Επιστημονικός Υπολογισμός 1

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Πανεπιστήμιο Πατρών

1η Εργαστηριακή Άσκηση

Created with LaTeX

Author:
Δούκας Ιωάννης

Αριθμός Μητρώου:
5509

Περιεχόμενα

1 Χαρακτηριστικά Υπολογιστικού Συστήματος

Τα χαρακτηριστικά του υπολογιστικού συστήματος στο οποίο υλοποιήθηκαν οι ασκήσεις στο περιβάλλον matlab περιγράφονται στον παρακάτω πίνακα.

Model Name:	MacBook Pro 9.2
Processor:	Intel Core i5 2.5GHz (dual core)
L2 Cache (per Core):	256 KB
L3 Cache:	3 MB
Merory (Ram):	4 GB
Operating System:	OS X 10.9.5
Matlab Version:	8.3.0.532 (R2014a) 64-bit
Διακριτότητα Χρονομετρητή:	2.9254e-07
Αποτέλεσμα LU από εντολή bench:	0.1141

Παρατήρηση: Η διακριτότητα του χρόνου βρέθηκε εκτελώντας το παρακάτω script.

```
nmax = 100:100:100000;
t=0;
a = 0; b=0; c=0;
for i=0:length(nmax)
    tic;
    a = b+c;
    t = t + toc;
end
t = t/length(nmax)
```

2 Χρονομέτρηση Συναρτήσεων

Σε αυτό το ερώτημα θα αξιολογήσουμε την επίδοση ενδογενών συναρτήσεων της Matlab και θα δούμε τη λειτουργία και τις διαφορές των δύο τρόπων χρονομέτρησης συναρτήσεων.

2.1 Περιγραφή των βασικών πράξεων των συναρτήσεων της Matlab: *lu, qr, svd, det, rank.*

Στο περιβάλλον matlab τα πάντα θεωρούνται πίνακες. Οπότε όλες οι συναρτήσεις ανάγονται σε πολλαπλασιασμούς μητρώων. Ακόμα και οι βαθμωτοί θεωρούνται πίνακες διαστάσεων 1×1. Η συνάρτηση lu εκτελεί την παραγοντοποίηση LU που γνωρίζουμε από τη γραμμική άλγεβρα. Σαν είσοδο μπορεί να πάρει ένα μητρώο A στο οποίο θα εφαρμόσει την παραγοντοποίηση. Για έξοδο επιστρέφει τα μητρώα τα μητρώα $[L, U, P, Q, R]$ τα οποία είναι το L το κάτω τριγωνικό μητρώο, το U είναι το άνω τριγωνικό, το P είναι το αριστερό μητρώο αντιμετάθεσης, το Q είναι το δεξί μητρώο αντιμετάθεσης, και το R το διαγώνιο μητρώο και για τα οποία ισχύει $PR^{-1}AQ = LU$. Οποιοδήποτε όμως από τα μητρώα εξόδου μπορεί να παραληφθεί και να εμφωλευθεί μέσα στα υπόλοιπα. Η συνάρτηση qr εκτελεί την παραγοντοποίηση QR (διαδικασία Gram-Schmidt). Για είσοδο δέχεται ένα μητρώο A, όπου $A \in \mathbb{R}^{m \times n}$ και για έξοδο επιστρέφει τα μητρώα Q και R τα οποία είναι ένα μοναδικό μητρώο $m \times m$ και ένα $m \times n$ άνω τριγωνικό μητρώο αντίστοιχα και για τα οποία ισχύει: $A = QR$. Επίσης μπορεί να επιστρέψει και ένα επιπλέον μητρώο E, αντιμετάθεσης, και για τα οποία θα ισχύει $AE = QR$. Η συνάρτηση svd εφαρμόζει Ανάλυση Ιδιαζουσών Τιμών (Singular Value Decomposition) στο μητρώο A το οποίο παίρνει σαν όρισμα. Για έξοδο επιστρέφει ένα διαγώνιο μητρώο ίδιων διαστάσεων με το το X και τα μητρώα U και V για τα οποία ισχύει: $X = USV^T$. Η συνάρτηση det βρίσκει την ορίζουσα του τετραγωνικού μητρώου X που δέχεται σαν όρισμα. Τέλος η συνάρτηση rank επιστρέφει την τάξη του μητρώου A που δέχεται σαν όρισμα. Επίσης μπορούμε να δώσουμε ένα επιπλέον όρισμα tol και τότε θα επιστρέψει τον αριθμό των ιδιοτιμών του μητρώου A που είναι μεγαλύτερα από την τιμή αυτή.

