1η Εργασία στον Επιστημονικό Υπολογισμό

Αλεβιζοπούλου Αφροδίτη Α.Μ:3879
alevizopou@ceid.upatras.gr

2 Νοεμβρίου 2014

1 Εισαγωγικά

Παρακάτω περιγράφονται τα χαρακτηριστικά του υπολογιστικού συστήματος το οποίο χρησιμοποιήσαμε για την υλοποίηση της εργαστηριακής άσκησης. Για τον υπολογισμό των χαρακτηριστικών χρησιμοποιήθηκαν δύο ειδικά προγράμματα, το CPU-Z και το PC Wizard.

(i) **Επεξεργαστής:** Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz

Ο εν λόγω επεξεργαστής διαθέτει τέσσερις πυρήνες και η συχνότητα λειτουργίας του είναι 3.30GHz.

Κρυφή Μνήμη: Η κρυφή του μνήμη, μεγέθους 6MB, αποτελείται από τρία επίπεδα, L1, L2, L3, με το πρώτο επίπεδο να χωρίζεται σε κρυφή μνήμη εντολών και κρυφή μνήμη δεδομένων. Συγκεκριμένα:

L1 Data Cache:	32 KBytes, 8-way set associative, 64-byte line size
L1 Instruction Cache:	32 KBytes, 8-way set associative, 64-byte line size
L2 Cache:	256 KBytes, 8-way set associative, 64-byte line size
L3 Cache:	6 MBytes, 12-way set associative, 64-byte line size

Είδος της πολιτικής εγγραφής στην κρυφή μνήμη: Write-Back.

Λειτουργικό σύστημα του υπολογιστή: Windows 7 Professional Service Pack 1 (64-bit).

(ii) Για την υλοποίηση της άσκησης χρησιμοποιήθηκε η έκδοση 8.01 (R2013a) της MATLAB.

(iii) Η διακριτότητα του χρονομετρητή tic/toc, δηλαδή πόσος χρόνος μεσολαβεί μεταξύ του tic και του toc αν δε διεξαχθεί κανένας υπολογισμός, εκτιμάται στα $8.2935e-08$ sec., δηλ. 8.2935×10^{-8} sec. Για την εκτίμηση της διακριτότητας του χρονομετρητή συντάξαμε τον ακόλουθο κώδικα, ο οποίος και εκτελεί τις εντολές 30 φορές.

```
% warm up
tic;
toc;

% Preallocation
t = zeros(30,1);

for i = 1:30
    tic;
    t(i) = toc;
end

time = sum(t)./30
```

(iv) Το αποτέλεσμα της συνάρτησης bench για τη διάσπαση μητρώων LU είναι: 0.0470 sec.

2 Χρονομέτρηση Συναρτήσεων

(i) Η συνάρτηση **lu** υπολογίζει την παραγοντοποίηση LU ενός μητρώου. Η παραγοντοποίηση LU αναλύει ένα μητρώο σε γινόμενο ενός κάτω τριγωνικού μητρώου επί ένα άνω τριγωνικό. Το γινόμενο μπορεί να περιλαμβάνει επιπλέον και ένα μητρώο διάταξης. Η παραγοντοποίηση $L * U$ έχει ως προϋπόθεση να μην υπάρχει καμία εναλλαγή γραμμών καθώς η απαλοιφή Gauss ανάγει το A σε U. Αν ισχύει αυτή η προϋπόθεση, τότε το κάτω τριγωνικό μητρώο *L* με πολλαπλασιαστές l_{ij} (και $l_{ii} = 1$) ανακτά το U από το A. Το μητρώο διάταξης (μετάθεσης) P χρησιμοποιείται ώστε να αποφευχθούν μηδενικά στις θέσεις των οδηγών. Η χρήση του έχει ως προϋπόθεση ο *A* να είναι αντιστρέψιμος. Συνεπώς, τα P, L, U είναι αντιστρέψιμα. Το P διεξάγει εκ των προτέρων τις εναλλαγές γραμμών. Η δεύτερη παραγοντοποίηση είναι η $P * A = L * U$.

Η συνάρτηση **qr** υπολογίζει την παραγοντοποίηση QR (Orthogonal-triangular decomposition). Η παραγοντοποίηση QR (Ορθογωνιοποίηση Gram-Schmidt) αναλύει ένα μητρώο σε ένα ορθογώνιο μητρώο Q και ένα άνω τριγωνικό μητρώο R. Χρησιμοποιείται συχνά για να λύσει το γραμμικό πρόβλημα των ελαχίστων τετραγώνων. Προϋπόθεση σε αυτή την παραγοντοποίηση είναι το A να έχει ανεξάρτητες στήλες. Αυτές βρίσκονται ορθογωνιοποιημένες στο Q μέσω της διαδικασίας Gram-Schmidt. Αν το A είναι τετραγωνικό, τότε $Q^{-1} = Q^T$.

