ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΠΙΣΤΗΜΟΝΙΚΟΣ ΥΠΟΛΟΓΙΣΜΟΣ Ι

ΠΡΩΤΗ ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Καφφέζας Γιώργος · ΑΜ 4465 · kaffezas@ceid.upatras.gr

×

Η αναφορά αυτή αφορά την πρώτη εργαστηριακή άσκηση του μαθήματος «Επιστημονικού Υπολογισμού» για το χειμερινό εξάμηνο του ακαδημαϊκού έτους 2013-2014. Συντάχθηκε με τη βοήθεια του LaTeX και του editor TeXMaker, ενώ το εξώφυλλο βασίστηκε σε κώδικα του Peter Wilson από την ιστοσελίδα www.latextemplates.com. Επιλέχθηκε πρότυπο βιβλίου και η σελιδοποίηση έγινε ώστε η αναφορά να εκτυπώνεται καλά σε μορφή φυλλαδίου μεγέθους A4.

×

ΕΡΩΤΗΜΑ 1 - Εισαγωγικά

i) Τα στοιχεία του υπολογιστικού συστήματος στο οποίο πραγματοποιήθηκε η άσκηση είναι τα ακόλουθα, όπως προκύπτουν και μετά από την χρήση των προγραμμάτων CPU-Z και PC Wizard:

Λειτουργικό σύστημα Windows 7 Professional SP1 ($\times 64$)

Τύπος επεξεργαστή Intel Core 2 Quad Q6600 @2.40GHz

Επίπεδα κρυφής μνήμης
L1 data cache: 4×32 KB, 8-way set assoc, 64-byte line size
L1 instruction cache: 4×32 KB, 8-way set assoc, 64-byte line size
L2 cache: 2×4096 KB, 16-way set assoc, 64-byte line size

Πολιτική εγγραφής στην cache write-back

ii) Η έκδοση MATLAB που χρησιμοποιήθηκε για την άσκηση είναι η R2012b (8.0.0.783) για λειτουργικό σύστημα 64-bit.

iii) Ακολουθώντας τη μέθοδο που περιλαμβάνεται στον κώδικα της συνάρτησης timeit.m και τα όσα έχουν ειπωθεί στο μάθημα και στο φροντιστήριο, εκτέλεσα τις εντολές tic και toc για 10 επαναλήψεις και για 140 φορές σε κάθε επανάληψη. Μετά από επιλογή του μικρότερου των αποτελεσμάτων, η διακριτότητα του χρονομετρητή φαίνεται ότι κυμαίνεται στα 1.1886×10^{-7} sec. Οι κώδικες με τους οποίους έγινε ο έλεγχος είναι οι εξής:

```
function [t] = tictoc_time_experiment
% Initializing elapsed variable.
elapsed = 0;
% Calling tic/toc 140 times.
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
```

```
tic; elapsed = elapsed + toc;
tic; elapsed = elapsed + toc;
% Return the average time.
t = elapsed / 140;
```

files/1/iii/tictoc_time_experiment.m

```
% Warming up tic/toc.
tic; toc; tic; toc; tic; toc;
% Specifying number of total repeats.
num_rep = 10;
% Initialization of matrix for results.
times = zeros(1,num_rep);
% Main loop.
for k = 1:num_rep
    times(k) = tictoc_time_experiment();
end
% Choose the lower result.
t = min(times)
```

files/1/iii/script_1_1_iii.m

iv) Το αποτέλεσμα της εντολής bench για τη διάσπαση μητρώων LU στο παραπάνω σύστημα και με την προαναφερθείσα έκδοση MATLAB ήταν 0.1298 sec.

ΕΡΩΤΗΜΑ 2 - Αξιολόγηση Ενδογενών Συναρτήσεων

i) Οι βασικές πράξεις γραμμικής άλγεβρας που μας ζητείται να εξηγήσουμε είναι οι ακόλουθες:

lu Η συνάρτηση αυτή υλοποιεί την παραγοντοποίηση LU, η οποία προέρχεται από την απαλοιφή Gauss[1] και χρησιμοποιείται συνήθως από υπολογιστές για την επίλυση γραμμικών συστημάτων της μορφής $Ax = b$. Ο πίνακας L είναι κάτω (lower) τριγωνικός και περιέχει τους πολλαπλασιαστές της απαλοιφής Gauss. Πιο συγκεκριμένα, στη διαγώνιό του περιέχει μονάδες ενώ στις υπόλοιπες μη-μηδενικές θέσεις τους πολλαπλασιαστές που αναφέραμε. Ο πίνακας U είναι άνω (upper) τριγωνικός που προκύπτει από την απαλοιφή Gauss και έχει στη διαγώνιό του τους αντίστοιχους οδηγούς. Στη γενικότερη μορφή της ο τύπος της είναι $PA = LU$, όπου ο P είναι ένας πίνακας μεταθέσεων που αντιστοιχεί στις εναλλαγές γραμμών κατά την απαλοιφή, και όταν δεν γίνεται κάποια ισχύει ότι $P = I$.