2.2 Μέτρηση χρόνου εκτέλεσης παραγοντοποίησης LU και πολλαπλασιασμού μητρώου με διάνυσμα

Σε αυτό το μέρος της άσκησης θα εστιάσουμε στον τρόπο με τον οποίο γίνονται οι χρονομετρήσεις για τα προγράμματα μας και τη σημασία του να γίνεται σωστά αυτή η διαδικασία. Τις χρονομετρήσεις θα τις πάρουμε για το χρόνο που χρειάζεται μία παραγοντοποίηση lu ενός μητρώου A και για τον πολλαπλασιασμό του μητρώου με ένα διάνυσμα b, όπου $A \in \mathbb{R}^{n \times n}$ και $b \in \mathbb{R}^{n \times 1}$, οπού $n = 2.\hat{}[7 : 10]$.

2.2.1 Χρονομέτρηση με τη χρήση των συναρτήσεων *tic, toc* και εκτελώντας κάθε πράξη μόνο μία φορά

Στην πρώτη περίπτωση χρονομετράμε μόνο μία φορά τον χρόνο που χρειάζεται κάθε μία πράξη, για τις διάφορες τιμές του n. Στη συνέχεια περιγράφεται ο κώδικας matlab με τον οποίο πραγματοποιήσαμε τις μετρήσεις.

```
function [] = ask2a()
    n = 2.^[7:10];
    tlu = zeros(length(n), 1);
    tmul = zeros(length(n), 1);
    for i=1:length(n)
%       veltistopoiisi gia apofigi
%       kathisterisis logo realocation
%       tou A kai tou b
        A = rand( n(length(n)) );
        b = rand( n(length(n)) ,1);

        tic;
        lu( A(1:n(i),1:n(i)) );
        tlu(i) = toc;

        tic;
        A(1:n(i),1:n(i))*b(1:n(i));
        tmul(i) = toc;
    end

    h=figure
    loglog(n,tlu,'b*-',n,tmul,'r*-');
    xlabel(' log matrix size ');
    ylabel(' log lu time (sec)');
    legend('lu time', 'mul time','Location','NorthWest');
    title('tic toc without loop');
    saveas(h,'ask2a','png')
end
```

Κώδικας Ερωτήματος 2.ii.α

2.2.2 Χρονομέτρηση με την χρήση των συναρτήσεων *tic, toc* με επανάληψη

Ο χρόνος εκτέλεσης ενός προγράμματος σε ένα υπολογιστικό σύστημα εξαρτάται κυρίως κατά μεγάλο βαθμό από το πλήθος των πράξεων που έχει να εκτελέσει αλλά όχι μόνο από αυτό. Για παράδειγμα στο λειτουργικό μας σύστημα μπορεί να εκτελούνται παράλληλα και άλλες διεργασίες, πέραν του προγράμματος που χρονομετράμε. Επίσης θα πρέπει να λάβουμε υπόψιν μας τις μεταφορές που χρειάζονται να γίνουν για να έρθουν τα δεδομένα από την δευτερεύουσα και την κύρια μνήμη, στην μνήμη cache και τους καταχωρητές. Οπότε για να αποφύγουμε αυτό

τον "χρονικό θόρυβο" αυτό το οποίο πρέπει να κάνουμε είναι να εκτελέσουμε το πρόγραμμα αρκετές φορές και να πάρουμε το μέσο όρο. Οπότε εκτελούμε την μέθοδο LU και τον πολλαπλασιασμό 100 φορές και περνούμε τον μέσο όρο των χρόνων.

```
function [] = ask2b()
    n = 2.^[7:10];
    tlu = zeros(length(n), 1);
    tmul = zeros(length(n), 1);

    for i=1:length(n)
%       veltistopoiisi gia apofigi
%       kathisterisis logo realocation
%       tou A kai tou b
        A = rand( n(length(n)) );
        b = rand(n(length(n)),1);

        loops = 100;
        tic;
        for j=1:loops
            lu( A(1:n(i),1:n(i)) );
        end
        tlu(i) = toc/loops

        tic;
        for j=1:loops
            A(1:n(i),1:n(i))*b(1:n(i));
        end
        tmul(i) = toc/loops

    end

    h=figure
    loglog(n,tlu,'bo-',n,tmul,'r*-');
    xlabel(' log matrix size ');
    ylabel(' log lu time (sec)');
    legend('lu time', 'mul time','Location','NorthWest');
    title('tic toc');
    saveas(h,'ask2b','png')
end
```

