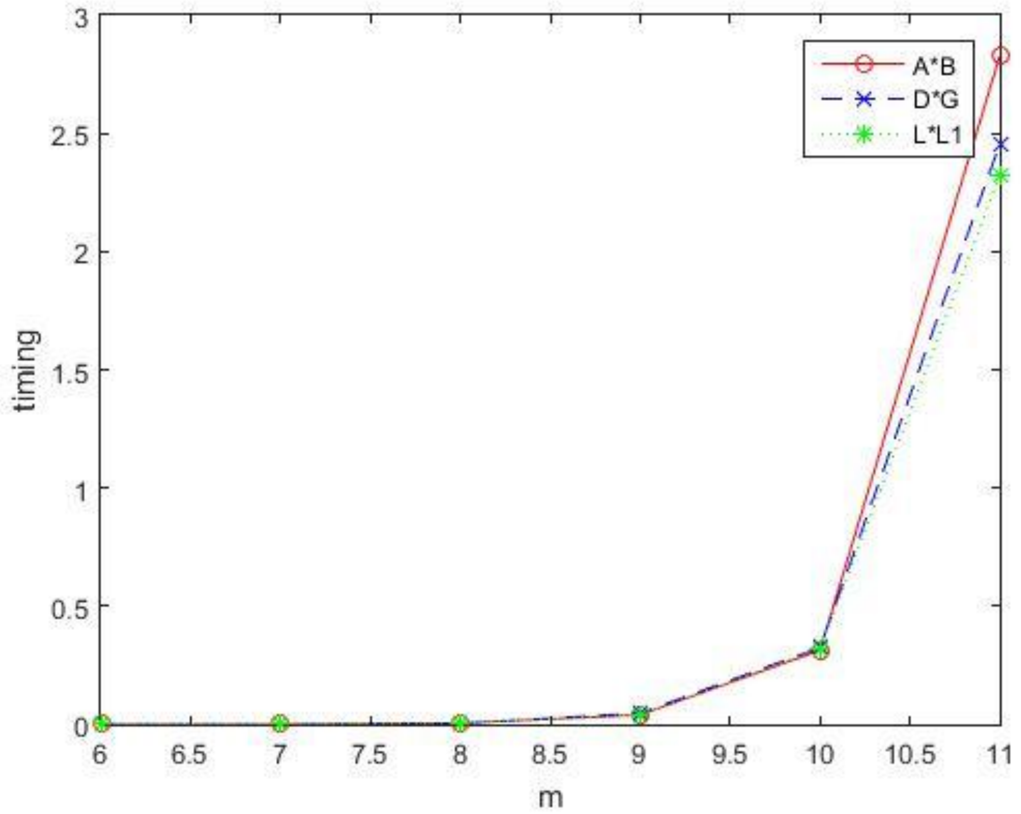
   0.0003   0.0010   0.0061   0.0363   0.2761   2.5299   (Χρόνοι για την εκτέλεση mtimes μεταξύ τριδιαγώνιων μητρώων)