Η συνάρτηση **svd** υπολογίζει μια σημαντική παραγοντοποίηση ενός πραγματικού ή μιγαδικού μητρώου, την Ανάλυση Ιδιαζουσών Τιμών (Singular Value Decomposition). Ένα μητρώο μπορεί να αναλυθεί ως το γινόμενο τριών μητρώων - ενός ορθογώνιου μητρώου *U*, ενός διαγώνιου μητρώου *S* και του αναστρόφου ενός ορθογώνιου μητρώου V. Οι πρώτες r στήλες των U και V αποτελούν ορθοκανονικές βάσεις των $C(A)$ και $C(A^T)$ με $A * v_i = s_i * u_i$ και ιδιάζουσα τιμή $s_i > 0$. Οι τελευταίες στήλες των U και V αποτελούν ορθοκανονικές βάσεις των μηδενοχώρων των A^T και A.

Η συνάρτηση **det** υπολογίζει την ορίζουσα ενός τετραγωνικού μητρώου. Η ορίζουσα ενός τετραγωνικού μητρώου είναι ένας μόνο αριθμός που, όμως, παρέχει σημαντικές πληροφορίες όταν ο πίνακας αποτελείται από τους συντελεστές ενός συστήματος γραμμικών εξισώσεων, ή όταν αντιστοιχεί σε ένα γραμμικό μετασχηματισμό ενός διανυσματικού χώρου. Στην πρώτη περίπτωση το σύστημα έχει μοναδική λύση ακριβώς όταν η ορίζουσα είναι μη μηδενική, ενώ όταν η ορίζουσα είναι μηδέν είτε δεν υπάρχουν λύσεις είτε υπάρχουν πολλές. Στη δεύτερη περίπτωση, για τις ίδιες συνθήκες, σημαίνει ότι για τον μετασχηματισμό ορίζεται η αντίστροφη πράξη.

Η συνάρτηση **rank** παρέχει μια εκτίμηση του αριθμού των γραμμικά ανεξάρτητων γραμμών ή στηλών ενός μητρώου. Το αληθινό μέγεθος ενός μητρώου A δίνεται από την τάξη του, δηλαδή το πλήθος των οδηγών. Αυτός είναι ο αριθμός r που επιστρέφεται από τη συνάρτηση rank. Έτσι, για να εκτελεστεί η εντολή r = rank(A), ο υπολογιστής απλά καταμετρά τους οδηγούς. Ένα μητρώο A έχει πλήρη τάξη γραμμών εάν κάθε γραμμή έχει έναν οδηγό, δηλαδή δεν υπάρχουν μηδενικές γραμμές στο αναγμένων γραμμών κλιμακωτής μορφής μητρώο R. Ένα μητρώο A έχει πλήρη τάξη στηλών εάν κάθε στήλη έχει έναν οδηγό, δηλαδή δεν υπάρχουν ελεύθερες μεταβλητές.

** Οι πηγές των παραπάνω πληροφοριών ήταν η Wikipedia, το documentation της Matlab και το βιβλίο ¨Εισαγωγή στη Γραμμική Άλγεβρα¨ του Gilbert Strang.*

(ii) Για τυχαία μητρώα $A \in \mathbb{R}^{n\times n}$ και $b \in \mathbb{R}^{n\times 1}$, όπου $n = 2.^{[7:10]}$, μετρήσαμε το χρόνο εκτέλεσης για τη συνάρτηση lu(A) και την πράξη A∗b.

(α′) Χρησιμοποιώντας τις συναρτήσεις tic, toc εκτελέσαμε κάθε πράξη μόνο μία φορά. Παραθέτουμε τον κώδικα που εκτελέστηκε (αρχείο ex2a.m):

```
% Preallocation
time = zeros(4,2);
k = 0;

% for-loop for matrices with size 2.^[7:10]
for n = 7:1:10

    % variable k is used in order to access matrix time
    k = k + 1;

    A = rand(2.^n);
    b = rand(2.^n,1);

    % These executions are not considered
    tic;
    toc;
    [L,U] = lu(A);
    y = A*b;

    % counting lu(X)
    tic;
    [L,U] = lu(A);
    time(k,1) = toc;

    % counting A*b
    tic;
    y = A*b;
    time(k,2) = toc;

end

time

T = [128 256 512 1024];
figure
%plot(T,time(:,1),'go-','linewidth',1,'markersize',10)
semilogy(T,time(:,1),'go-','linewidth',1,'markersize',10);
hold on
%plot(T,time(:,2),'rx-','linewidth',1,'markersize',10)
semilogy(T,time(:,2),'rx-','linewidth',1,'markersize',10);
legend('lu(A)','A*b','Location','NorthWest')
xlabel('Matrix Dimensions')
ylabel('Time (sec)')
title('2.ii.a')
grid on
print(gcf, '-djpeg', 'ex2iia.jpg')
hold off
```

Προέκυψαν οι ακόλουθοι χρόνοι:

Μέγεθος/Πράξη	**lu(A)**	**A*b**
128	0.0005	0.0000
256	0.0015	0.0000
512	0.0036	0.0004
1024	0.0185	0.0016