Κώδικας Ερωτήματος 2.ii.β

2.2.3 Χρονομέτρηση με τη χρήση της συνάρτησης timeit

Ένας άλλος εναλλακτικός, και ποιο σύγχρονος, τρόπος να μετράμε χρόνους συναρτήσεων στη matlab είναι με τη βοήθεια της συνάρτησης timeit, η οποία μετράει με μεγαλύτερη ακρίβεια τον χρόνο εκτέλεσης συναρτήσεων, απαλείφοντας άλλες παραμέτρους.

```
function [] = ask2g()
    n = 2.^[7:10];
    tlu = zeros(length(n), 1);
    tmul = zeros(length(n), 1);
    for i=1:length(n)
%       veltistopoiisi gia apofigi
%       kathisterisis logo realocation
%       tou A kai tou b
        A = rand( n(length(n)) );
        b = rand( n(length(n)) ,1);

        f = @() lu( A(1:n(i),1:n(i)) );
        tlu(i) = timeit(f,2)

        f = @() mtimes(A(1:n(i),1:n(i)),b(1:n(i)));
        tmul(i) = timeit(f,1)
    end

    h=figure
    loglog(n,tlu,'b*-',n,tmul,'r.-');
    xlabel(' log matrix size ');
    ylabel(' log lu time (sec)');
    legend('lu time', 'mul time', 'Location','NorthWest');
    title('timeit');
    saveas(h,'ask2g','png')
end
```

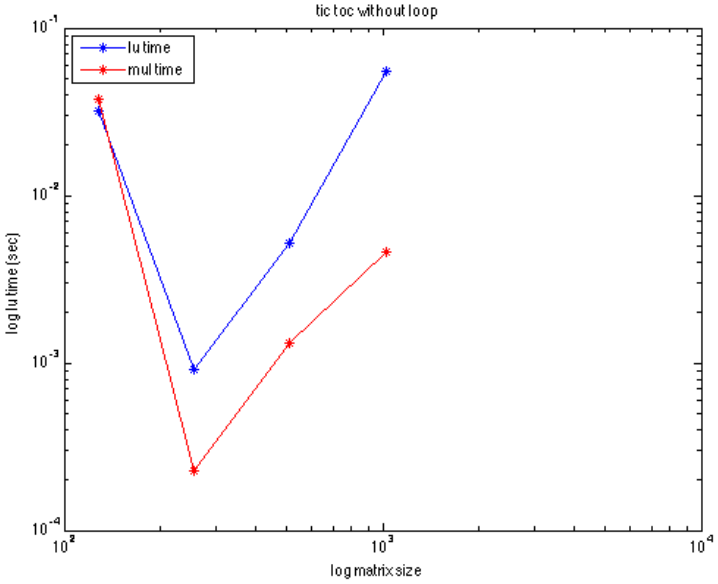
Κώδικας Ερωτήματος 2.ii.γ

2.3 Οπτικοποίηση αποτελεσμάτων του χρόνου με το μέγεθος της εισόδου

Σε αυτό το κεφάλαιο θα δούμε και θα συγκρίνουμε τους χρόνους εκτέλεσης που μετρήσαμε.

2.3.1 Οπτικοποίηση του χρόνου των συναρτήσεων *tic, toc* με εκτέλεση κάθε πράξη μόνο μία φορά

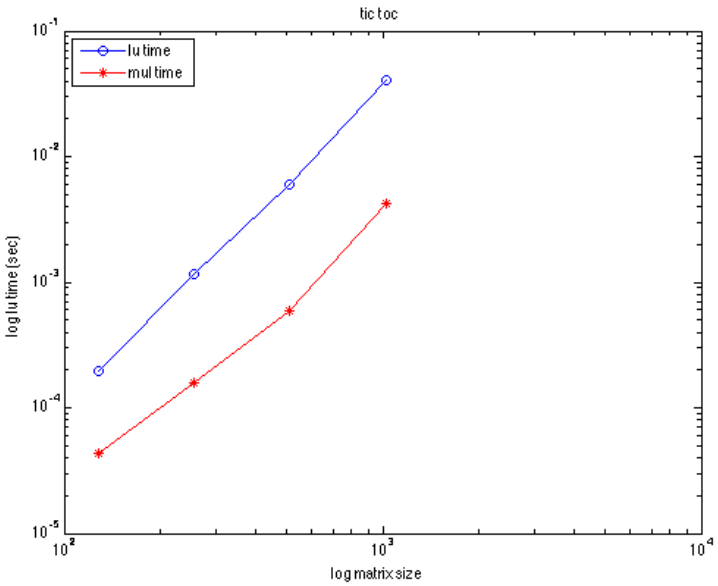
Από τη γραφική παράσταση παρατηρούμε ένα σχετικά περίεργο "φαινόμενο" στις δυο γραφικές παραστάσεις. Η πρώτη μέτρηση και στις δύο περιπτώσεις είναι αρκετά μεγαλύτερες από το αναμενόμενο. Αυτό οφείλεται στο γεγονός ότι το σύστημα χρειάζεται να αρχικοποιηθεί, να κάνει το λεγόμενο warm up, και για αυτό εμφανίζεται αυτή η μεγάλη καθυστέρηση.