timing2 =

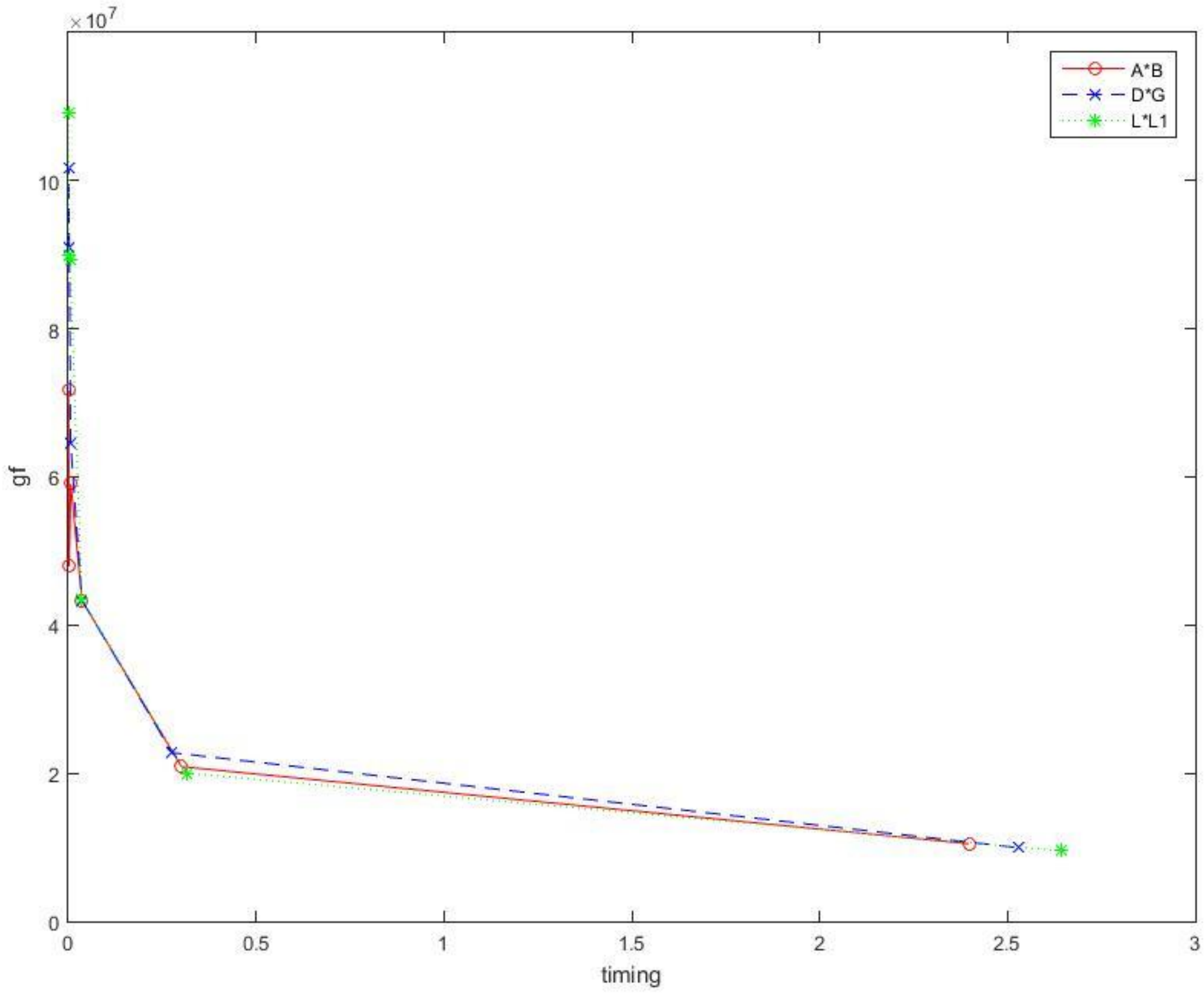
   0.0002   0.0011   0.0044   0.0361   0.3142   2.6434   (Χρόνοι για την εκτέλεση mtimes μεταξύ άνω τριγωνικών μητρώων)

Γ)

Σχεδιάγραμμα Χρόνου



Σχεδιάγραμμα Ρυθμού Εκτέλεσης(Gflops/s)



- **A*B (Τετραγωνικά Μητρώα)**
- **D*G (Τριδιαγώνια Μητρώα)**
- **L*L1 (Άνω Τριγωνικά Μητρώα)**

Δ) Για έναν τυχαίο μητρώο A,B οι χρόνοι εκτέλεσης είναι σχετικά μεγαλύτεροι σε σύγκριση με τα άλλα. Οι χρόνοι εκτέλεσης του άνω τριγωνικού μητρώου είναι αρκετά μειωμένοι λόγω του ότι το συγκεκριμένο μητρώο αποτελείται από σχεδόν όσα μηδενικά όσα και στοιχεία. Το ίδιο ακριβώς ισχύει και για τα τριδιαγώνια καθώς τα στοιχεία τους (μη μηδενικά) είναι μόνο αυτά που βρίσκονται πάνω στην κύρια διαγώνιο και στις 2 εφαπτόμενές της.

ΕΡΩΤΗΜΑ 4

1) Θα πρέπει αρχικα να αναλύσουμε την πολυπλοκότητα του γινομένου: $(uu^T + vv^T)^p*b$

Οι πράξεις uu^T και vv^T ειναι εξωτερικά γινόμενα διανυσμάτων $\in R^n$, από τα οποία προκύπτουν δύο μητρώα (έστω B και C, αντίστοιχα) και ο αριθμός πράξεων που χρειάζεται για τον υπολογισμό κάθε εξωτερικού γινομένου είναι n^2. Οπότε απαιτούνται $2n^2$ πράξεις και επιπλέον n^2 πράξεις για την επακόλουθη πρόσθεση των B και C. Επομένως στην παρένθεση προκύπτει ενα μοναδικό μητρώο A (το αποτέλεσμα της πράξης B + C) με αναγκαίο αριθμό πράξεων <u>$3n^2$</u>.

