Επιστημονικός Υπολογισμός Ι
1η Εργαστηριακή Άσκηση

Παπαρροδοπούλου Αναστασία ΑΜ 3873
12/11/13

Ερώτημα 1 - Εισαγωγικά:

(i) Για τον υπολογισμό των παρακάτω χαρακτηριστικών του υπολογιστικού συστήματος στο οποίο υλοποιήθηκε η εργαστηριακή άσκηση, χρησιμοποιήθηκαν δύο ειδικά προγράμματα(τα οποία κατεβάστηκαν από τη διεύθυνση http://www.cpuid.com/), το CPU-Z και το PC Wizard.

Τύπος και συχνότητα λειτουργίας επεξεργαστή :

Intel Mobile Core 2Duo P8600 @ 2.40GHz

Μέγεθος και αριθμός πιπέδων κρυφής μνήμης :

L1 D-Cache : Size 32 Kbytes x2, Descriptor 8-way set associative, 64-byte line size
L1 I-Cache: Size 32 Kbytes x2, Descriptor 8-way set associative, 64-byte line size
L2 Cache : Size 3072 Kbytes, Descriptor 12-way set associative, 64-byte line size

Το **είδος της πολιτικής εγγραφής** στην κρυφή μνήμη είναι Write-Back.

Το λειτουργικό σύστημα του μηχανήματος είναι Windows 7 Professional Service Pack 1 (64-bit).

(ii) Η **έκδοση Matlab** που χρησιμοποιήσα είναι η R2012b(64-bit).

(iii) Για μια εκτίμηση της διακριτότητας του χρονομετρητή, μέσω των εντολών tic toc και όπως αυτό υποδείχτηκε στις διαλέξεις του μαθήματος, συνένταξα τον ακόλουθο κώδικα ο οποίος και εκτελεί τις εντολές 20 φορές.

```
tic
toc;
t=zeros(20,1);
for i=1:20
    tic
    t(i)=toc;
end
time=sum(t)/20;
```

Η διακριτότητα λοιπόν, εκτιμάται στα 1.8397×10^{-7} sec.

Σε αυτό το σημείο θα μπορούσα να προσθέσω και έναν επιπλέον κώδικα, ο οποίος δημιουργήθηκε σύμφωνα με τον εσωτερικό κώδικα της timeit, και ο οποιός εκτελεί τις tic toc 100 φορές, και είναι ο ακόλουθος:

```
function [t] = my_tic_toc
elapsed = 0;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
.
.
.
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
t = elapsed / 100;
```

Ενδεικτικά λοιπόν, αυτός ο κώδικας φαίνεται να δίνει μια ακρίβεια των 1.9509×10^{-7} sec.

(iv) Το αποτέλεσμα της εντολής bench για τη διάσπαση μητρώων LU είναι 0.3050 sec.

Ερώτημα 2 – Αξιολόγηση Ενδογενών Συναρτήσεων:

(i) Ακολουθεί η ανάλυση των βασικών πράξεων γραμμικής άλγεβρας που υλοποιούν οι συναρτήσεις του Matlab lu, qr, svd και eig.

lu : Στη γραμμική άλγεβρα η διάσπαση LU(ή παραγοντοποίηση LU), όπου το LU αντικαθιστά το 'Lower Upper', παράγει ένα μητρώο σα γινόμενο ενός κάτω τριγωνικού μητρώου L και ενός άνω τριγωνικού μητρώου U. Η διάσπαση LU μπορεί επίσης να θεωρηθεί μία απαλοιφή Gauss σε μορφή μητρώου. Οι υπολογιστές λύνουν συνήθως τετραγωνικά συστήματα γραμμικών εξισώσεων χρησιμοποιώντας στη διάσπαση αυτή, ενώ αποτελεί επίσης ένα βήμα της αναστροφής ενός μητρώου και του υπολογισμού της ορίζουσάς του.

qr : Η διάσπαση QR ενός μητρώου τώρα, είναι η διάσπαση ενός μητρώου A σε ένα γινόμενο A=QR, ενός ορθογωνίου μητρώου Q και ενός άνω τριγωνικού μητρώου R. Πιο συγκεκριμένα το A είναι ένα mxn μητρώο, που παράγει ένα mxn άνω τριγωνικό μητρώο R και ένα mxm μοναδιαίο μητρώο Q. Η διάσπαση QR χρησιμοποιείται συνήθως για να λύσει γραμμικά προβήματα ελαχίστων τετραγώνων, και αποτελεί τη βάση του QR αλγορίθμου.

svd : Η ανάλυση ιδιαζουσών τιμών [U,S,V] = svd(X) είναι μια παραγοντοποιήση ενός mxn μητρώου X, τετραγωνικού ή ορθογώνιου σχήματος, σε ένα mxm μοναδιαίο μητρώο U, ένα διαγώνιο mxn μητρώο S με όχι αρνητικά στοιχεία στη διαγώνιό του, τα οποία είναι σε φθήνουσα σειρά, και ένα μοναδιαίο μητρώο V ώστε X = U*S*V'(όπου V', το V transpose του V).

eig : Η εύρεση ιδιοτιμών είναι η παραγοντοποίηση ενός μητρώου σε μία κανονικοποιημένη μορφή, όπου το μητρώο παρουσιάζεται ως προς τις ιδιοτιμές του και τα ιδιοδιανύσματά του. Μόνο τα διαγωνιοποιήσιμα μητρώα μπορούν να διασπαστούν έτσι. Έτσι το E = eig(X) είναι ένα διάνυσμα που περιέχει τις ιδιοτιμές ενός τετραγωνικού μητρώου X.