Λογαριθμικοί Χρόνοι Εκτέλεσης Ερωτήματος 2.iii.α

2.3.2 Οπτικοποίηση του χρόνου των συναρτήσεων *tic, toc* με επανάληψη

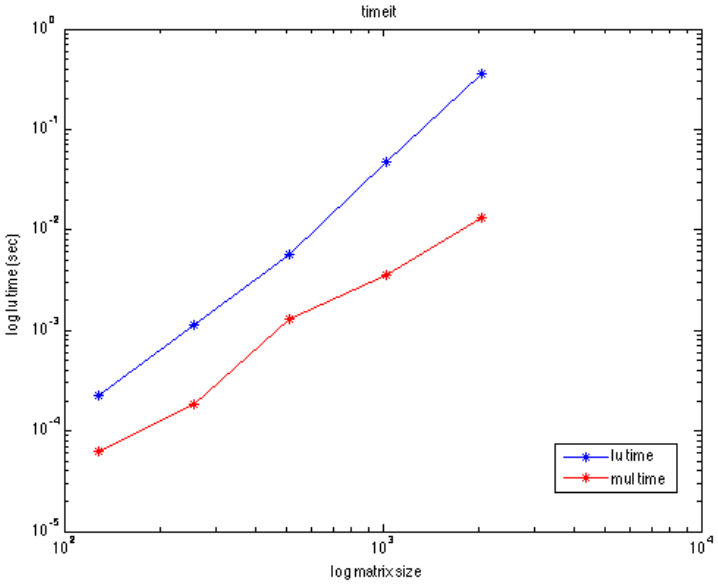
Παρατηρούμε ότι η καμπύλη της LU είναι λίγο μετατοπισμένη προς τα κάτω σε σχέση με τον πολλαπλασιασμό, δηλαδή χρειάστηκε λίγο λιγότερο χρόνο από αυτόν που βρήκαμε στην πρώτη περίπτωση, ενώ ο χρόνος του πολλαπλασιασμού μητρώου με διάνυσμα μειώνεται αρκετά περισσότερο από ότι βρήκαμε με τον προηγούμενο τρόπο.



Λογαριθμικοί Χρόνοι Εκτέλεσης Ερωτήματος 2.iii.β

2.3.3 Οπτικοποίηση του χρόνου της συνάρτησης *timeit*

Παρατηρούμε τώρα με τη χρήση της timeit, ότι ο χρόνος για την LU έχει παραμείνει ίδιος, ενώ για τον πολλαπλασιασμό μητρώου με διάνυσμα έχει μετατοπιστεί προς τα κάτω, δηλαδή ο χρόνος ο οποίος χρειάζεται είναι μικρότερος από ότι είχαμε υπολογίσει και με τους δύο άλλους τρόπους.



Λογαριθμικοί Χρόνοι Εκτέλεσης Ερωτήματος 2.iii.γ

3 Αξιολόγηση Ενδογενών Συναρτήσεων

Σε αυτό το μέρος της εργαστηριακής άσκησης θα αξιολογήσουμε ενδογενείς συναρτήσεις της matlab. Πιο συγκεκριμένα θα χρονομετρήσουμε την πράξη επίλυσης συστήματος εξισώσεων $Ax = b$ όπου $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^{n \times 1}$, $b \in \mathbb{R}^{n \times 1}$

3.1 Χρονομέτρηση της συνάρτησης *mldivide* χρησιμοποιώντας τη συνάρτηση timeit

Θα χρονομετρήσουμε την συνάρτηση *mldivide* για συγκεκριμένους τύπους μητρώων μεγέθους *n* = 2.^[7 : 10].

3.1.1 Τυχαία μητρώα A

Κατασκευάζουμε τα τυχαία μητρώα A με το παρακάτω script και περνούμε μετρήσεις.

```
n = 2.^[7:10];
tmldividea = zeros(length(n),1);

for i=1:length(n)
    A = rand( n(length(n)) );
    b = rand( n(length(n)) ,1);

    f = @() mldivide( A(1:n(i),1:n(i)), b(1:n(i)) );
    tmldividea(i) = timeit(f,1)
end
```

Κώδικας Ερωτήματος 3.α.i

3.1.2 Τυχαία τριγωνικά μητρώα L

Κατασκευάζουμε τα τυχαία μητρώα L με το παρακάτω script και περνούμε μετρήσεις για το χρόνο εκτέλεσης.

```
n = 2.^[7:10];
tmldivideb = zeros(length(n),1);

for i=1:length(n)
    L = tril( rand( n(i) ) );
    b = rand( n(i), 1 );

    f = @() mldivide( L, b );
    tmldivideb(i) = timeit(f,1)
end
```