Πίνακας 1: Single execution

(β') Χρησιμοποιώντας τις συναρτήσεις tic, toc, εκτελέσαμε κάθε πράξη αρκετές φορές. Όσον αφορά στον τρόπο με τον οποίο έγιναν οι μετρήσεις, προτιμήσαμε σε πράξεις που εκτελούνται σε σύντομο χρονικό διάστημα να κάνουμε πολλές επαναλήψεις ώστε να περιορίσουμε το σχετικό σφάλμα, ενώ σε πράξεις που ολοκληρώνονται σε μεγαλύτερο χρόνο ο αριθμός των επαναλήψεων να είναι μικρότερος. Παραθέτουμε τον κώδικα που εκτελέστηκε (αρχείο ex2b.m):

```
% Preallocation
time1 = zeros(4,1);
time2 = zeros(4,1);
k = 0;

% for-loop for matrices with size 2.^[7:10]
for n = 7:1:10

    % variable k is used in order to access matrix time
    k = k + 1;

    A = rand(2.^n);
    b = rand(2.^n,1);

    % These executions are not considered
    tic;
    toc;
    [L,U] = lu(A);
    y = A*b;

    % counting the execution time of lu(X)
    elapsed = 0;
    for i = 1:1:20
        % counting lu(X)
        tic;
        [L,U] = lu(A);
        elapsed = elapsed + toc;
    end
    time1(k) = elapsed/20;

    % counting the execution time of A*b
    elapsed = 0;
    for i = 1:1:30
        % counting A*b
        tic;
        y = A*b;
        elapsed = elapsed + toc;
    end
    time2(k) = elapsed/30;
end

time1
time2

T = [128 256 512 1024];
figure
semilogy(T,time1,'go-','linewidth',1,'markersize',10);
%plot(T,time1,'go-','linewidth',1,'markersize',10);
hold on
semilogy(T,time2,'rx-','linewidth',1,'markersize',10);
%plot(T,time2,'rx-','linewidth',1,'markersize',10);
legend('lu(A)','A*b','Location','NorthWest')
xlabel('Matrix Dimensions')
ylabel('Time (sec)')
title('2.ii.b')
grid on
print(gcf, '-djpeg', 'ex2iib.jpg')
hold off
```

Προέκυψαν οι ακόλουθοι χρόνοι:

Μέγεθος/Πράξη	**lu(A)**	**A*b**
128	0.0004	0.0000
256	0.0007	0.0000
512	0.0038	0.0000
1024	0.0179	0.0004

Πίνακας 2: Multiple execution

(γ') Χρησιμοποιώντας τη συνάρτηση timeit της MATLAB. Παραθέτουμε τον κώδικα που εκτελέστηκε (αρχείο ex2c.m):

```
% Preallocation
time = zeros(4,2);
k = 0;

% for-loop for matrices with size 2.^[7:10]
for n = 7:1:10

    % variable k is used in order to access matrix time
    k = k + 1;

    A = rand(2.^n);
    b = rand(2.^n,1);

    % counting lu(X)
    flu = @() lu(A);
    time(k,1) = timeit(flu, 2);

    % counting A*b
    f = @() A*b;
    time(k,2) = timeit(f, 1);

end

time

T = [128 256 512 1024];
figure
semilogy(T,time(:,1),'go-','linewidth',1,'markersize',10);
%plot(T,time(:,1),'go-','linewidth',1,'markersize',10);
hold on
semilogy(T,time(:,2),'rx-','linewidth',1,'markersize',10);
%plot(T,time(:,2),'rx-','linewidth',1,'markersize',10);
legend('lu(A)','A*b','Location','NorthWest')
xlabel('Matrix Dimensions')
ylabel('Time (sec)')
title('2.ii.c')
grid on
print(gcf, '-djpeg', 'ex2iic.jpg')
hold off
```

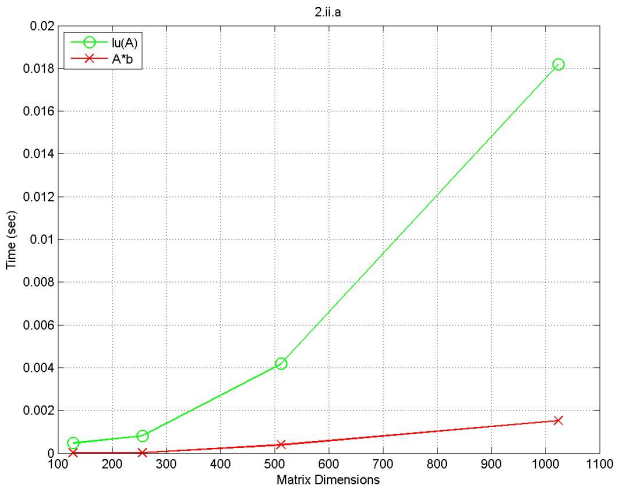
Προέκυψαν οι ακόλουθοι χρόνοι:

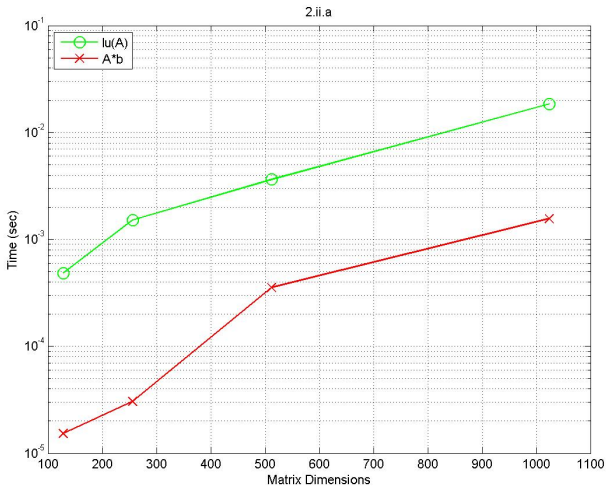
Μέγεθος/Πράξη	**lu(A)**	**A*b**
128	0.0002	0.0000
256	0.0007	0.0000
512	0.0036	0.0000
1024	0.0178	0.0004