Σε πιο γενική μορφή δηλαδή: $(A)^p*b$

Το $(A)^p$ αναλύεται στο γινόμενο A*A*A...A*A, p φορές. Για το συγκεκριμένο γινόμενο χρειαζόμαστε p-1 πολλαπλασιασμούς μεταξύ πανομοιότυπων μητρώων. Εκτενέστερα, για τον πολλαπλασιασμό δύο μητρώων εκτελούνται $n^2*(2n-1)$ πράξεις, αλλα εμείς το θέλουμε p-1 φορές, οπότε συνολικά <u>$(p-1)*(2n-1)*n^2$</u>. Τελικώς, προκύπτει ένα μητρώο nxn, όπως το A, το οποίο απλώς μένει να το πολλαπλασιάσουμε με το b. Το αποτέλεσμα θα είναι ένα διάνυσμα nx1, με απαιτούμενο αριθμό πράξεων <u>$n*(2n-1)$</u>.

Άρα, το σύνολο όλων των πράξεων προκύπτει: <u>$3n^2 + (p-1)*(2n-1)*n^2 + n*(2n-1)$</u>.

2) Αν για p=10, τότε η συνάρτηση μετατρέπεται σε: $3n^2 + 9*(2n-1)*n^2 + n*(2n-1)$ = <u>$18n^3 -4n^2 -n$</u>.

Σε κώδικα Matlab η παραπάνω συνάρτηση θα υλοποιούταν ως εξής:

```
function [F1] = rank2_power(u,v,b)
A = u*transpose(u) + v*transpose(v);
F1 = A;
for i=1:9
   F1 = F1*A;
end
F1 = F1*b;
end
```

3) Άν παίρναμε ένα-ένα τα μητρώα και τα πολλαπλασιάζαμε κάθε φορά με το δεξιότερο μέλος της πράξης θα παρατηρούσαμε ότι:

Ως υπενθύμιση και για περισσότερη διάυγεια, έχουμε: <u>A*A*A*A*A*A*A*A*A*A*b</u> (σε πολύ απλοποιημένη μορφή).

Υποθέτοντας, ότι αρχικά εκτελούσαμε τον πολ/σμο A*b θα προέκυπτε ένα διάνυσμα, nx1 διαστάσεων, και με την ίδια ακολουθία, θα είχαμε να κάνουμε πράξεις πάντα μεταξύ μητρώων και νέεων παραγόμενων διανυσμάτων. Φέρνοντάς το εις πέρας, θα πέρναμε το αναμενόμενο διάνυσμα nx1, άλλωστε όπως και στην προηγούμενη εκτέλεση, με τη διαφορά πως εδώ έχουμε δημιουργήσει 10 πράξεις των <u>$n*(2n-1)$</u>.