Κώδικας Ερωτήματος 3.α.ii

3.1.3 Τυχαία ψυχολογικά κάτω τριγωνικά μητρώα L

Κατασκευάζουμε τα τυχαία μητρώα L με το παρακάτω script και περνούμε μετρήσεις για το χρόνο εκτέλεσης.

```
n = 2.^[7:10];
tmldivideg = zeros(length(n),1);

for i=1:length(n)
    p = randperm( n(i) );
    L = tril( rand( n(i) ));
    L = L(:,p);
    b = rand( n(i), 1 );

    f = @() mldivide( L, b );
    tmldivideg(i) = timeit(f,1)
end
```

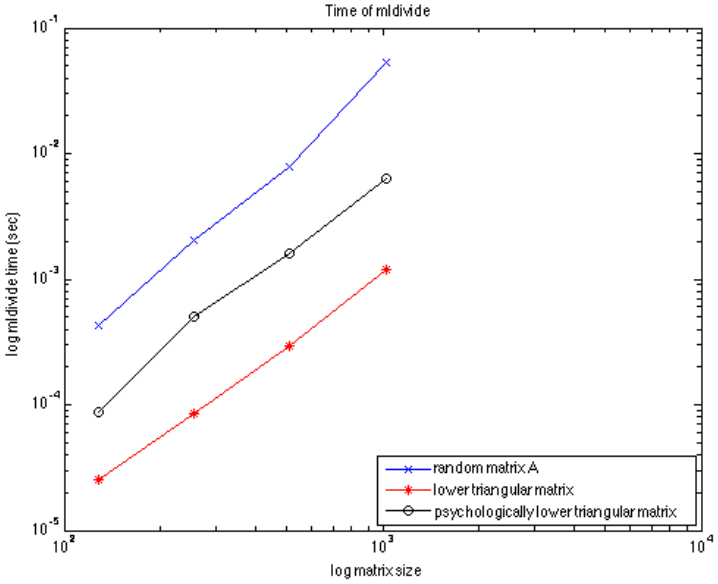
Κώδικας Ερωτήματος 3.α.iii

3.2 Οπτικοποίηση επίδοσης της συνάρτησης *mldivide* και σχολιασμός επιδόσεων

Παρουσιάζονται σε ένα διάγραμμα οι γραφικές παραστάσεις του χρόνου εκτέλεσης της συνάρτησης mldivide για τους προηγούμενους τύπους μητρώων. Οι γραφική παράσταση δημιουργήθηκε εκτελώντας το παρακάτω script.

```
ask3a
ask3b
ask3g

h=figure
loglog(n,tmldividea,'bx-', n,tmldivideb,'r*-', ...
    n,tmldivideg,'ko-')
xlabel(' log matrix size ');
ylabel(' log mldivide time (sec)');
legend('random matrix A', 'lower triangular matrix', ...
    'psychologically lower triangular matrix', ...
    'Location','SouthEast');
title('Time of mldivide');
saveas(h,'askplot3','png')
```

Κώδικας Ερωτήματος 3.β



Χρόνοι Εκτέλεσης Ερωτήματος 3.β

Από τη γραφική παράσταση βλέπουμε πως ο χρόνος ο οποίος χρειάζεται για την επίλυση του συστήματος είναι μικρότερος στην περίπτωση που το μητρώο μας έχει ειδική μορφή. Συγκεκριμένα βλέπουμε πως η matlab καταλαβαίνει και την τριγωνική μορφή του μητρώου, καθώς χρειάζεται τον λιγότερο χρόνο από όλα τα μητρώα, καθώς και την μορφή των ψυχολογικά κάτω τριγωνικών, καθώς ο χρόνος τους είναι μικρότερος από ένα τυχαίο μητρώο ίδιων διαστάσεων.