qr Η συνάρτηση αυτή υλοποιεί την παραγοντοποίηση QR ή αλλιώς εφαρμόζει τη μέθοδο Gram-Schmidt σε έναν πίνακα A με γραμμικά ανεξάρτητες στήλες.[2] Η παραγοντοποίηση QR χρησιμοποιείται συχνά για την επίλυση του προβλήματος των ελαχίστων τετραγώνων και αποτελεί τη βάση για τον αλγόριθμο QR, έναν συγκεκριμένο αλγόριθμο ιδιοτιμών.[3] Μέσω αυτής της διαδικασίας και δοθέντος ενός τετραγωνικού πίνακα A, υπολογίζουμε έναν άνω τριγωνικό πίνακα R και έναν ορθογώνιο πίνακα Q, έτσι ώστε $A = QR$.

svd Η συνάρτηση αυτή υλοποιεί την ανάλυση ιδιαζουσών τιμών (αγγλιστί singular value decomposition), η οποία αποτελεί ένα από τα πιο σημαντικά αποτελέσματα της γραμμικής άλγεβρας[4] και χρησιμοποιείται συχνά στην επεξεργασία σημάτων και στη στατιστική ανάλυση.[5] Δεδομένου ενός πίνακα A, τετραγωνικού ή ορθογώνιου σχήματος, η SVD μας επιστρέφει έναν διαγώνιο πίνακα S, ίδιων διαστάσεων με τον A, και τους ορθογώνιους πίνακες U και V, έτσι ώστε να ισχύει $A = USV'$.

eig Η συνάρτηση αυτή, ανάλογα με τα ορίσματα, μπορεί να υπολογίσει τις ιδιοτιμές και τα ιδιοδιανύσματα ενός τετραγωνικού πίνακα A, όπου «ιδιοδιάνυσμα» είναι ένα μη-μηδενικό διάνυσμα v που όταν πολλαπλασιαστεί με τον A ισούται με το αρχικό διάνυσμα, πολλαπλασιασμένο με έναν αριθμό λ, την λεγόμενη «ιδιοτιμή» του A που αντιστοιχεί στο v, έτσι ώστε $Av = \lambda v$.

ii) Για την πραγματοποίηση των μετρήσεων δημιουργούμε σε κάθε ερώτημα τυχαία μητρώα μεγέθους 200×200 έως 1200×1200 με βήμα 200, όπως υποδεικνύεται στην εκφώνηση, με χρήση της συνάρτησης randn. Οι κώδικες κάθε ερωτήματος (μαζί με το κομμάτι που υλοποιεί τις γραφικές παραστάσεις για το τρίτο ερώτημα) και τα αντίστοιχα αποτελέσματα παρατίθενται στη συνέχεια.

[1] G. Strang, «Εισαγωγή στη Γραμμική Άλγεβρα», σ. 105
[2] ό.π., σσ. 284-286
[3] en.wikipedia.org/wiki/QR_decomposition
[4] G. Strang, «Εισαγωγή στη Γραμμική Άλγεβρα», σ. 443
[5] en.wikipedia.org/wiki/Singular_value_decomposition

α) Ο κώδικας για το παρόν ερώτημα και οι χρόνοι εκτέλεσης είναι:

```
% Variable for random matrix sizes.
matrix_size = 200:200:1200;
% A little warm-up for tic-toc.
tic; toc;

% Initialization of rt_lu to store results.
rt_lu = zeros(1,6);
for j = 1:6 % Loop for lu(A).
    % Random matrix and lu(A) warm-up.
    A = randn(matrix_size(j)); [L,U] = lu(A);
    % Time calculation with tic-toc.
    tic; [L,U] = lu(A); rt_lu(j) = toc;
end
% Initialization of rt_qr to store results.
rt_qr = zeros(1,6);
for j = 1:6 % Loop for qr(A).
    % Random matrix and qr(A) warm-up.
    A = randn(matrix_size(j)); [Q,R] = qr(A);
    % Time calculation with tic-toc.
    tic; [Q,R] = qr(A); rt_qr(j) = toc;
end
% Initialization of rt_svd to store results.
rt_svd = zeros(1,6);
for j = 1:6 % Loop for svd(A).
    % Random matrix and svd(A) warm-up.
    A = randn(matrix_size(j)); [U,S,V] = svd(A);
    % Time calculation with tic-toc.
    tic; [U,S,V] = svd(A); rt_svd(j) = toc;
end
% Initialization of rt_eig to store results.
rt_eig = zeros(1,6);
for j = 1:6 % Loop for eig(A).
    % Random matrix and eig(A) warm-up.
    A = randn(matrix_size(j)); [V,D] = eig(A);
    % Time calculation with tic-toc.
    tic; [V,D] = eig(A); rt_eig(j) = toc;
end

figure; % Plotting the results.
plot(matrix_size,rt_lu,'b*-',...
     matrix_size,rt_qr,'r+-',...
     matrix_size,rt_svd,'gx-',...
     matrix_size,rt_eig,'cd-',...
     'LineWidth',2);
title('2-ii-a','fontWeight','bold');
xlabel('Matrix size n');
ylabel('Runtime (sec)');
legend('lu(A)','qr(A)','svd(A)','eig(A)',...
        'Location','NorthWest');
grid on;
```

files/2/ii/script_1_2_ii_a.m

	n=200	**n=400**	**n=600**	**n=800**	**n=1000**	**n=1200**
rt_lu	0.0015	0.0055	0.0132	0.0262	0.0468	0.0714
rt_qr	0.0044	0.0178	0.0473	0.0946	0.1599	0.2629
rt_svd	0.0257	0.1162	0.2911	0.5593	0.9633	1.7669
rt_eig	0.1099	0.3424	0.9330	1.7727	3.1803	5.2603