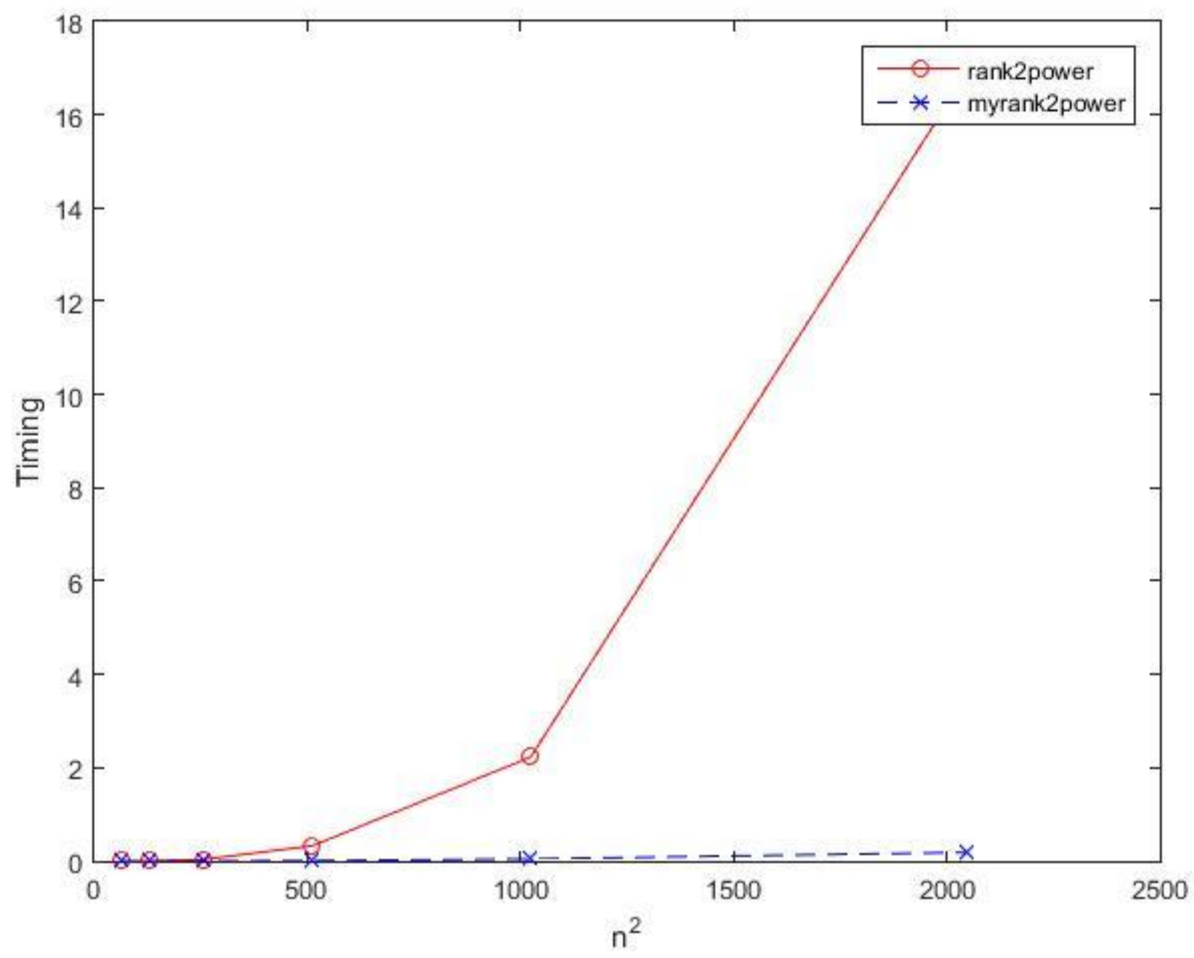
Αναλυτικότερα, ο αριθμός των πράξεων σ'αυτήν την περίπτωση υπολογίζεται σε: <u>$3n^2 + 10n(2n-1)$</u>.

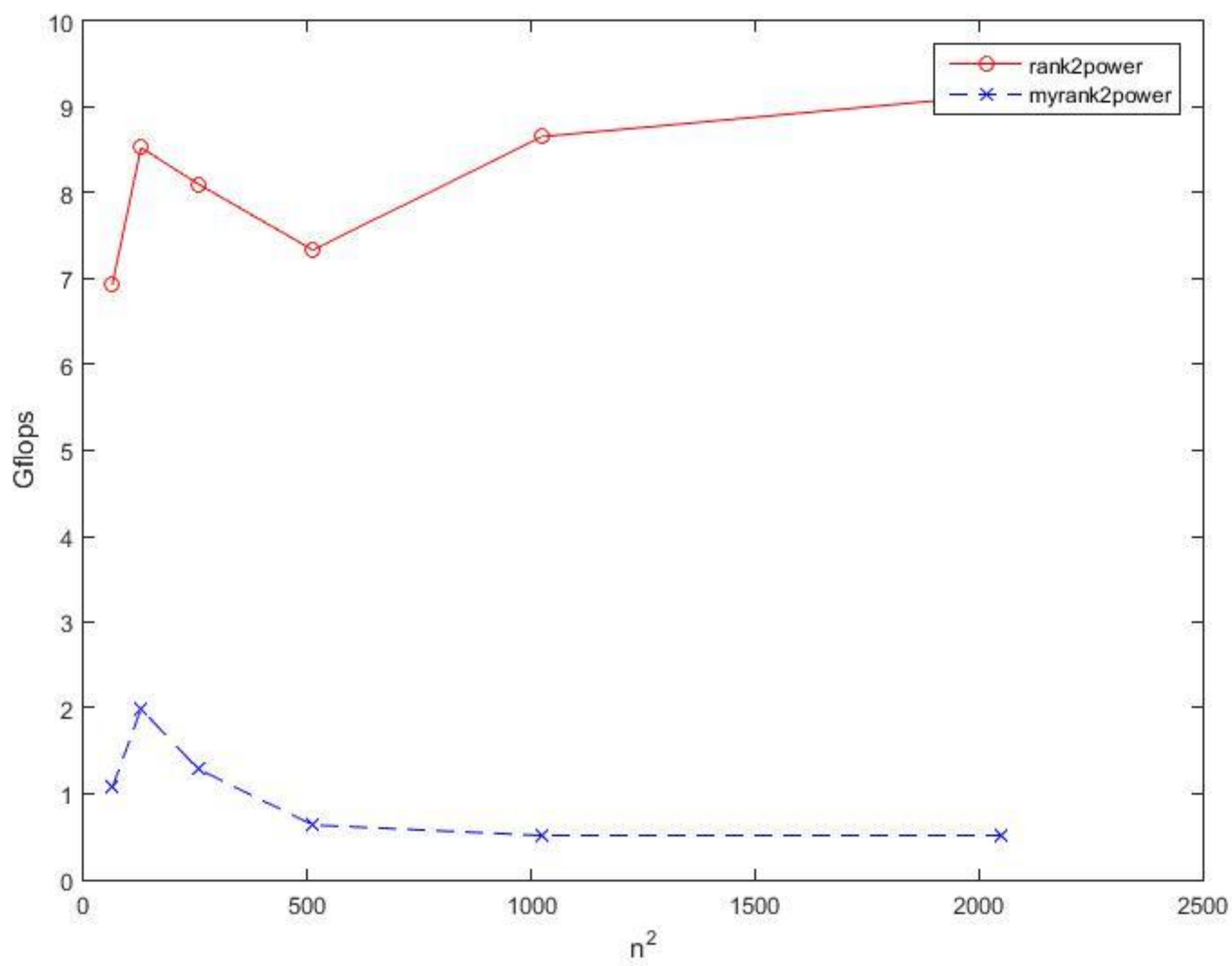
(Δεν πρέπει να ξεχνάμε εδώ, και τις ευνόητες πράξεις μεταξύ των αρχικών διανυσμάτων δηλ. $3n^2$.)