3.3 Χρονομέτρηση βασικών αλγεβρικών πράξεων για μητρώα ειδικής μορφής

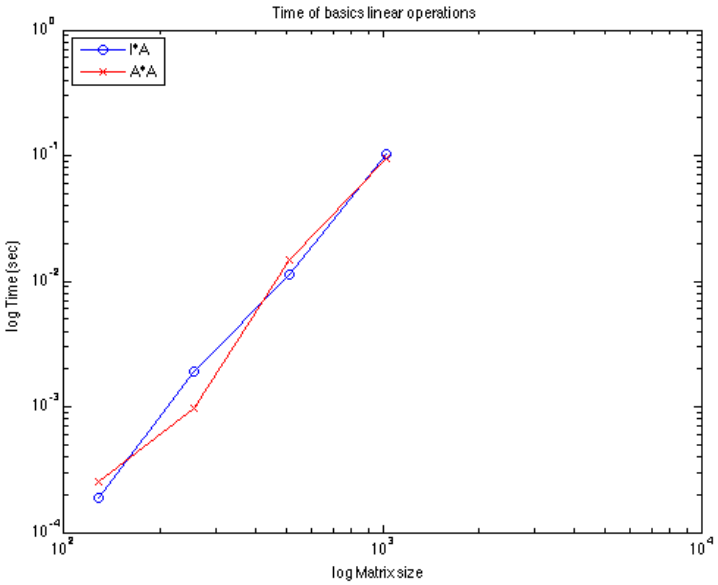
Σε αυτό το σημείο θα εμβαθύνουμε στο τι συμβαίνει όταν έχουμε μητρώα ειδικής μορφής και κατά πόσο αυτό μπορεί να επηρεάσει τους χρόνους εκτέλεσης των βασικών των βασικών αλγεβρικών πράξεων. Αναλυτικότερα συγκρίνουμε τους χρόνους εκτέλεσης για τις πράξεις $I \cdot A, A \cdot A$ όπου $I \in \mathbb{R}^{n \times n}$ είναι το ταυτοτικό μητρώο και $A \in \mathbb{R}^{n \times n}$ είναι ένα τυχαίο μητρώο. Εκτελώντας το παρακάτω script στη Matlab χρονομετράμε τις επιδώσεις των δυο εσωτερικών γινόμενων.

```
n = 2.^[7:10];
teye = zeros( length(n) ,1);
tsq = zeros( length(n) ,1);
for i=1:length(n)
    I = eye(n(i),n(i));
    A = rand(n(i),n(i));

    f = @() mtimes(A, A);
    tsq(i) = timeit(f,1);
```

```
10
11      f = @() mtimes(I,A);
12      teye(i) = timeit(f,1);
13
14      fprintf('%d/%d \n', i, length(n));
15  end
16
17  h=figure;
18  loglog(n,teye,'bo-', n,tsq, 'rx-');
19  xlabel('log Matrix size');
20  ylabel('log Time (sec)');
21  l=legend('I*A', 'A*A','Location','NorthWest');
22  title('Time of basics linear operations');
23  saveas(h,'ask3d','png') %save figure
```

Κώδικας Ερωτήματος 3.δ

Από το το script παίρνουμε σαν έξοδο την παρακάτω γραφική παράσταση.



Χρόνοι Εκτέλεσης Ερωτήματος 3.δ

Παρατηρούμε ότι η matlab δεν μπορεί να καταλάβει τον πολλαπλασιασμό με το ταυτοτικό μητρώο και εκτελεί κανονικά τις πράξεις όπως θα έκανε και με ένα άλλο οποιοδήποτε μητρώο. Δηλαδή η matlab παρ'όλο που μπορεί να καταλάβει την τριγωνική και την ψυχολογική τριγωνική μορφή δεν μπορεί να καταλάβει το ταυτοτικό μητρώο.

4 Σύκριση Υλοποιήσεων

Σε αυτό το σημείο, και σύμφωνα και με τα προηγούμενα συμπεράσματα, είμαστε πλέον σε θέση να σχεδιάσουμε και να αξιολογήσουμε τόσο από πλευρά χρόνου όσο και από πλευρά πράξεων διαφορετικούς τρόπους υλοποίησης του ιδού υπολογιστικού προβλήματος. Συγκεκριμένα θα λύσουμε ένα απλό πρόβλημα της γραμμικής άλγεβρας και θα αξιολογήσουμε την απόδοσή τους με κριτήριο τον χρόνο εκτέλεσης αυτών.

4.1 Θεωρητικός υπολογισμός πράξεων κινητής υποδιαστολής του γινομένου $\prod_{k=1}^{p}(I - uv^T)$

Υπολογίζουμε θεωρητικά το απαιτούμενο πλήθος πράξεων κινητής υποδιαστολής συνάρτηση των n και p για τον υπολογισμό του γινομένου $\prod_{k=1}^{p}(I - uv^T)$ όπου $I \in \mathbb{R}^{n\times n}$ είναι το ταυτοτικό μητρώο, $u \in \mathbb{R}^{n\times 1}$ και $v \in \mathbb{R}^{n\times 1}$ τυχαία διανύσματα. Θεωρούμε ότι οι υπολογισμοί γίνονται από αριστερά προς τα δεξιά και τηρείται η προτεραιότητα των πράξεων.