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

(ii) (α) Ο κώδικας που υλοποιήθηκε με τη χρήση των συναρτήσεων tic, toc και εκτελώντας κάθε πράξη μόνο μία φορά, είναι ο εξής:

```
n = [200 400 600 800 1000 1200];
t1 = zeros(6,1);
t2 = zeros(6,1);
t3 = zeros(6,1);
t4 = zeros(6,1);
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [L,U] = lu(R);
    tic;
    [L,U] = lu(R);
    t1(i) = toc;
end
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [Q,R] = qr(R);
    tic;
    [Q,R] = qr(R);
    t2(i) = toc;
end
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [U,S,V] = svd(R);
    tic;
    [U,S,V] = svd(R);
    t3(i) = toc;
end
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [V,D] = eig(R);
    tic;
    [V,D] = eig(R);
    t4(i) = toc;
end
figure
plot(n,t1,'*-',n,t2,'^-',n,t3,'+-',n,t4,'x-','LineWidth',2);
legend('lu','qr','svd','eig','Location','Best');
xlabel('n');
ylabel('time(sec)');
title('2.ii.a');
grid on;
```

Οι χρόνοι που υπολογίστηκαν αναλυτικά λοιπόν είναι οι ακόλουθοι:

sec/n	200	400	600	800	1000	1200
t1	0.0025	0.0168	0.0504	0.0512	0.0902	0.1552
t2	0.0009	0.0052	0.0131	0.0272	0.0404	0.0584
t3	0.0206	0.0984	0.2760	0.7230	1.5179	2.7543
t4	0.0667	0.2879	0.9190	2.0331	3.7738	6.6047

(ii) (β) Ο κώδικας που υλοποιήθηκε με τη χρήση των συναρτήσεων tic, toc όπως αυτός υποδείχτηκε στις διαλέξεις, είναι ο εξής:

```
n = [200 400 600 800 1000 1200];
t1 = zeros(6,1);
t2 = zeros(6,1);
t3 = zeros(6,1);
t4 = zeros(6,1);
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [L,U] = lu(R);
    elapsed = 0;
    for k=1:1:30
        tic;
        [L,U] = lu(R);
        elapsed = elapsed + toc;
    end
    t1(i) = elapsed/30;
end
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [Q,R] = qr(R);
    elapsed = 0;
    for k=1:1:25
        tic;
        [Q,R] = qr(R);
        elapsed = elapsed + toc;
    end
    t2(i) = elapsed/25;
end
```

```
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [U,S,V] = svd(R);
    elapsed = 0;
    for k=1:1:20
        tic;
        [U,S,V] = svd(R);
        elapsed = elapsed + toc;
    end
    t3(i) = elapsed/20;
end
for i=1:1:6
    R = rand(n(i));
    tic;
    toc;
    [V,D] = eig(R);
    elapsed = 0;
    for k=1:1:15
        tic;
        [V,D] = eig(R);
        elapsed = elapsed + toc;
    end
    t4(i) = elapsed/15;
end
figure
plot(n,t1,'*-',n,t2,'^-',n,t3,'+-',n,t4,'x-','LineWidth',2);
legend('lu','qr','svd','eig','Location','Best');
xlabel('n');
ylabel('time(sec)');
title('2.ii.b');
grid on;
```

Οι χρόνοι που υπολογίστηκαν αναλυτικά είναι οι ακόλουθοι:

sec/n	200	400	600	800	1000	1200
t1	0.0025	0.0087	0.0198	0.0377	0.0658	0.1084
t2	0.0009	0.0050	0.0131	0.0244	0.0400	0.0585
t3	0.0201	0.1015	0.2799	0.7307	1.5316	2.7706
t4	0.0685	0.2827	0.9179	2.1064	3.9657	6.4397

(ii) (γ) Ο κώδικας που υλοποιήθηκε με τη χρήση της συνάρτησης timeit, είναι ο εξής:

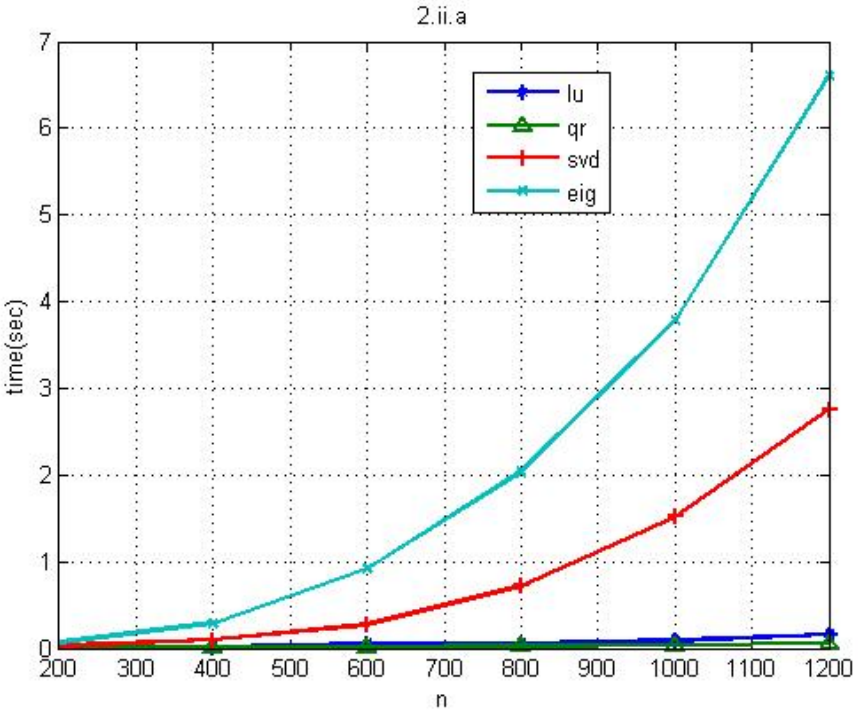
```
n = [200 400 600 800 1000 1200];
t1 = zeros(6,1);
t2 = zeros(6,1);
t3 = zeros(6,1);
t4 = zeros(6,1);
for i=1:1:6
    R = rand(n(i));
    flu = @() lu(R);
    t1(i) = timeit(flu,2);
end
for i=1:1:6
    R = rand(n(i));
    fqr = @() qr(R);
    t2(i) = timeit(fqr,2);
end
for i=1:1:6
    R = rand(n(i));
    fsvd = @() svd(R);
    t3(i) = timeit(fsvd,3);
end
for i=1:1:6
    R = rand(n(i));
    feig = @() eig(R);
    t4(i) = timeit(feig,2);
end
figure
plot(n,t1,'*-',n,t2,'^-',n,t3,'+-',n,t4,'x-','LineWidth',2);
legend('lu','qr','svd','eig','Location','Best');
xlabel('n');
ylabel('time(sec)');
title('2.ii.c');
grid on;
```