Στο ερώτημα αυτό τρέχουμε στην αρχή μια φορά τις εντολές tic και toc για προθέρμανση, και στη συνέχεια εκτελούμε κάθε μία από τις υπό εξέταση εντολές, μια φορά για «προθέρμανση» και από μία φορά για κάθε είσοδο. Εν συνεχεία, αποθηκεύουμε τα αποτελέσματα σε ένα μητρώο και σχεδιάζουμε στο ίδιο σχεδιάγραμμα τις γραφικές αναπαραστάσεις που προκύπτουν.

β) Ο κώδικας για το παρόν ερώτημα και οι χρόνοι εκτέλεσης είναι:

```
% Variable for random matrix sizes.
matrix_size = 200:200:1200;
% A little warm-up for tic-toc.
tic; toc;

% Initialization of rt_lu to store results.
rt_lu = zeros(1,6);
for j = 1:6 % Loop for lu(A).
    A = randn(matrix_size(j));
    % lu(A) warm-up and initialization.
    [L,U] = lu(A); sum = 0;
    for i = 1:20 % Counting 20 times.
        % Time calculation with tic-toc.
        tic; [L,U] = lu(A);
        sum = sum + toc; % Increasing sum.
    end
    rt_lu(j) = sum / 20; % Calcuating median.
end
% Initialization of rt_qr to store results.
rt_qr = zeros(1,6);
for j = 1:6 % Loop for qr(A).
    A = randn(matrix_size(j));
    % qr(A) warm-up and initialization.
    [Q,R] = qr(A); sum = 0;
    for i = 1:20 % Counting 20 times.
        % Time calculation with tic-toc.
        tic; [Q,R] = qr(A);
        sum = sum + toc; % Increasing sum.
    end
    rt_qr(j) = sum / 20; % Calcuating median.
end
% Initialization of rt_svd to store results.
rt_svd = zeros(1,6);
for j = 1:6 % Loop for svd(A).
    A = randn(matrix_size(j));
    % svd(A) warm-up and initialization.
    [U,S,V] = svd(A); sum = 0;
    for i = 1:20 % Counting 20 times.
        % Time calculation with tic-toc.
        tic; [U,S,V] = svd(A);
        sum = sum + toc; % Increasing sum.
    end
    rt_svd(j) = sum / 20; % Calcuating median.
end
% Initialization of rt_eig to store results.
rt_eig = zeros(1,6);
for j = 1:6 % Loop for eig(A).
    A = randn(matrix_size(j));
    % eig(A) warm-up and initialization.
    [V,D] = eig(A); sum = 0;
    for i = 1:20 % Counting 20 times.
```

```
        % Time calculation with tic-toc.
        tic; [V,D] = eig(A);
        sum = sum + toc; % Increasing sum.
    end
    rt_eig(j) = sum / 20; % Calcuating median.
end

figure; % Plotting the results.
plot(matrix_size,rt_lu,'b*-',...
     matrix_size,rt_qr,'r+-',...
     matrix_size,rt_svd,'gx-',...
     matrix_size,rt_eig,'cd-',...
     'LineWidth',2);
title('2-ii-b','fontWeight','bold');
xlabel('Matrix size n');
ylabel('Runtime (sec)');
legend('lu(A)','qr(A)','svd(A)','eig(A)',...
       'Location','NorthWest');
grid on;
```

files/2/ii/script_1_2_ii_b.m

	n=200	**n=400**	**n=600**	**n=800**	**n=1000**	**n=1200**
rt_lu	0.0014	0.0070	0.0175	0.0323	0.0569	0.0936
rt_qr	0.0050	0.0216	0.0581	0.1155	0.2033	0.3396
rt_svd	0.0340	0.1516	0.3877	0.8191	1.3154	2.1085
rt_eig	0.1266	0.4462	1.2190	2.3042	4.0377	6.3840

Στο ερώτημα αυτό ακολουθείται η ίδια τακτική με το προηγούμενο ερώτημα, μόνο που τώρα αντί για μία φορά εκτελούμε την κάθε εντολή 20 φορές και υπολογίζεται ο μέσος όρος των εκτελέσεων. Όπως ειπώθηκε και στο μάθημα, όσο μεγαλύτερος αριθμός επαναλήψεων, τόσο μεγαλύτερη η ακρίβεια των αποτελεσμάτων, οπότε ο αριθμός 20 επιλέχτηκε με το σκεπτικό του ότι δεν είναι ότι πολύ μικρός, αλλά ούτε και πολύ μεγάλος (και έτσι, ιδιαίτερα χρονοβόρος). Παρόλο που οι πράξεις που αφορούν στον υπολογισμό SVD και των ιδιοτιμών καταναλώνουν περισσότερο χρόνο, δεν επιλέχτηκαν λιγότερες επαναλήψεις για τις μετρήσεις, καθώς ο συνολικός χρόνος εκτέλεσής τους στο χρησιμοποιηθέν υπολογιστικό σύστημα ήταν σχετικά σύντομος. Έτσι, παρέχεται περίπου η ίδια ακρίβεια για όλες τις παραπάνω μετρήσεις.