Πίνακας 3: Using timeit

(iii) Παρακάτω οπτικοποιούμε τα αποτελέσματά μας για τις διάφορες πράξεις σε κοινή γραφική παράσταση για καθέναν από τους παραπάνω τρόπους. Παραθέτουμε δύο γραφικές αναπαραστάσεις για την κάθε περίπτωση, τόσο σε απλή όσο και σε λογαριθμική αναπαράσταση στον άξονα των y (εκτελώντας τον κώδικα με semilog αντί για plot) για μεγαλύτερη ευκρίνεια:

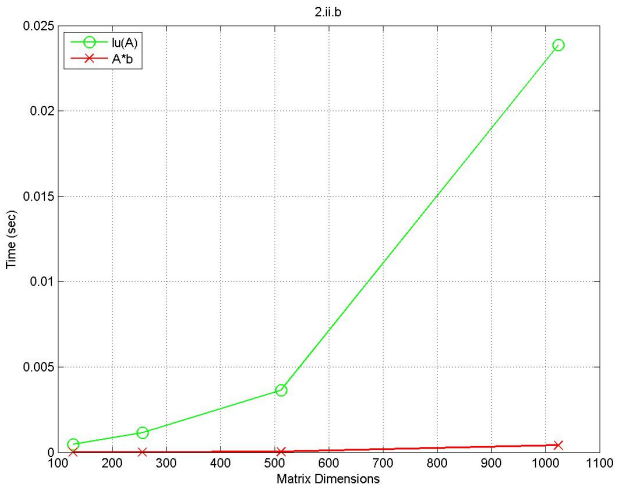
Εκτελώντας κάθε πράξη μόνο μία φορά προέκυψαν οι ακόλουθες γραφικές παραστάσεις:

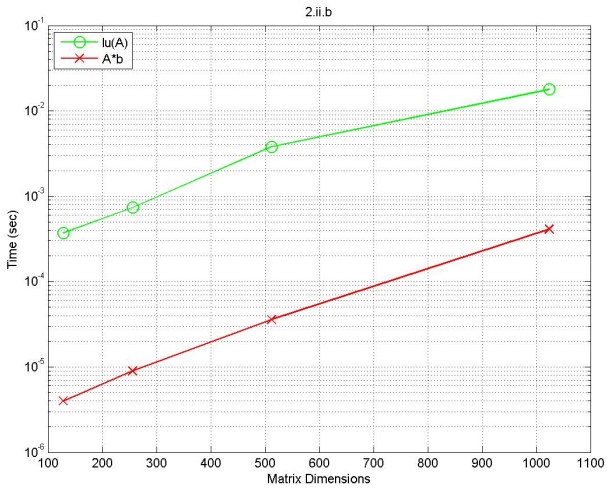




Σχήμα 1: Single execution

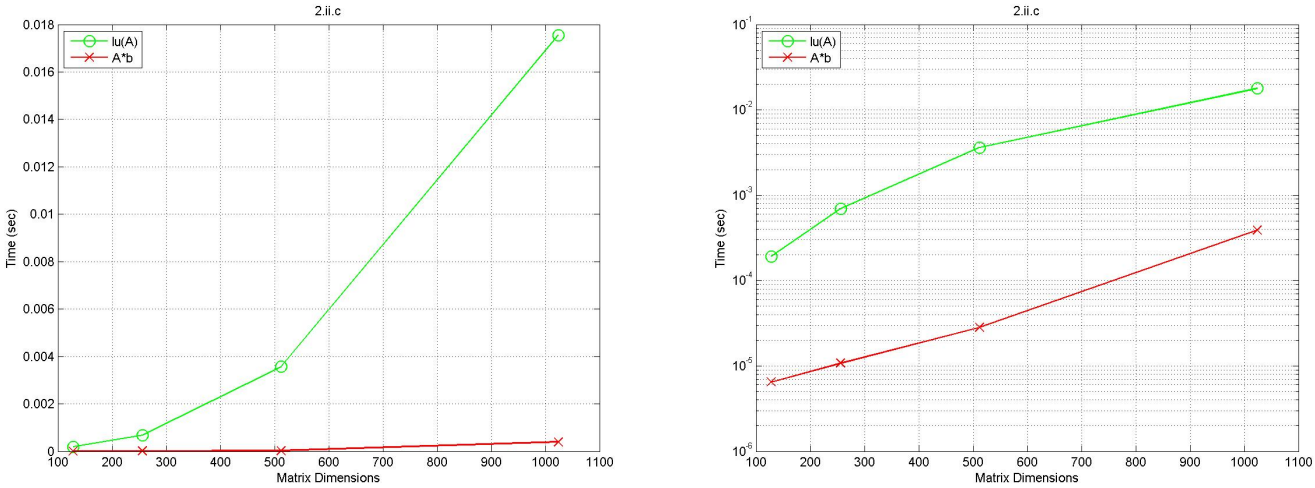
Εκτελώντας κάθε πράξη περισσότερες φορές προέκυψαν οι ακόλουθες γραφικές παραστάσεις:





Σχήμα 2: Multiple execution

Χρησιμοποιώντας τη συνάρτηση timeit προέκυψαν οι ακόλουθες γραφικές παραστάσεις:



Σχήμα 3: Using timeit

Παρατηρούμε ότι η πράξη A∗b χρειάζεται το λιγότερο χρόνο σε κάθε μέτρηση που κάνουμε, ενώ η συνάρτηση lu(A) σημαντικά μεγαλύτερο. Επίσης, παρατηρούμε πως ενώ ναι μεν υπάρχει αύξηση του χρόνου τόσο στη συνάρτηση lu(A) όσο και στην πράξη A∗b καθώς αυξάνεται το μέγεθος του n, ο ρυθμός αύξησης του χρόνου στην lu είναι πολύ μεγαλύτερος. Επίσης, άξια σχολιασμού είναι και τα αποτελέσματα των μετρήσεων που παίρνουμε και φαίνονται και στα figures. Όπως γνωρίζουμε, η πρώτη μέτρηση για μία πράξη δεν είναι φερέγγυα και μία φορά δεν αρκεί. Έτσι, λοιπόν, μπορεί να δικαιολογηθεί το ¨λάθος¨ των χρόνων στην πρώτη περίπτωση. Με το δεύτερο τρόπο μετρήσεων (εκτελώντας κάθε πράξη περισσότερες φορές) όχι μόνο μετριάζεται το ¨λάθος¨, αλλά παρατηρούμε πως οι χρόνοι σχεδόν ταυτίζονται με αυτούς της εκτέλεσης με την timeit. Επομένως, οδηγούμαστε στο συμπέρασμα πως κάναμε καλή επιλογή αριθμού επαναλήψεων τόσο για τη συνάρτηση lu(A) όσο και για την πράξη A∗b, γεγονός που μας οδήγησε σε μέσους χρόνους όπως τους αναμέναμε και θεωρητικά. Τέλος, τη μεγαλύτερη ακρίβεια στους χρόνους εκτέλεσης την παίρνουμε στην υλοποίηση με την timeit.

3 Αξιολόγηση Ενδογενών Συναρτήσεων

(α΄) Χρονομετρήσαμε τη συνάρτηση mldivide χρησιμοποιώντας τη συνάρτηση timeit. Παραθέτουμε τον κώδικα που εκτελέστηκε και για τους τρεις τύπους μητρώων (αρχείο ex3.m):

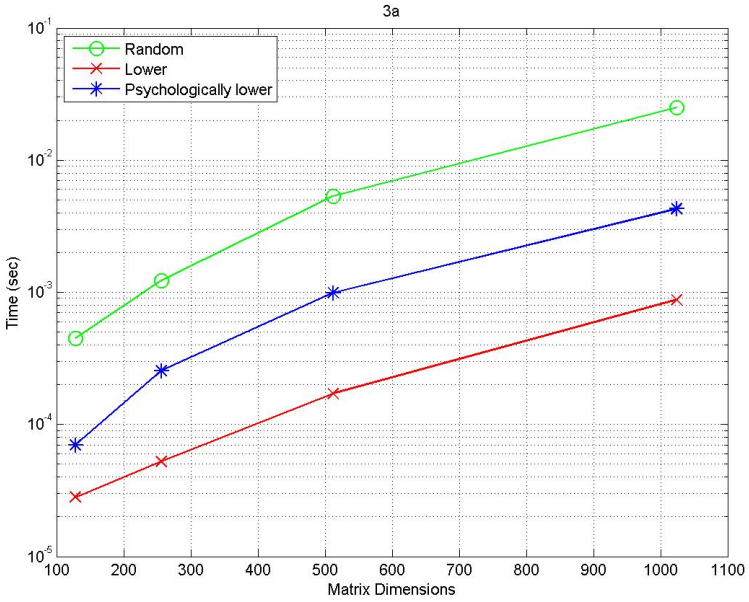
```
% Preallocation
time = zeros(4,3);
k = 0;

% for-loop for matrices with size 2.^[7:10]
for n = 7:1:10

    % variable k is used in order to access matrix time
    k = k + 1;

    A = rand(2.^n);
    b = rand(2.^n,1);

    % for random matrices
    f = @() mldivide(A, b);
    time(k,1) = timeit(f,1);

    % for lower triangular matrices
    L = tril(A);
    f = @() mldivide(L,b);
    time(k,2) = timeit(f,1);

    % for psychologically lower triangular matrices
    P = L(:,randperm(end));
    f = @() mldivide(P,b);
    time(k,3) = timeit(f,1);

end

time

T = [128 256 512 1024];
figure
semilogy(T,time(:,1),'go-','linewidth',1,'markersize',10);
hold on
semilogy(T,time(:,2),'rx-','linewidth',1,'markersize',10);
hold on
semilogy(T,time(:,3),'b*-','linewidth',1,'markersize',10);
legend ('Random', 'Lower', 'Psychologically lower', 'Location', 'NorthWest')
xlabel('Matrix Dimensions')
ylabel('Time (sec)')
title ('3a')
grid on
print(gcf, '-djpeg', 'ex3a.jpg')
hold off
```