Οι χρόνοι που υπολογίστηκαν αναλυτικά είναι οι ακόλουθοι:

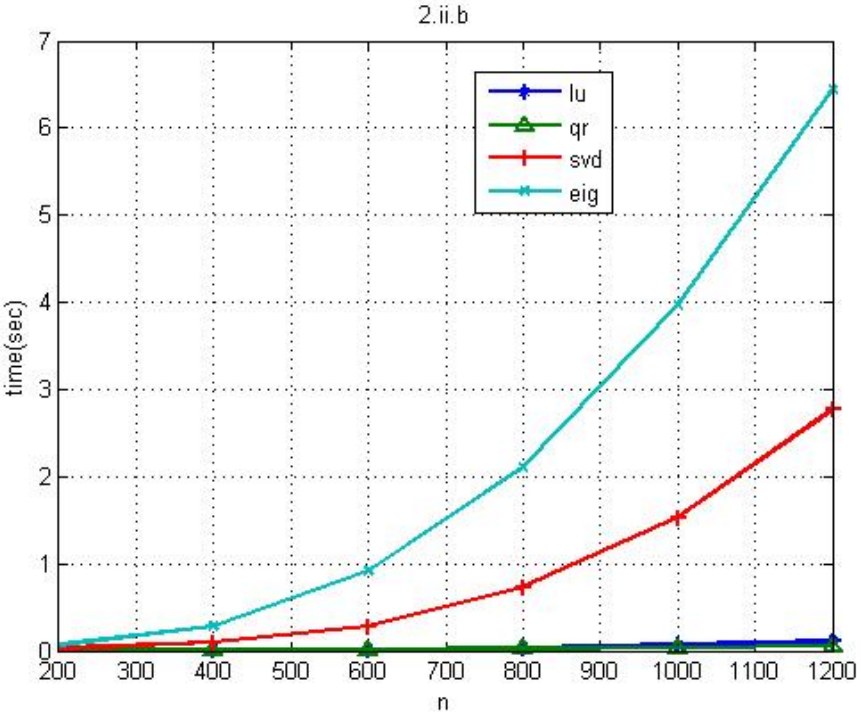
sec/n	200	400	600	800	1000	1200
t1	0.0027	0.0087	0.0173	0.0397	0.0648	0.1118
t2	0.0034	0.0201	0.0630	0.1369	0.2514	0.4168
t3	0.0199	0.1018	0.2842	0.7418	1.5168	2.7587
t4	0.0665	0.2800	0.9196	2.0580	3.7536	6.4827

(iii) Για την οπτικοποίηση των αποτελεσμάτων για τις διάφορες πράξεις και για καθένα από τους τρεις παραπάνω τρόπους, παρήχθησαν τα παρακάτω τρία figures:

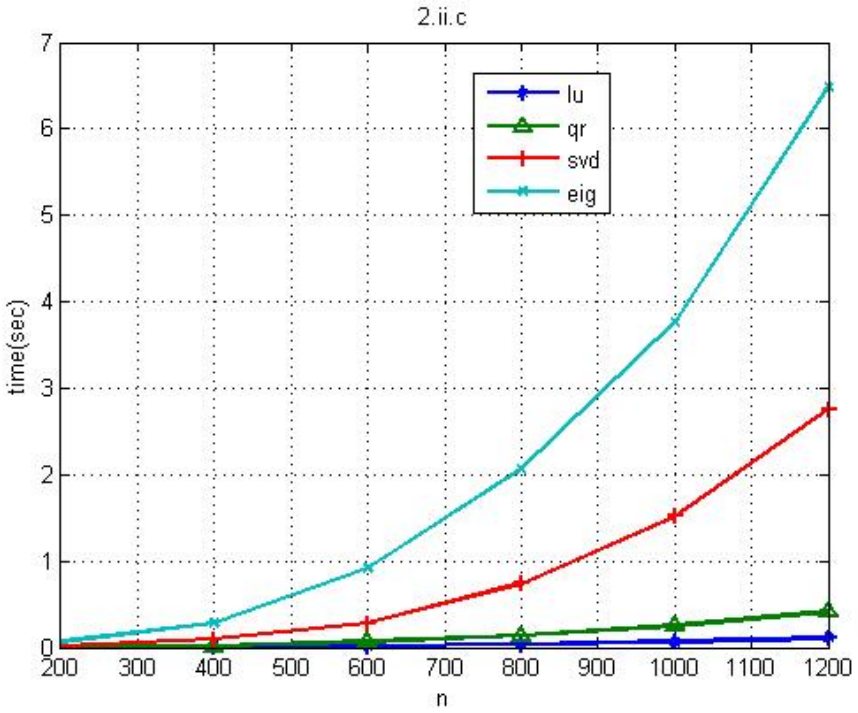
Για το (α) ερώτημα:



Για το (β) ερώτημα:



Για το (γ) ερώτημα:



Παρατηρώ γενικότερα ότι οι συναρτήσεις lu και qr χρειάζονται το λιγότερο χρόνο σε κάθε μέτρηση που κάνουμε, ενώ οι svd και eig σημαντικά μεγαλύτερο. Αυτό οφείλεται στο ότι οι συναρτήσεις αυτές, με τη σειρά που μας δίνονται και που τις αναπαριστούμε κι εμείς αντίστοιχα, αυξάνονται σε πολυπλοκότητα. Σε αυτό το σημείο μπορούμε να κάνουμε και την παρατήρηση, ότι ενώ ναι μεν υπάρχει αύξηση του χρόνου σε κάθε συνάρτηση καθώς αυξάνεται το μέγεθος του n, ο ρυθμός αύξησης του χρόνου στις lu και qr είναι πολύ μικρότερος σε σχέση με αυτό των svd και eig. Στις δύο τελευταίες βλέπουμε μεγάλη αύξηση από πολύ νωρίς, η οποία γίνεται πιο έντονη στο διάστημα που το n είναι 800 και γίνεται 1000, ενώ η μεγαλύτερη αύξηση παρατηρείται στο διάστημα 1000-1200.

Κάτι άλλο επίσης που θα ήταν άξιο σχολιασμού, είναι τα αποτελέσματα των χρόνων που παίρνουμε και φαίνονται και στα figures, για τις lu και qr. Οι χρόνοι στον $3^ο$ τρόπο μετρήσεων(ερώτημα γ), είναι όπως θα τους αναμέναμε και θεωρητικά. Στον $1^ο$ και το $2^ο$ όμως(ερώτημα α και β αντίστοιχα), η lu φαίνεται να παίρνει παραπάνω χρόνο από την qr, αν και με ελάχιστη διαφορά. Αυτό μπορούμε εύκολα να συμπεράνουμε πως οφείλεται στον τρόπο που έχουν παρθεί οι μετρήσεις και στην αξιοπιστία τους. 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 στις δύο πρώτες συναρτήσεις. Αυτό με το $2^ο$ τρόπο μετρήσεων(ερώτημα β), φαίνεται πως μετριάζεται και τα αποτελέσματα είναι καλύτερα από τις ακραίες τιμές. Το πρόβλημα αυτό λοιπόν όπως αναφέρθηκε πάλι, παύει να υπάρχει στην υλοποίηση με την timeit όπου έχουμε μεγαλύτερη ακρίβεια στους χρόνους της κάθε συνάρτησης.

**Οι κώδικες για την παραγωγή των figures έχουν συμπεριληφθεί στους κώδικες των παραπάνω υποερωτημάτων.*

Ερώτημα 3 – Σύγκριση Υλοποιήσεων:

(i) Κατασκεύασα δύο συναρτήσεις, μία για πολλαπλασιασμό μητρώου με διάνυσμα κατά γραμμές, και μία κατά στήλες.

Ο κώδικας για τη συνάρτηση mv_ij(πολλαπλασιασμός κατά γραμμές) είναι ο εξής:

```
function [y] = mv_ij(A,x);
[m,n]= size(A);
for i=1:m
    y(i) = 0;
end;
for i = 1:m
    for j = 1:n
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

Ο κώδικας για τη συνάρτηση mv_ji(πολλαπλασιασμός κατά στήλες) είναι ο εξής:

```
function [y] = mv_ji(A,x);
[m,n]= size(A);
for i=1:m
    y(i) = 0;
end;
for j = 1:n
    for i = 1:m
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

(ii) Ακολουθεί ο κώδικας για το script test_mv με τις λειτουργίες που έχουν ζητηθεί από την εκφώνηση. Να σημειωθεί πως εχουμε θεωρήσει ότι ο αριθμός των πράξεων είναι ίδιος για όλες τις υλοποιήσεις, όπως μας προτάθηκε και στο εργαστήριο.