γ) Ο κώδικας για το παρόν ερώτημα και οι χρόνοι εκτέλεσης είναι:

```
% Variable for random matrix sizes.
matrix_size = 200:200:1200;

% Initialization of rt_lu to store results.
rt_lu = zeros(1,6);
for j = 1:6 % Loop for lu(A).
    % Random matrix.
    A = randn(matrix_size(j));
    % Time calculation with timeit.
    x = @() lu(A);
    rt_lu(j) = timeit(x,2);
end
% Initialization of rt_qr to store results.
```

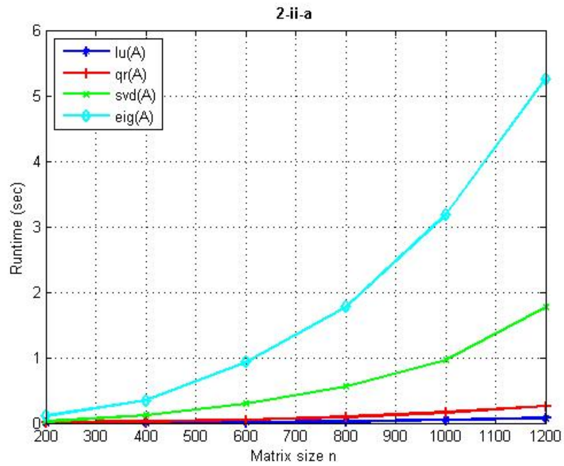
```
rt_qr = zeros(1,6);
for j = 1:6 % Loop for qr(A).
    % Random matrix.
    A = randn(matrix_size(j));
    % Time calculation with timeit.
    x = @() qr(A);
    rt_qr(j) = timeit(x,2);
end
% Initialization of rt_svd to store results.
rt_svd = zeros(1,6);
for j = 1:6 % Loop for svd(A).
    % Random matrix.
    A = randn(matrix_size(j));
    % Time calculation with timeit.
    x = @() svd(A);
    rt_svd(j) = timeit(x,3);
end
% Initialization of rt_eig to store results.
rt_eig = zeros(1,6);
for j = 1:6 % Loop for eig(A).
    % Random matrix.
    A = randn(matrix_size(j));
    % Time calculation with timeit.
    x = @() eig(A);
    rt_eig(j) = timeit(x,2);
end

figure; % Plotting the results.
plot(matrix_size,rt_lu,'b*-',...
     matrix_size,rt_qr,'r+-',...
     matrix_size,rt_svd,'gx-',...
     matrix_size,rt_eig,'cd-',...
     'LineWidth',2);
title('2-ii-c','fontWeight','bold');
xlabel('Matrix size n');
ylabel('Runtime (sec)');
legend('lu(A)','qr(A)','svd(A)','eig(A)',...
       'Location','NorthWest');
grid on;
```

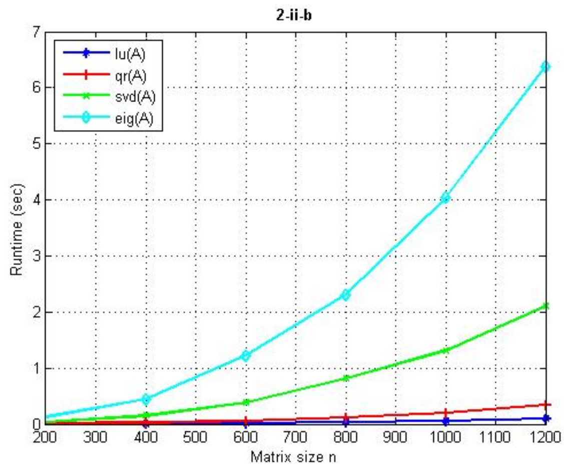
files/2/ii/script_1_2_ii_c.m

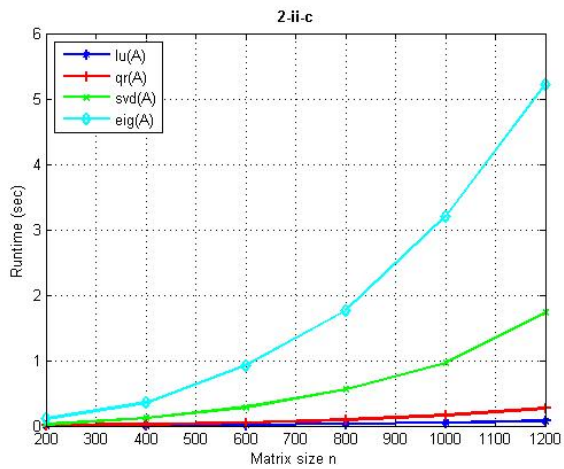
	n=200	**n=400**	**n=600**	**n=800**	**n=1000**	**n=1200**
rt_lu	0.0013	0.0058	0.0135	0.0265	0.0466	0.0731
rt_qr	0.0042	0.0180	0.0468	0.0932	0.1620	0.2640
rt_svd	0.0250	0.1181	0.2853	0.5535	0.9589	1.7321
rt_eig	0.1123	0.3563	0.9244	1.7631	3.2017	5.2269