Προέκυψαν οι ακόλουθοι χρόνοι:

Μέγεθος/Μητρώο	**random**	**lower**	**psych. lower**
128	0.0004	0.0000	0.0001
256	0.0012	0.0001	0.0003
512	0.0053	0.0002	0.0010
1024	0.0249	0.0009	0.0043

Πίνακας 4: Χρονομέτρηση της mldivide

(β΄) Παρουσίαση των αποτελεσμάτων σε μία κοινή γραφική παράσταση χρησιμοποιώντας λογαριθμική κλίμακα στον άξονα των y για μεγαλύτερη ευκρίνεια:



Σχήμα 4: Κοινό γράφημα για χρονικές μετρήσεις της mldivide

(γ') Από τη χρονομέτρηση παρατηρούμε ότι τα κάτω τριγωνικά μητρώα επιτυγχάνουν τους καλύτερους χρόνους, γεγονός που μας δείχνει ότι η δομή των μητρώων παίζει ρόλο στο χρόνο εκτέλεσης πράξεων που τα χρησιμοποιούν. Επομένως, είναι λογικό τα κάτω τριγωνικά μητρώα να επιτυγχάνουν τους καλύτερους χρόνους, μετά να έρχονται τα ψυχολογικά κάτω τριγωνικά μητρώα και τέλος τα τυχαία, καθώς τα πρώτα έχουν πιο αυστηρή δομή από τα δεύτερα και ίσο αριθμό μηδενικών στοιχείων και τα δεύτερα έχουν μεγαλύτερο αριθμό μηδενικών στοιχείων από τα τρίτα.

(δ') Εκτελέσαμε τις πράξεις $I \cdot A, A \cdot A$, όπου $I \in \mathbb{R}^{n \times n}$ είναι το ταυτοτικό μητρώο και $A \in \mathbb{R}^{n \times n}$ τυχαίο μητρώο (αρχείο ex3d.m):

```
% Preallocation
time = zeros(4,2);
k = 0;

% for-loop for matrices with size 2.^[7:10]
for n = 7:1:10

    % variable k is used in order to access matrix time
    k = k + 1;

    A = rand(2.^n);
    b = rand(2.^n,1);
    I = eye(2.^n);

    f = @() I*A;
    time(k,1) = timeit(f, 1);

    f = @() A*A;
    time(k,2) = timeit(f, 1);

end

time

T = [128 256 512 1024];
figure
plot(T,time(:,1),'go-','linewidth',1,'markersize',10);
hold on
plot(T,time(:,2),'rx-','linewidth',1,'markersize',10);
```

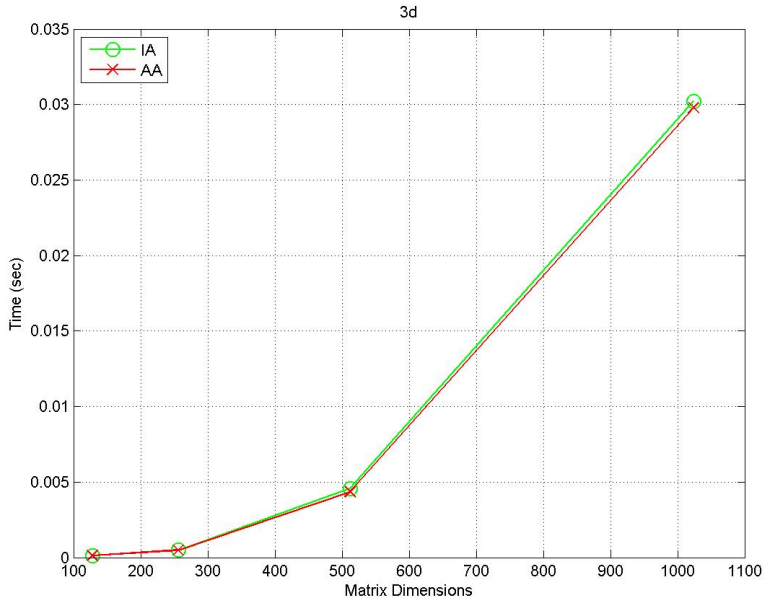
```
legend ('IA', 'AA', 'Location', 'NorthWest')
xlabel('Matrix Dimensions')
ylabel('Time (sec)')
title ('3d')
grid on
print(gcf, '-djpeg', 'ex3d.jpg')
hold off
```

Προέκυψαν οι ακόλουθοι χρόνοι:

Μέγεθος/Πράξη	IA	AA
128	0.0001	0.0001
256	0.0005	0.0005
512	0.0046	0.0044
1024	0.0302	0.0298

Πίνακας 5: Χρόνοι εκτέλεσης για τις πράξεις $I \cdot A, A \cdot A$

Παρουσίαση των αποτελεσμάτων σε μία κοινή γραφική παράσταση:



Σχήμα 5: Γράφημα χρονικών μετρήσεων για τις πράξεις $I \cdot A, A \cdot A$

Παρατηρούμε πως οι χρόνοι εκτέλεσης είναι πολύ κοντά. Επομένως η ειδική μορφή του μητρώου I δεν έπαιξε ρόλο. Παρατηρούμε αυτή τη διαφορά μεταξύ των συμπερασμάτων των ερωτημάτων (γ') και (δ') επειδή στο (γ') χρησιμοποιούμε έτοιμη συνάρτηση του MATLAB, ενώ στο (δ') εκτελούμε απλά την πράξη. Οι έτοιμες συναρτήσεις του MATLAB πολλές φορές είναι πιο γρήγορες από τις απλές πράξεις, καθώς εμπεριέχουν βελτιστοποιημένο κώδικα που εκμεταλλεύεται την ιδιαιτερότητα κάποιων μητρώων.