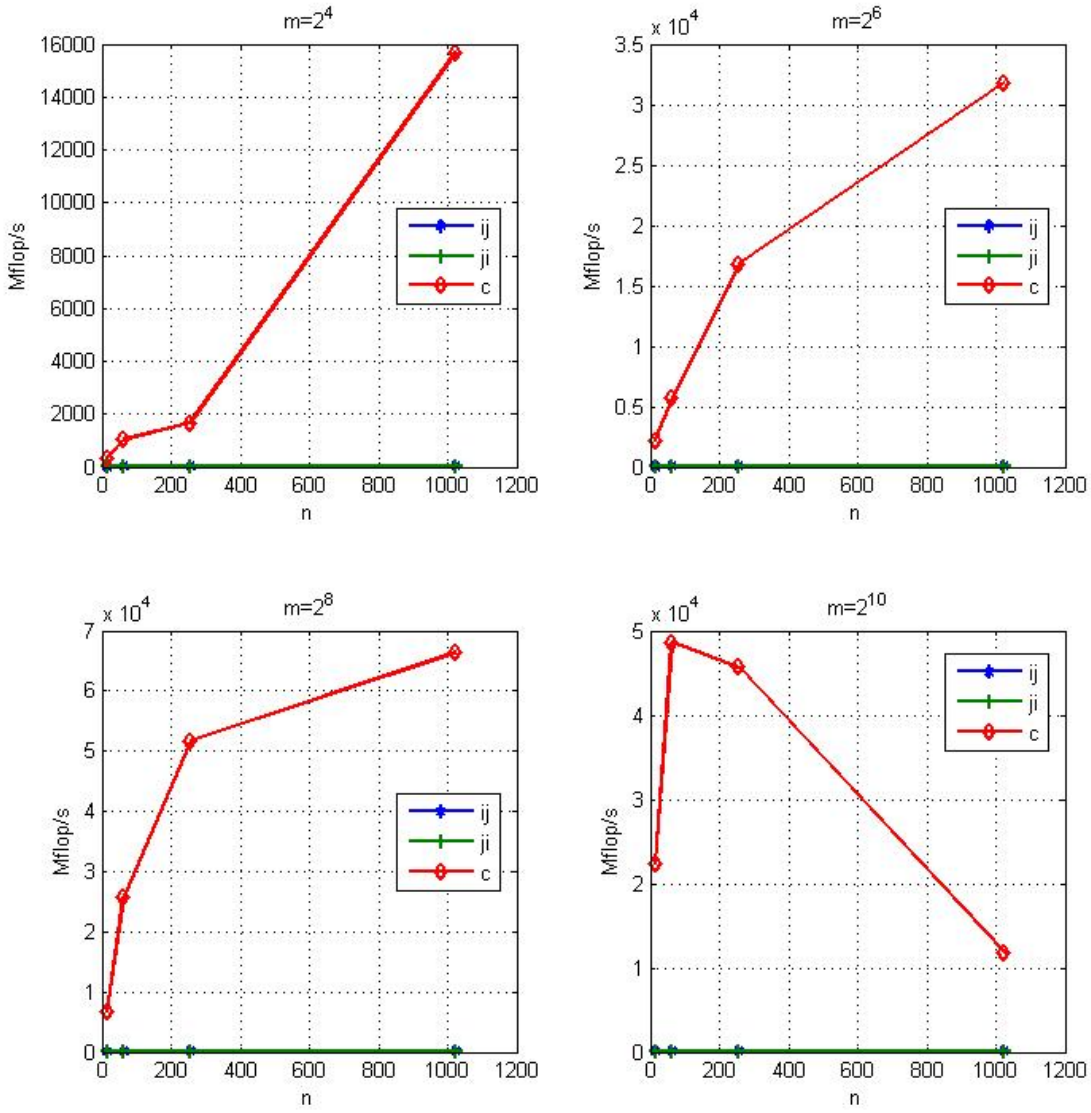
```
A = rand(m,n);
b = rand(n,1);
h_ij = @() mv_ij(A,b);
h_ji = @() mv_ji(A,b);
h_c = @() mtimes(A,b);
time_ij = timeit(h_ij);
time_ji = timeit(h_ji);
time_c = timeit(h_c);
total_flops = 2*m*n;
mflops_ij = (total_flops/time_ij)*10e-6;
mflops_ji = (total_flops/time_ji)*10e-6;
mflops_c = (total_flops/time_c)*10e-6;
disp(sprintf('Computing Matrix Vector Multiplication c = A*b'))
disp(sprintf(' Number of Matrix rows m = %i',m))
disp(sprintf(' Number of Matrix columns n = %i',n))
disp(sprintf(' Number of floating point operations =
%i\n',total_flops))
disp(sprintf(' Method\t\tCpu Seconds\t\tMegaFlops'))
disp(sprintf(' --------\t--------------\t--------------'))
disp(sprintf(' IJ\t\t\t%d\t%i',time_ij,round(mflops_ij)))
disp(sprintf(' JI\t\t\t%d\t%i',time_ji,round(mflops_ji)))
disp(sprintf(' Ac\t\t\t%d\t%i',time_c,round(mflops_c)))
```

(iii) Για την αναπαράσταση σε figures των παραπάνω συναρτήσεων για τις διαστάσεις που δίνονται στην εκφώνηση, ο κώδικας είναι ο εξής:

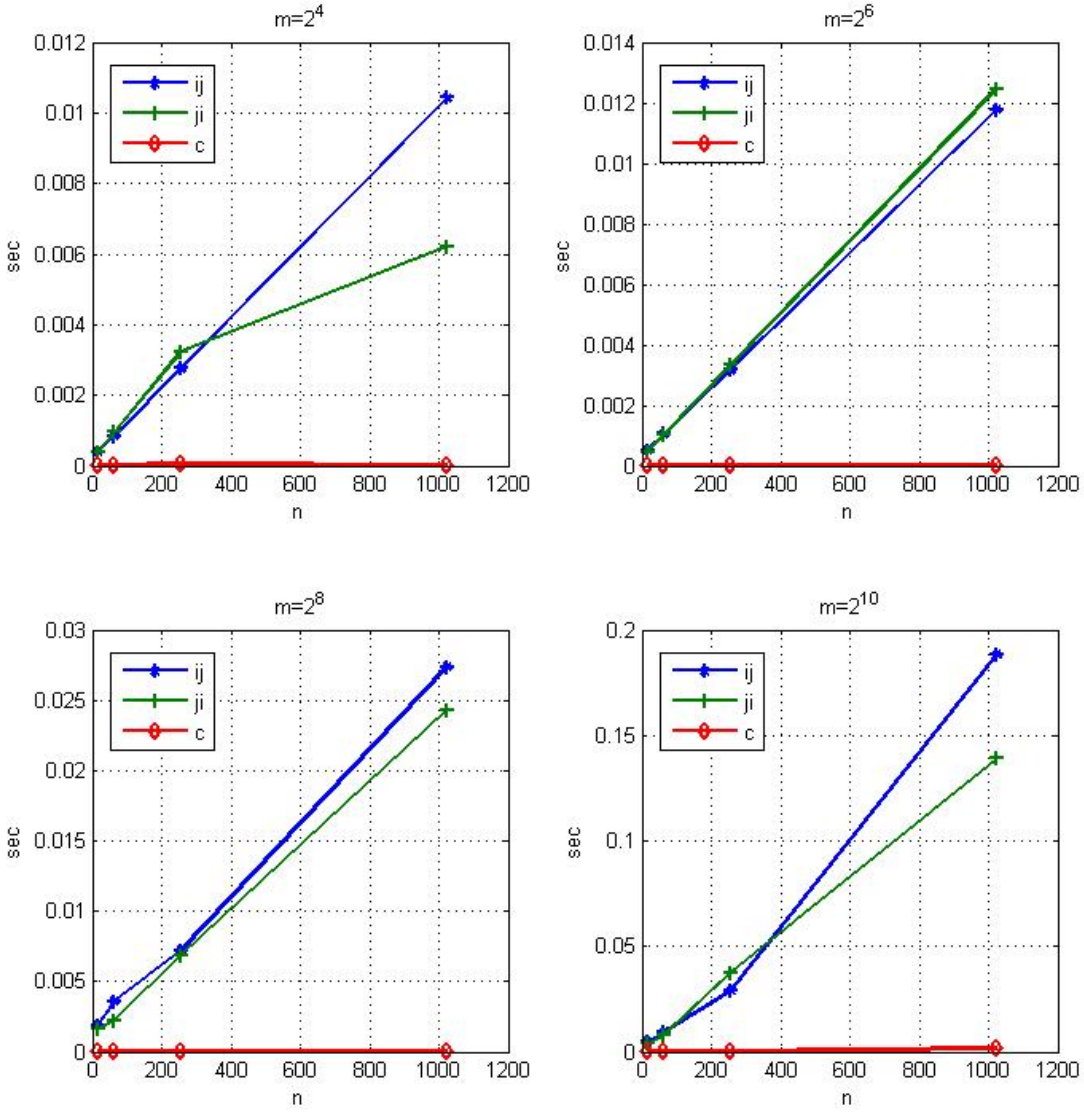
```
m = 2.^[4:2:10];
n = m;
time_ij = zeros(4,4);
time_ji = zeros(4,4);
time_c = zeros(4,4);
total_flops = zeros(4,4);
mflops_ij = zeros(4,4);
mflops_ji = zeros(4,4);
mflops_c = zeros(4,4);
for x=1:4
    for y=1:4
        A = rand(m(x),n(y));
        b = rand(n(y),1);
        h_ij = @() mv_ij(A,b);
        h_ji = @() mv_ji(A,b);
        h_c = @() mtimes(A,b);
        time_ij(x,y) = timeit(h_ij);
        time_ji(x,y) = timeit(h_ji);
        time_c(x,y) = timeit(h_c);
        total_flops(x,y) = 2*m(x)*n(y);
        mflops_ij(x,y) = (total_flops(x,y)/time_ij(x,y))*10e-6;
        mflops_ji(x,y) = (total_flops(x,y)/time_ji(x,y))*10e-6;
        mflops_c(x,y) = (total_flops(x,y)/time_c(x,y))*10e-6;
        disp(sprintf('Computing Matrix Vector Multiplication c =
A*b'))
        disp(sprintf(' Number of Matrix rows m = %i',m(x)))
        disp(sprintf(' Number of Matrix columns n = %i',n(y)))
        disp(sprintf(' Number of floating point operations =
%i\n',total_flops(x,y)))
        disp(sprintf(' Method\t\tCpu Seconds\t\tMegaFlops'))
        disp(sprintf(' --------\t--------------\t--------------
'))
        disp(sprintf('
IJ\t\t\t%d\t%i',time_ij(x,y),round(mflops_ij(x,y))))
        disp(sprintf('
JI\t\t\t%d\t%i',time_ji(x,y),round(mflops_ji(x,y))))
        disp(sprintf('
Ac\t\t\t%d\t%i',time_c(x,y),round(mflops_c(x,y))))
    end
end
```

```
figure;
subplot(2,2,1);
plot(n,time_ij(1,:),'*-',n,time_ji(1,:),'+-',n,time_c(1,:),'d-
','LineWidth',2);
title('m=2^4'); xlabel('n'); ylabel('sec');
legend('ij','ji','c','Location','Best');
grid on;
subplot(2,2,2);
plot(n,time_ij(2,:),'*-',n,time_ji(2,:),'+-',n,time_c(2,:),'d-
','LineWidth',2);
title('m=2^6'); xlabel('n'); ylabel('sec');
legend('ij','ji','c','Location','Best');
grid on;
subplot(2,2,3);
plot(n,time_ij(3,:),'*-',n,time_ji(3,:),'+-',n,time_c(3,:),'d-
','LineWidth',2);
title('m=2^8'); xlabel('n'); ylabel('sec');
legend('ij','ji','c','Location','Best');
grid on;
subplot(2,2,4);
plot(n,time_ij(4,:),'*-',n,time_ji(4,:),'+-',n,time_c(4,:),'d-
','LineWidth',2);
title('m=2^{10}'); xlabel('n'); ylabel('sec');
legend('ij','ji','c','Location','Best');
grid on;
%
figure;
subplot(2,2,1);
plot(n,mflops_ij(1,:),'*-',n,mflops_ji(1,:),'+-
',n,mflops_c(1,:),'d-','LineWidth',2);
title('m=2^4'); xlabel('n'); ylabel('Mflop/s');
legend('ij','ji','c','Location','Best');
grid on;
subplot(2,2,2);
plot(n,mflops_ij(2,:),'*-',n,mflops_ji(2,:),'+-
',n,mflops_c(2,:),'d-','LineWidth',2);
title('m=2^6'); xlabel('n'); ylabel('Mflop/s');
legend('ij','ji','c','Location','Best');
grid on;
subplot(2,2,3);
plot(n,mflops_ij(3,:),'*-',n,mflops_ji(3,:),'+-
',n,mflops_c(3,:),'d-','LineWidth',2);
title('m=2^8'); xlabel('n'); ylabel('Mflop/s');
legend('ij','ji','c','Location','Best');
grid on;
subplot(2,2,4);
plot(n,mflops_ij(4,:),'*-',n,mflops_ji(4,:),'+-
',n,mflops_c(4,:),'d-','LineWidth',2);
title('m=2^{10}'); xlabel('n'); ylabel('Mflop/s');
legend('ij','ji','c','Location','Best');
grid on;
```