Στο τελευταίο ερώτημα μετρήσεων έγινε χρήση της συνάρτησης timeit, όπως προτείνεται και στην εκφώνηση. Έτσι, δεν κάναμε κάποια «προθέρμανση», ενώ ο αριθμός των επαναλήψεων για κάθε εντολή καθορίζεται από αυτήν, και αποθηκεύεται κατευθείαν το τελικό αποτέλεσμα που μας ενδιαφέρει.

iii) Οι ζητούμενες γραφικές παραστάσεις παρατίθενται στη συνέχεια με την α-νάλογη σειρά. Όπως είναι προφανές και από τα υπομνήματα σε κάθε σχήμα, ισχύει η εξής αντιστοιχία συνάρτησης-χρώματος-σημείων: για την lu(A) σκούρο μπλε χρώμα, αστεράκια ως σημεία, για την qr(A) κόκκινο χρώμα και το σύμβολο «+», για την svd(A) ανοιχτό πράσινο χρώμα και το σύμβολο «x», και για την eig(A) ανοιχτό γαλάζιο χρώμα και ρόμβοι (η αλλιώς διαμάντια).







Όσον αφορά τον σχολιασμό των αποτελεσμάτων, τώρα. Σα μια πρώτη και πρόχειρη παρατήρηση, θα λέγαμε ότι η lu(A) εκτελείται πιο γρήγορα σε σχέση με την qr(A), η οποία με τη σειρά της εκτελείται πιο γρήγορα από την svd(A), η οποία πάλι εκτελείται πιο γρήγορα από την eig(A), πάντα για το ίδιο μέγεθος μητρώου εισόδου. Αυτό οφείλεται στην διαφορά μεταξύ της πολυπλοκότητας των πράξεων αυτών.

Σαν μια πιο γενική παρατήρηση, θα μπορούσαμε να πούμε ότι ο χρόνος εκτέλεσης κάθε εντολής αυξάνεται καθώς αυξάνεται και το μέγεθος του μητρώου εισόδου. Πιο αναλυτικά, όμως, θα λέγαμε ότι παρατηρούμε έντονες αυξήσεις κυρίως στις συναρτήσεις που αφορούν τον υπολογισμό SVD (svd(A)) και τις ιδιοτιμές (eig(A)). Ο χρόνος εκτέλεσης της svd(A) βλέπουμε ότι αυξάνεται με μεγαλύτερο ρυθμό κατά τη μεταβολή του μεγέθους του μητρώου εισόδου από 1000×1000 σε 1200×1200. Αντίστοιχα, ο χρόνος εκτέλεσης της eig(A) αυξάνεται έντονα αρχικά από 400×400 και ύστερα από 1000×1000 σε 1200×1200, με προφανή την ευρύτερη ισχυρή τάση αύξησής του. Αυτό οφείλεται στον τρόπο με τον οποίο υλοποιούνται οι πράξεις αυτές και στον αριθμό των πράξεων που αυξάνεται, όσο αυξάνεται το μέγεθος του μητρώου εισόδου.

Τέλος, αξίζει να πούμε ότι η χρονομέτρηση με τη χρήση της timeit φαίνεται πως έχει τα πιο αξιόπιστα αποτελέσματα, δεδομένου και του τρόπου λήψης τους. Δηλαδή, σε σχέση με τον πρώτο τρόπο μέτρησης, βλέπουμε ότι έχει παραπλήσια αποτελέσματα, με ποικίλες μικρές διαφορές μεταξύ τους. Ενώ σε σχέση με τον δεύτερο βλέπουμε ότι έχει πιο μεγάλες διαφορές, κυρίως στις μετρήσεις που αφορούσαν μεγαλύτερα μεγέθη μητρώων. Αυτό πιθανώς να οφείλεται στο ότι στον αριθμό των επαναλήψεων που είχαν επιλεχθεί από εμάς παρουσιάστηκαν και κάποιες πιο ακραίες μετρήσεις, οι οποίες δεν «αμβλύνθηκαν». Ενώ κατά τη μέτρηση με την τιμειτ αυτές είτε δεν είχαν την ευκαιρία να παρουσιαστούν, είτε εξομαλύνθηκε η επίδρασή τους στο μέσο όρο.