4 Σύγκριση Υλοποιήσεων

(1) Πρώτα θα γίνουν οι πράξεις στην παρένθεση, δηλ. $(I - uv^T)$, για τις οποίες χρειαζόμαστε: n^2 πράξεις για τον πολλαπλασιασμό και n^2 για την αφαίρεση, άρα $2n^2$.

Η πράξη $(I - uv^T)$ παράγει μητρώο $n \times n$, άρα θα γίνουν $p - 1$ πολλαπλασιασμοί μητρώων $n \times n$. Συνεπώς, ο πολλαπλασιασμός μητρώων χρειάζεται για το ένα στοιχείο n πολλαπλασιασμούς και $n - 1$ προσθέσεις, άρα για τα n^2 στοιχεία χρειάζονται $n^2(2n-1)$. Αυτές οι πράξεις θα γίνουν $p-1$ φορές, άρα χρειάζονται $n^2(2n - 1)(p - 1)$ πράξεις.

Επομένως, θεωρητικά το απαιτούμενο πλήθος πράξεων κινητής υποδιαστολής είναι:
$2n^2+n^2(2n-1)(p-1) = 2n^2+(2n^3-n^2)(p-1) = 2n^2+2pn^3-2n^3-pn^2+n^2 = (2p-2)n^3+(3-p)n^2$

(2) Αν θέλουμε να υπολογίσουμε μόνο μια στήλη του πίνακα, τότε χρειαζόμαστε τις $2n^2$ πράξεις του υπολογισμού $(I - uv^T)$ και στη συνέχεια χρειαζόμαστε τις πράξεις μητρώο επί διάνυσμα. Δηλαδή, n πολλαπλασιασμούς, $n - 1$ προσθέσεις επί n στοιχεία, άρα $(2n - 1)n$. Στη συνέχεια θα έχουμε p πολλαπλασιασμούς διάνυσμα επί μητρώο άρα $p * n(2n - 1)$ πράξεις.

Επομένως, ο αριθμός των πράξεων με σειρά από δεξιά προς τα αριστερά θα είναι: $2n^2 + pn(2n - 1)$

Ο λόγος που μειώνεται ο αριθμός των πράξεων είναι ότι αντί να εκτελούμε πολλαπλασιασμό μητρώου με μητρώο εκτελούμε πολλαπλασιασμό μητρώου με διάνυσμα.

(3) Υλοποιήσαμε τη συνάρτηση με όνομα my_func (αρχείο my_func.m):

```
1
2  function [ output_args ] = my_func(varargin)
3
4  if length(varargin) < 3
5      disp('Few arguments!');
6  elseif length(varargin) == 3
7          p = varargin{1};
8          u = varargin{2};
9          v = varargin{3};
10         n = length(u);
11         if n == length(v)
12             I = eye(n);
13             A = (I-u*transpose(v))^p
14         else
15             disp('Wrong vectors dimensions!');
16         end
17 elseif length(varargin) == 4
18         p = varargin{1};
19         u = varargin{2};
20         v = varargin{3};
21         col = varargin{4};
22         n = length(u);
23         if n == length(v)
24             I = eye(n);
25             e = I(:,col);
26             A = I-u*transpose(v);
27             b = A*e;
28             if p >= 2
29                 for i = 2:p,
30                     b = A*b;
31                 end
32                 b
33             else
34                 b
35             end
36         else
37             disp('Wrong vectors dimensions!');
38         end
39 else
40     disp('Too many arguments!');
41 end
42
43 end
```

(4) Υλοποιήσαμε τη συνάρτηση με όνομα test (αρχείο test.m). Ως ορίσματα της test χρησιμοποιήσαμε τις τιμές $p = 3$ και $col = 1$. Η τιμή του p που χρησιμοποιήσαμε είναι αρκετά μικρή ώστε να μην καθυστερεί υπερβολικά η εκτέλεση στο σύστημά σας και αρκετά μεγάλη ώστε να είναι χρονομετρήσιμη η διαφορά των δύο περιπτώσεων 4i, 4ii.