Για το γινόμενο uv^T έχουμε: $\Omega_{uv^T} = n^2$. Οπότε για το $A = I - uv^T$ θα χρειαστούμε $\Omega_A = \Omega_{uv^T} + n^2 = 2n^2$ και αφού η πράξη αυτή θα εκτελεστεί p φορές έχουμε ότι $\Omega_{A,p} = 2pn^2$. Οπότε τώρα έχουμε υπολογίσει το μητρώο Α p φορές μένει να πολλαπλασιάσουμε $p-1$ φορές με τον αυτό του. Οπότε το συνολικό κόστος είναι:

$$\Omega_1 = 2pn^2 + (p-1)(2n-1)n^2 = 2(p-1)n^3 + (p+1)n^2 \tag{4.1.1}$$

4.2 Θεωρητικός υπολογισμός πράξεων κινητής υποδιαστολής μόνο μίας στήλης του γινομένου $(\prod_{k=1}^{p}(I - uv^T))e_k$

Τώρα θέλουμε να υπολογίσουμε μόνο μία στήλη από το προηγούμενο γινόμενο. Για να το κάνουμε αυτό αρκεί να πολλαπλασιάσουμε το γινόμενο από δεξιά με ένα διάνυσμα e_k, όπου είναι η k-οστή στήλη του ταυτοτικού μητρώου, $(\prod_{k=1}^{p}(I - uv^T))e_k$. Αυτή την φορά όμως κάνουμε την υπόθεση ότι οι πράξεις εκτελούνται από δεξιά προς τα αριστερά. Οπότε τώρα μπορούμε να υπολογίσουμε το μητρώο.

Η θεώρηση ότι οι πράξεις εκτελούνται από δεξιά προς τα αριστερά μας δίνει το πλεονέκτημα ότι θα έχουμε να εκτελέσουμε πολλαπλασιασμό μητρώου με διάνυσμα και όχι μητρώο με μητρώο. Όπως είδαμε και από το προηγούμενο ερώτημα το κόστος για τον υπολογισμό του γινομένου $I - uv^T$, p φορές είναι $\Omega_{A,p} = 2pn^2$. Στη συνέχεια θα πολλαπλασιάσει την δεξιότερη παρένθεση με το διάνυσμα e_k το

οποίο χρειάζεται $\Omega_{Ae_k} = n(2n-1) = 2n^2 - n$ και μας δίνει σαν αποτέλεσμα ένα διάνυσμα διαστάσεων $n \times 1$. Οπότε θα έχουμε συνολικά να εκτελέσουμε p πολλαπλασιασμούς μητρώου με διάνυσμα και το συνολικό κόστος βγαίνει:

$$\Omega_2 = \Omega_{A,p} + p\Omega_{Ae_k} = 2pn^2 + p(2n^2 - n) = 4pn^2 - pn \tag{4.2.2}$$

Παρατηρούμε ότι το Ω_2 είναι μίας τάξης μικρότερο από το Ω_1, δηλαδή θα έχουμε αρκετά λιγότερες πράξεις.

4.3 Υλοποίηση συνάρτηση: $(\prod_{k=1}^{p}(I - uv^T))e_k$

Υλοποιούμε την συνάρτηση my_func η οποία υλοποιεί τον παραπάνω υπολογισμό. Η συνάρτηση αυτή δέχεται σαν όρισμα με την σειρά τις παραμέτρους p, τα διανύσματα εισόδου u, v και την τιμή col που υποδηλώνει τον αριθμό της στήλης που θέλουμε να υπολογίσουμε. Αν παραλείψουμε την παράμετρο col τότε η συνάρτηση θα πρέπει να επιστρέφει το πλήρες αποτέλεσμα και όχι μόνο συγκεκριμένη στήλη. Παρακάτω παρουσιάζεται η συνάρτηση σε κώδικα matlab.

```
function [A] = my_func(p, u, v, col)
    if nargin < 3 || nargin > 4
        fprintf('Error! \n function [A] = my_func(p, u, v, ...
            col).\n You have two give at least three ...
            arguments \n');
        return;
    end
    n = length(u);
    I = eye(n);

    if nargin == 3
        A = eye( n );
        for i=1:p
            A = A*(I - u*v');
        end
    else %nargin == 4
        A = zeros( n,1 );
        A(col) = 1;
        for i=1:p
            A = (I - u*v')*A;
        end
    end
end
```

Κώδικας Ερωτήματος 4.3

4.4 Χρονομέτρηση της συνάρτησης my_func και αξιολόγηση

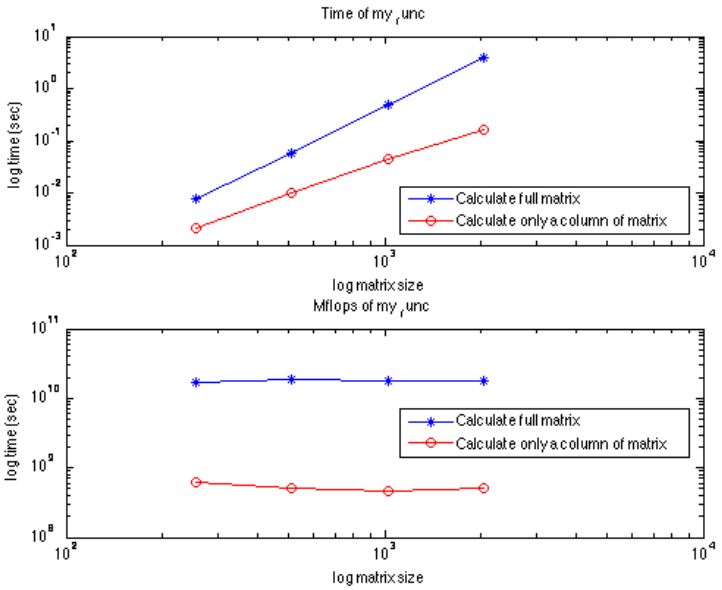
Για *n* = 2.^[7 : 10] και τυχαία διανύσματα u και v χρονομετράμε δύο περίπτωσης εκτέλεσης της συνάρτησης my_func. Μία με το όρισμα col και μία χωρίς αυτό.