ΕΡΩΤΗΜΑ 3 - Σύγκριση Υλοποιήσεων

i) Οι δύο ζητούμενες συναρτήσεις πολλαπλασιασμού είναι οι εξής:

```
function [c] = mv_ij(A,b)
% Vertical size of A matrix.
m = size(A,1);
% Initialization of c to store result.
c = zeros(m,1);
% Loop to perform the multiplication.
for i = 1:m
     c(i) = c(i) + A(i,:)*b;
end
```

files/3/i/mv_ij.m

```
function [c] = mv_ji(A,b)
% Vertical & horizontal size of A.
[m,n] = size(A);
% Initialization of c to store result.
c = zeros(m,1);
% Loop to perform the multiplication.
for j = 1:n
     c = c + A(:,j)*b(j);
end
```

files/3/i/mv_ji.m

ii) Το ζητούμενο script με τα χαρακτηριστικά που προσδιορίζονται είναι το εξής:

```
% Temporary numbers to test functionality.
m = 10; n = 5;
% Creating random matrix and vector.
A = randn(m,n);
b = randn(n,1);
% Calculating execution time with timeit.
x = @() mv_ij(A,b); rt_mv_ij = timeit(x);
y = @() mv_ji(A,b); rt_mv_ji = timeit(y);
z = @() mtimes(A,b); rt_mv_ab = timeit(z);
% Calculating flops.
flops = 2 * m * n;
% Calculating MFlop/s for each function.
ij_mflops = (flops/rt_mv_ij) * 10e-6;
ji_mflops = (flops/rt_mv_ji) * 10e-6;
ab_mflops = (flops/rt_mv_ab) * 10e-6;
% Printing the requested results.
fprintf('--------------------------------------------------\n')
fprintf('Computing Matrix-Vector Multiplication c = A*b\n')
fprintf('--------------------------------------------------\n')
fprintf('>>> Number of matrix rows m: %i\n',m)
fprintf('>>> Number of matrix columns n: %i\n',n)
fprintf('>>> Number of flops: %i\n\n',flops)
fprintf('Method\t\t   CPU (sec)    \t\tMFlop/s\n')
fprintf('------\t\t------------ \t\t-------\n')
fprintf('mv_ij \t\t %d   \t\t %i\n',rt_mv_ij,round(ij_mflops))
fprintf('mv_ji \t\t %d   \t\t %i\n',rt_mv_ji,round(ji_mflops))
fprintf('mv_ab \t\t %d   \t\t %i\n',rt_mv_ab,round(ab_mflops))
```

files/3/ii/test_mv.m

Για να υπολογίσουμε τα MFlop/s θεωρούμε ότι ο αριθμός των πράξεων είναι ίδιος και για τους τρεις τρόπους, και ισούται με $2mn$. Για τους δύο πρώτους τρόπους αυτό είναι προφανές, καθώς οι πράξεις υλοποιούνται με τον τρόπο $c = c + Ab$, άρα ο αριθμός των πράξεων θα είναι $2mn - m + m$, δηλαδή $2mn$. Για την ενδογενή συνάρτηση της MATLAB θεωρούμε ότι είναι ο ίδιος, σύμφωνα και με τις οδηγίες που δόθηκαν στο εργαστήριο του μαθήματος.

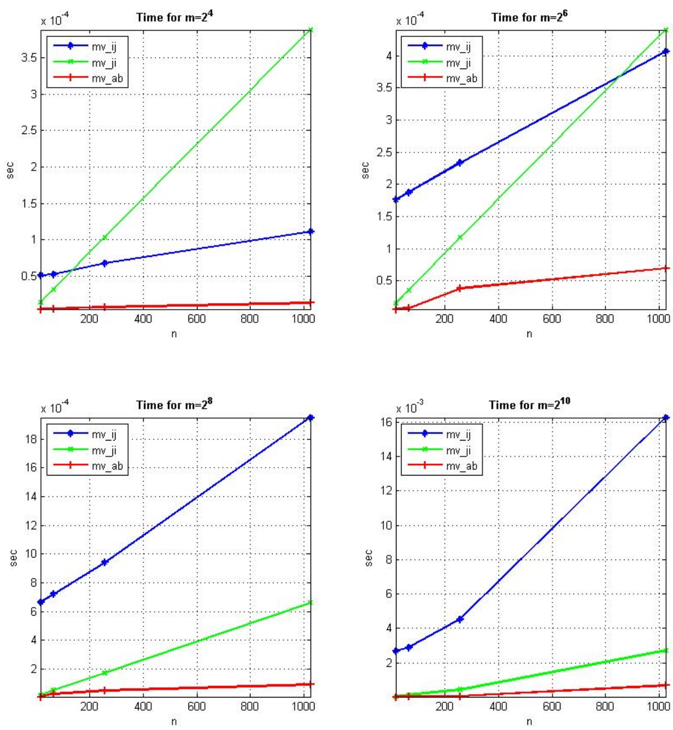
iii) Το script για το ερώτημα αυτό βασίζεται σε αυτό του προηγούμενου ερωτήματος. Έχουν γίνει κάποιες αλλαγές ώστε να εκτελείται για όλα τα ζητούμενα μεγέθη των μητρώων και διανυσμάτων εισόδου. Στο τέλος υπάρχει επίσης και το κομμάτι του κώδικα που αφορά στην σχεδίαση των γραφικών παραστάσεων που ζητούνται, βάσει των αποτελεσμάτων (που περιλαμβάνονται στο αρχείο *files/3/iii/1_3_iii_results.txt*), ενώ οι εν λόγω γραφικές παραστάσεις (για το χρόνο και για τα MFlop/s) παρατίθενται μετά από τον κώδικα.