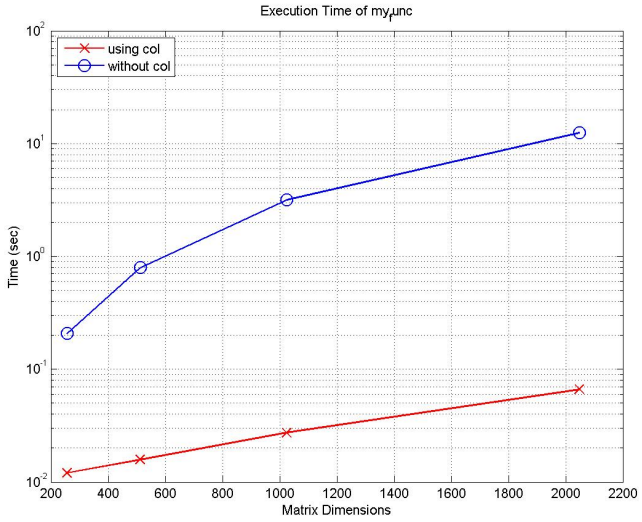
```
function [ output_args ] = test(p, col)

n = 2.^[8:11];

for i = 1:length(n)
    u = rand(n(i),1);
    v = rand(n(i),1);

    %time using col
    f = @() my_func(p,u,v,col);
    tcol(i) = timeit(f);

    %Mflop/s using col
    m1(i) = (2*n(i)^2 + p*n(i)*(2*n(i)-1))/(tcol(i)*10^6);

    %time without col
    f = @() my_func(p,u,v);
    t(i) = timeit(f);

    %Mflop/s without col
    m2(i) = (2*n(i)^2 + (2*n(i)-1)*n(i)^2*(p-1))/(t(i)*10^6);
end

tcol
t
m1
m2

figure(1);
%plot(n,tcol,'rx-',n,t,'bo-','linewidth',1,'markersize',10);
semilogy(n,tcol,'rx-',n,t,'bo-','linewidth',1,'markersize',10);
legend ('using col', 'without col', 'Location', 'NorthWest')
xlabel('Matrix Dimensions')
ylabel('Time (sec)')
title ('Execution Time of my_func')
grid on
print(gcf, '-djpeg', 'ex4da.jpg')
hold off

figure(2);
plot(n,m1,'rx-',n,m2,'bo-','linewidth',1,'markersize',10);
legend ('using col', 'without col', 'Location', 'NorthWest')
xlabel('Matrix Dimensions')
ylabel('Mflop/s')
title ('Mflop/s')
grid on
print(gcf, '-djpeg', 'ex4db.jpg')
hold off
```

Προέκυψαν οι ακόλουθοι χρόνοι:

Μέγεθος/Πράξη	**tcol**	**t**	**m1 (Mflops/s)**	**m2 (Mflops/s)**
256	0.0120	0.2073	43.4673	0.3238×10^3
512	0.0158	0.7950	132.4905	0.6753×10^3
1024	0.0274	3.1693	306.1367	1.3552×10^3
2048	0.0663	12.4623	506.3486	2.7571×10^3

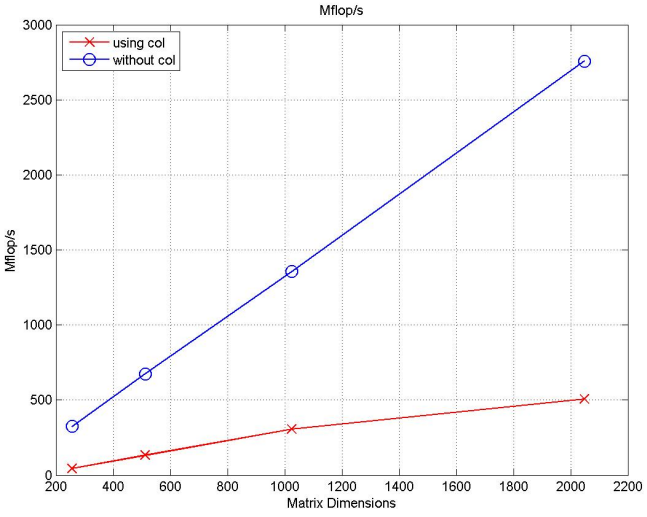
Πίνακας 6: Μετρήσεις με και χωρίς το όρισμα col

(α') Παρουσίαση των χρονομετρήσεων σε μία κοινή γραφική παράσταση, χρησιμοποιώντας λογαριθμική κλίμακα στον άξονα y για μεγαλύτερη ευκρίνεια:



Σχήμα 6: Κοινό γράφημα για χρονικές μετρήσεις

(β') Παρουσίαση των Mflop/s σε μία κοινή γραφική παράσταση:



Σχήμα 7: Κοινό γράφημα για Mflop/s

Εκτελώντας το test.m παίρνουμε τις παραπάνω γραφικές παραστάσεις, όπου παρατηρούμε τα εξής:

• Προφανώς η δεύτερη περίπτωση, όπου δε χρησιμοποιούμε το όρισμα col, χρειάζεται περισσότερο χρόνο για να εκτελεστεί σε σχέση με την πρώτη περίπτωση (χρησιμοποιώντας το όρισμα col), αφού έχουμε μεγαλύτερο όγκο δεδομένων (διάνυσμα vs. πίνακας) και για τον ίδιο λόγο έχουμε και περισσότερες πράξεις το δευτερόλεπτο.

• Όμως, όσο μεγαλύτερο είναι το μητρώο, τόσο οι πράξεις ανά δευτερόλεπτο αυξάνονται! Αυτό συμβάνει γιατί τα μητρώα έχουν καλύτερη τοπικότητα σε σχέση με τα διανύσματα κι έτσι μειώνονται οι μεταφορές όσο μεγαλώνει το μέγεθός τους.