Η γραφική αναπαράσταση των αποτελεσμάτων για τον αριθμό των πράξεων ανά δευτερόλεπτο είναι η ακόλουθη:



Και στη συνέχεια παραθέτω τη γραφική αναπαράσταση των αποτελεσμάτων για το χρόνο εκτέλεσης των συναρτήσεων.



(iv) Θα κάνω τώρα ένα γενικότερο σχολιασμό που περιλαμβάνει και τα δύο (α) και (β) υποερωτήματα.

Παρατηρούμε λοιπόν πως η ενδογενής συνάρτηση πολλαπλασιασμού είναι η πιο γρήγορη χρονικά από τις τρεις, για όλα τα μεγέθη των μητρώων, καθώς επίσης ότι και τα MFlop/s της σε κάθε περίπτωση (δηλαδή η ταχύτητα εκτέλεσης των πράξεων) είναι περισσότερα από των άλλων δύο αριθμητικά. Σύμφωνα με αυτά λοιπόν, θα επιλέγαμε την ενδογενή συνάρτηση για κάθε περίπτωης και για όλα τα μεγέθη.

Όσον αφορά τη σύγκριση μεταξύ των δύο άλλων τρόπων, των συναρτήσεων για πολλαπλασιασμό κατά γραμμές και κατά στήλες, παρατηρούμε πως σε κάποιες περιπτώσεις συμφέρει να επιλέγουμε τη μία και σε κάποιες την άλλη. Για παράδειγμα όταν $m=2^4$ και $m=2^{10}$, στο διάστημα που το n παίρνει τιμές από 0 μέχρι περίπου 300, συμφέρει να επιλέξουμε τον πολλαπλασιασμό κατά γραμμές, μιας και χρειάζεται λιγότερο χρόνο για να υλοποιηθεί, ενώ στη συνέχεια που το n αυξάνεται κι άλλο, συμφέρει ο πολλαπλασιασμός κατά στήλες.

Ερώτημα 4 – Εμπειρική Εκτίμηση Επίδοσης:

(i)-(ii) Μιας και ο κώδικας του πρώτου ερωτήματος συμπεριλαμβάνεται στην υλοποίηση του $2^{ου}$, ένωσα τα δύο υποερωτήματα του $4^{ου}$ και τελευταίου μέρους της άσκησης. Στη συνέχεια λοιπόν παραθέτω τον κώδικα, στον οποίο έχω προσθέσει και τις ανάλογες εντολές για να πάρω και γραφικές παραστάσεις που αναπαριστούν τα αποτελέσματα που καλούμαι να σχολιάσω στο $3^{ο}$ ερώτημα. Ο κώδικας λοιπόν είναι ο εξής:

```
n = [50 100 200 400 800];
time_c = zeros(1,5);
for i=1:5
    A = rand(n(i));
    b = rand(n(i),1);
    h_c = @() mtimes(A,b);
    time_c(i) = timeit(h_c,1);
end
%
A1 = polyfit(n,time_c,1);
A2 = polyfit(n,time_c,2);
A3 = polyfit(n,time_c,3);
A4 = polyfit(n,time_c,4);
%
B1 = polyval(A1,n);
B2 = polyval(A2,n);
B3 = polyval(A3,n);
B4 = polyval(A4,n);
%
time_c1 = time_c./n;
time_c2 = time_c./(n.^2);
time_c3 = time_c./(n.^3);
time_c4 = time_c./(n.^4);
%
```

```
figure
subplot(2,2,1);
semilogy(n,time_c1,'*-','LineWidth',2);
title('t/n'); xlabel('n'); ylabel('sec');
subplot(2,2,2);
semilogy(n,time_c2,'*-','LineWidth',2);
title('t/n^2'); xlabel('n'); ylabel('sec');
subplot(2,2,3);
semilogy(n,time_c3,'*-','LineWidth',2);
title('t/n^3'); xlabel('n'); ylabel('sec');
subplot(2,2,4);
semilogy(n,time_c4,'*-','LineWidth',2);
title('t/n^4'); xlabel('n'); ylabel('sec');
```