Εκτελώντας το παρακάτω script χρονομετρήσαμε την συνάρτηση και μετρήσαμε και τα Mflop/s.

```
p = 5;
n=2.^[8:11]';
t = zeros(length(n),1);
t_col = zeros(length(n),1);
for i=1:length(n)

    u = rand(n(i),1);
    v = rand(n(i),1);
    col = randi(n(i));

    f = @() my_func(p,u,v);
    t(i) = timeit(f,1)

    f = @() my_func(p,u,v,col);
    t_col(i) = timeit(f,1)
end

h=figure
subplot(2,1,1);
loglog(n,t,'b*-',n,t_col,'ro-');
xlabel(' log matrix size ');
ylabel(' log time (sec)');
legend('Calculate full matrix', 'Calculate only a column of ...
    matrix', 'Location','best');
title('Time of my_func');

mips = ( 2*(p-1).*n.^3+(p+1).*n.^2 ) ./ t;
mips_col = ( 4*p.*n.^2 - p.*n ) ./ t_col;
subplot(2,1,2);
loglog(n,mips,'b*-',n,mips_col,'ro-');
xlabel(' log matrix size ');
ylabel(' log time (sec)');
legend('Calculate full matrix', 'Calculate only a column of ...
    matrix', 'Location','best');
title('Mflops of my_func');

saveas(h,'ask4_4_a','png')
```

Κώδικας Ερωτήματος 4.4

Εκτελώντας το παραπάνω script παίρνουμε τους παρακάτω χρόνους.



Χρόνοι Εκτέλεσης Ερωτήματος 4.4

Με πρώτη ματιά βλέπουμε πως οι χρόνοι οι οποίοι έχουμε όταν βρίσκουμε ολόκληρο το μητρώο είναι είναι αρκετά μεγαλύτερος από όταν όταν βρίσκουμε μόνο μία στήλη. Όμως αν κοιτάξουμε και τη γραφική παράσταση με τα Gflops θα δούμε ότι οι στην περίπτωση που υπολογίζουμε ολόκληρο το μητρώο τα Gflops είναι πολύ περισσότερα, δηλαδή εκτελούνται πολύ περισσότερες πράξεις το δευτερόλεπτο και άρα κάνουμε καλύτερη αξιοποίηση των πόρων του υπολογιστικού μας συστήματος.

Αν θα θέλαμε να βελτιώσουμε τους χρόνους εκτέλεσης των προγραμμάτων θα μπορούσαμε να υπολογίζαμε το μητρώο $A = I - uv^T$ μόνο μία φορά και να το αποθηκεύαμε σε μία μεταβλητή αντί να το υπολογίζουμε p φορές. Επίσης ανάλογα την μνήμη cashe που έχει ο υπολογιστής που το τρέχουμε θα μπορούσαμε να κρατάμε και σε μεταβλητές τις τιμές A^2, A^4, A^8, A^{16}... ανάλογα του p που θα είχαμε. Αυτό θα μείωνε κατά πολύ τις πράξεις αλλά θα μας αύξανε το κόστος της μνήμης που θα θέλαμε ($n \times n$ για κάθε ένα μητρώο). Επίσης για τον υπολογισμό του μόνο της μίας στήλης του γινομένου θα μπορούσαμε να εφαρμόσουμε την επιμεριστική ιδιότητα και να μετατρέψουμε την αφαίρεση μητρώου με μητρώου σε αφαίρεση διανυσμάτων. Γενικά αν έχουμε να υπολογίσουμε μία αριθμητική παράσταση με μητρώα και διανύσματα είναι καλύτερο να κάνουμε πρώτα τις πράξεις που θα μας μικρύνουν την τάξη του προβλήματος.

Βιβλιογραφία

[1] . Gilbert Strang. Εισαγωγή στη γραμμική άλγεβρα. Εκδόσεις Πανεπιστήμιο Πατρών 2012

[2] . Επιστημονικός υπολογισμός Ι. Ευστράτιος Γαλλόπουλος. 2013

[3] . *http://www.mathworks.com/help/matlab/*