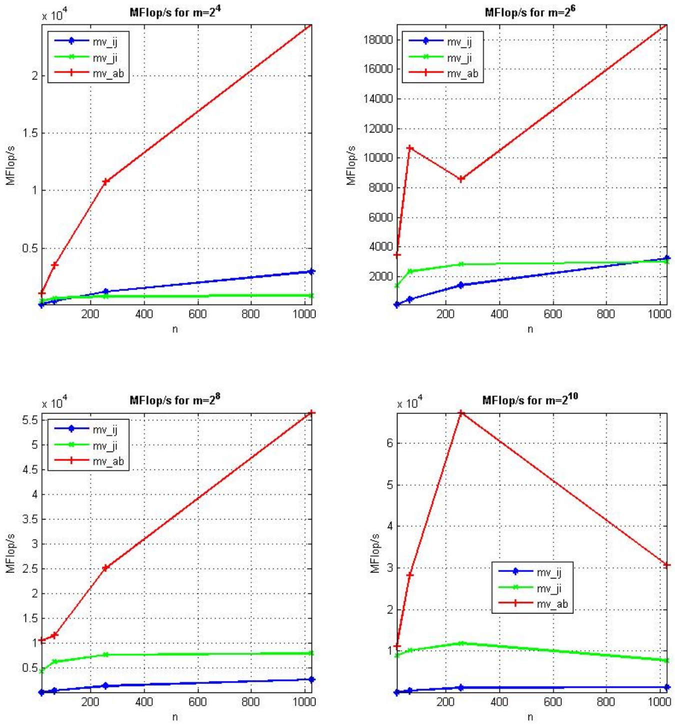
```
% Several sizes of matrices.
m = 2.^[4:2:10];
n = m;
% Initialization of matrices to save results.
rt_mv_ij = zeros(4,4);
rt_mv_ji = zeros(4,4);
rt_mv_ab = zeros(4,4);
flops = zeros(4,4);
ij_mflops = zeros(4,4);
ji_mflops = zeros(4,4);
ab_mflops = zeros(4,4);
% Loops for all possible sizes.
for k = 1:4 % 16 different pairs.
    for l = 1:4
        % Creating random matrices.
        A = randn(m(k),n(l));
        b = randn(n(l),1);
        % Calculating execution time with timeit.
        x = @() mv_ij(A,b); rt_mv_ij(k,l) =  timeit(x);
        y = @() mv_ji(A,b); rt_mv_ji(k,l) = timeit(y);
        z = @() mtimes(A,b); rt_mv_ab(k,l) = timeit(z);
        % Calculating flops.
        flops(k,l) = 2 * m(k) * n(l);
        % Calculating MFlop/s for each function.
        ij_mflops(k,l) = (flops(k,l)/rt_mv_ij(k,l)) * 10e-6;
        ji_mflops(k,l) = (flops(k,l)/rt_mv_ji(k,l)) * 10e-6;
        ab_mflops(k,l) = (flops(k,l)/rt_mv_ab(k,l)) * 10e-6;
        % Printing the requested results.
        fprintf('--------------------------------------------------\n')
        fprintf('Computing Matrix-Vector Multiplication c = A*b\n')
        fprintf('--------------------------------------------------\n')
        fprintf('>>> Number of matrix rows m: %i\n',m(k))
        fprintf('>>> Number of matrix columns n: %i\n',n(l))
        fprintf('>>> Number of flops: %i\n\n',flops(k,l))
        fprintf('Method\t\t   CPU (sec)    \t\tMFlop/s\n')
        fprintf('------\t\t-------------- \t\t--------\n')
        fprintf('mv_ij \t\t %d  \t\t %i\n',...
                    rt_mv_ij(k,l),round(ij_mflops(k,l)))
        fprintf('mv_ji \t\t %d  \t\t %i\n',...
                    rt_mv_ji(k,l),round(ji_mflops(k,l)))
        fprintf('mv_ab \t\t %d  \t\t %i\n',...
                    rt_mv_ab(k,l),round(ab_mflops(k,l)))
    end
end
```

```
% Plotting figure for time.
figure; % Main figure.
subplot(2,2,1); % Plot for m=2^4.
plot(n,rt_mv_ij(1,:),'b*-',...
      n,rt_mv_ji(1,:),'gx-',...
      n,rt_mv_ab(1,:),'r+-',...
      'LineWidth',2);
title('Time for m=2^4','fontWeight','bold');
xlabel('n'); ylabel('sec');
legend('mv\_ij','mv\_ji','mv\_ab',...
        'Location','NorthWest');
grid on; axis tight;
subplot(2,2,2); % Plot for m=2^6.
plot(n,rt_mv_ij(2,:),'b*-',...
      n,rt_mv_ji(2,:),'gx-',...
      n,rt_mv_ab(2,:),'r+-',...
      'LineWidth',2);
title('Time for m=2^6','fontWeight','bold');
xlabel('n'); ylabel('sec');
legend('mv\_ij','mv\_ji','mv\_ab',...
        'Location','NorthWest');
grid on; axis tight;
subplot(2,2,3); % Plot for m=2^8.
plot(n,rt_mv_ij(3,:),'b*-',...
      n,rt_mv_ji(3,:),'gx-',...
      n,rt_mv_ab(3,:),'r+-',...
      'LineWidth',2);
title('Time for m=2^8','fontWeight','bold');
xlabel('n'); ylabel('sec');
legend('mv\_ij','mv\_ji','mv\_ab',...
        'Location','NorthWest');
grid on; axis tight;
subplot(2,2,4); % Plot for m=2^10.
plot(n,rt_mv_ij(4,:),'b*-',...
      n,rt_mv_ji(4,:),'gx-',...
      n,rt_mv_ab(4,:),'r+-',...
      'LineWidth',2);
title('Time for m=2^1^0','fontWeight','bold');
xlabel('n'); ylabel('sec');
legend('mv\_ij','mv\_ji','mv\_ab',...
        'Location','NorthWest');
grid on; axis tight;

% Plotting figure for MFlop/s.
figure; % Main figure.
subplot(2,2,1); % Plot for m=2^4.
plot(n,ij_mflops(1,:),'b*-',...
      n,ji_mflops(1,:),'gx-',...
      n,ab_mflops(1,:),'r+-',...
      'LineWidth',2);
title('MFlop/s for m=2^4','fontWeight','bold');
xlabel('n'); ylabel('MFlop/s');
legend('mv\_ij','mv\_ji','mv\_ab',...
        'Location','NorthWest');
grid on; axis tight;
subplot(2,2,2); % Plot for m=2^6.
plot(n,ij_mflops(2,:),'b*-',...
      n,ji_mflops(2,:),'gx-',...
      n,ab_mflops(2,:),'r+-',...
      'LineWidth',2);
title('MFlop/s for m=2^6','fontWeight','bold');
```

```
xlabel('n'); ylabel('MFlop/s');
legend('mv\_ij','mv\_ji','mv\_ab',...
       'Location','NorthWest');
grid on; axis tight;
subplot(2,2,3) % Plot for m=2^8.
plot(n,ij_mflops(3,:),'b*-',...
     n,ji_mflops(3,:),'gx-',...
     n,ab_mflops(3,:),'r+-',...
     'LineWidth',2);
title('MFlop/s for m=2^8','fontWeight','bold');
xlabel('n'); ylabel('MFlop/s');
legend('mv\_ij','mv\_ji','mv\_ab',...
       'Location','NorthWest');
grid on; axis tight;
subplot(2,2,4) % Plot for m=2^10.
plot(n,ij_mflops(4,:),'b*-',...
     n,ji_mflops(4,:),'gx-',...
     n,ab_mflops(4,:),'r+-',...
     'LineWidth',2);
title('MFlop/s for m=2^1^0','fontWeight','bold');
xlabel('n'); ylabel('MFlop/s');
legend('mv\_ij','mv\_ji','mv\_ab',...
       'Location','Best');
grid on; axis tight;
```