Παραθέτω τώρα τα πολυώνυμα που προκύπτουν από την polyfit:

$1^{ης}$ τάξης πολυώνυμο : $(0.0020n - 0.2400)x10^{-3}$
$2^{ης}$ τάξης πολυώνυμο : $(0.0000n^2 - 0.0011n + 0.1119)x10^{-3}$
$3^{ης}$ τάξης πολυώνυμο : $(0.0000n^3 - 0.0001n^2 + 0.0149n - 0.4680)x10^{-4}$
$4^{ης}$ τάξης πολυώνυμο : $(0.0000n^4 - 0.0000n^3 + 0.0002n^2 + 0.0463n - 0.7275)x10^{-5}$

Αντίστοιχα, ακολουθούν τα αποτελέσματα των παραπάνω πολυωνύμων με τη χρήση της polyval, ώστε να δούμε πόσο κοντά είναι με τις αρχικές μας μετρήσεις.

sec/n	50	100	200	400	800
time_c	0.0000	0.0001	0.0001	0.0002	0.0016
B1	-0.0001	-0.0000	0.0002	0.0006	0.0014
B2	0.0001	0.0000	0.0000	0.0003	0.0016
B3	0.0000	0.0001	0.0001	0.0002	0.0016
B4	0.0000	0.0001	0.0001	0.0002	0.0016

Να σημειώσω πως σε αρκετές περιπτώσεις τα παραπάνω νούμερα είναι στρογγυλοποιήσεις από τη Matlab, και δε σημαίνει πως είναι ουσιαστικά μηδενικά πχ αυτά που παρουσιάζονται ως 0.0000. Πιο συγκεκριμένο παράδειγμα είναι η τιμή της time_c για n = 50, που ενώ φαίνεται μηδενική, στην πραγματικότητα είναι ίση με $2.0020x10^{-5}$. Αυτό ισχύει και για τους συντελεστές των πολυονύμων που έχουν υπολογιστεί.

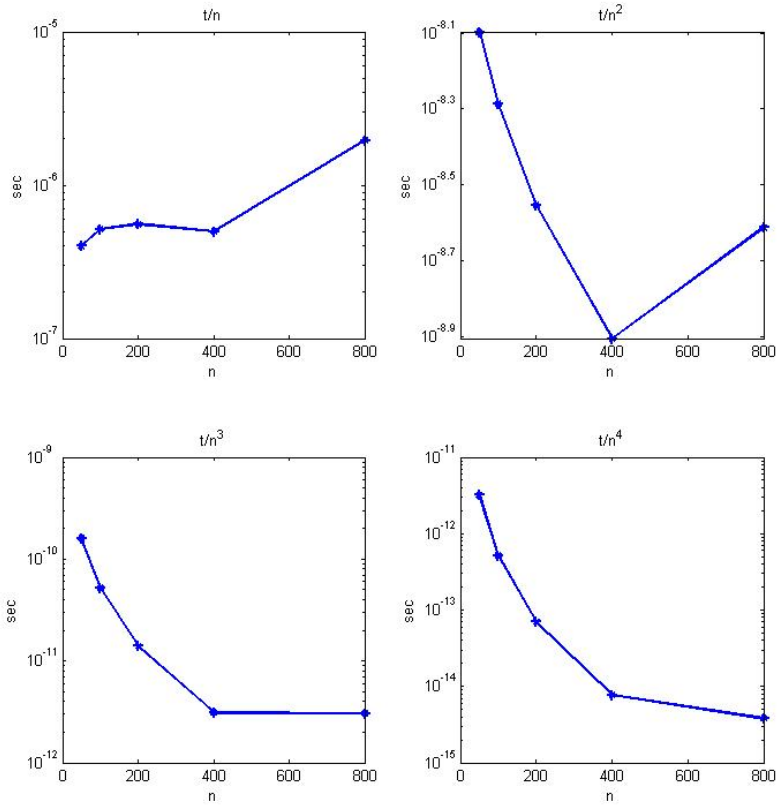
(iii) Κατά την εκτέλεση των παραπάνω προέκυψε το εξής warning:

Warning: Polynomial is badly conditioned. Add points with distinct X values, reduce the degree of the polynomial, or try centering and scaling as described in HELP POLYFIT.
> In polyfit at 76
  In erwthma4 at 13

Στη γραμμή 13 του κώδικά μου βρίσκεται η polyfit για το πολυώνυμο 4ης τάξης. Το warning μας λέει ότι το πολυώνυμό μας είναι κακώς ορισμένο και μας προτείνει κάποιους τρόπους για να λύσουμε το πρόβλημα. Λόγω αυτού υποψιαζόμαστε πως το πολυώνυμο $4^{ης}$ τάξης παρ'όλο που τα αποτελέσματά του είναι πολύ κοντά στα αποτελέσματα της polyval, δεν κάνει ως μαθηματικό μοντέλο γι αυτό που ψάχνουμε.

Για να επαληθεύσουμε αυτή την υποψία ακολουθούμε τη μέθοδο που υποδείχτηκε και στο μάθημα και στο εργαστήριο, για να βρούμε την πολυπλοκότητα του πολυονύμου που είναι καταλληλότερο για τις αρχικές τιμές που έχουμε.

Τέλος παραθέτω τα αποτελέσματα στις ακόλουθες γραφικές παραστάσεις:



Παρατηρούμε λοιπόν πως η γραφική μας παράσταση σταθεροποιείται για βαθμό $3^{ης}$ τάξης. Συνεπώς θα επιλέξουμε το πολυώνυμο $3^{ης}$ τάξης και όχι της $4^{ης}$.