Σε κώδικα Matlab η παραπάνω συνάρτηση θα υλοποιούταν ως εξής:

```
function [F2] = my_rank2_power(u,v,b)
A = u*transpose(u) + v*transpose(v);
F2 = A;
F2 = F2*b;
for i=1:9
   F2 = A*F2;
end
end
```

4) Για n = 2.^[7:12] παρατηρούμε τις επιδόσεις ως έχουν:





Είναι ιδιαίτερα εμφανές ότι καθώς κυλάει ο χρόνος και η πολυπλοκότητα των πράξεων εντείνεται λόγω του αυξανόμενου n, η επίδοση της rank2_power μειώνεται δραματικά σε αντίθεση με της my_rank2_power. Αυτό οφείλεται, αποκλειστικά, στην απλούστευση των πράξεων που πραγματοποιήσαμε παραπάνω για την κλήση της συνάρτησης my_rank2_power.

Στη συνέχεια παραθέτω το main πρόγραμμα της Matlab που έτρεξα, παράλληλα με την κλήση των προαναφερθεισών m-functions(rank2_power και my_rank2_power) :

```
for i=6:11;
   n = pow2(i);
   u = rand(n,1);
   v = rand(n,1);
   b = rand(n,1);


   %ftiaxnoume ta handle gia ka8e sinartisi
   f = @rank2_power;
   g = @my_rank2_power;

   %kai ta antistoixa handle gia tin klisi twn timeit
   tf = @ () f(u,v,b);
   tg = @ () g(u,v,b);

   Ftiming(i-5) = timeit(tf);
   Gtiming(i-5) = timeit(tg);

   %ipologismos twn Flops/s
   Fop(i-5) = 18*(n^3) - 4*(n^2) - n; %ari8mos praksewn rank2_power
   GflopsF(i-5) = (Fop(i-5)./Ftiming(i-5))*(10^-9);
   Gop(i-5) = 23*(n^2) - 10*n; %ari8mos praksewn my_rank2_power
   GflopsG(i-5) = (Gop(i-5)./Gtiming(i-5))*(10^-9);
end

figure

i=6:11;
n = pow2(i);

plot(n,Ftiming,'r-o')
hold on
plot(n,Gtiming,'b--x')
hold off
xlabel('n^2')
ylabel('Timing')
legend('rank2power','myrank2power')

figure

i=6:11;
n = pow2(i);

plot(n,GflopsF,'r-o')
hold on
plot(n,GflopsG,'b--x')
hold off
xlabel('n^2')
ylabel('Gflops')
legend('rank2power','myrank2power')
```