files/3/iii/test_mv_iii.m





iv) α. Σαν ένα πρώτο σχόλιο, και όσον αφορά τις επιδόσεις των τριών υλοποιήσεων, θα μπορούσαμε να πούμε ότι σε κάθε περίπτωση η ενδογενής συνάρτηση της MATLAB είναι η γρηγορότερη από άποψη χρόνου, ανεξαρτήτως μεγέθους. Αυτό είναι λογικό, μιας και η συνάρτηση αυτή χρησιμοποιεί κάθε δυνατή βελτίωση για την εκτέλεση του πολλαπλασιασμού μητρώου επί διανύσματος.

Από άποψη επίδοσης, μια εικόνα μπορούμε να πάρουμε από την εξέταση των γραφημάτων που αφορούν τα MFlop/s. Θεωρητικά, όσο μεγαλύτερος είναι ο αριθμός τους, τόσο καλύτερη είναι η επίδοση της υλοποίησης, μιας και αυτό σημαίνει ότι εκτελεί περισσότερες πράξεις ανά μονάδα χρόνου. Και σε αυτόν τον τομέα βλέπουμε ότι η ενδογενής συνάρτηση έχει τις καλύτερες τιμές, ανεξαρτήτως μεγέθους.

Σε δύο περιπτώσεις αξίζει ένας μικρός σχολιασμός, μιας και στο γράφημα για $m = 2^6$ δείχνει μια μικρή πτώση στο διάστημα για n περίπου από 80 μέχρι 256, ενώ στο γράφημα για $m = 2^1 0$ επέρχεται πτώση του αριθμού από το σημείο όπου $n = 256$ και έπειτα. Αυτό μπορεί να οφείλεται στη θεώρησή μας ότι χρειάζεται τον ίδιο αριθμό πράξεων με τις άλλες δύο υλοποιήσεις, ενώ στην πραγματικότητα μπορεί να χρειάζεται λιγότερες, μιας και ο χρόνος εκτέλεσής της για το μεγαλύτερο μέγεθος της εκφώνησης είναι αρκετά μικρότερος από των άλλων δύο. Επίσης, αυτή η απότομη αύξηση στην αρχή μπορεί να οφείλεται και στην καλύτερη χρήση της μνήμης cache από την ενδογενή.

Τέλος, παρατηρούμε ότι για τα δύο πρώτα σχήματα, για $m = 2^4$ και $m = 2^6$ η επίδοση της υλοποίησης mv_ij εναλλάσσεται με την επίδοση της mv_ji ως προς το ποια είναι καλύτερη. Στην πρώτη περίπτωση παρατηρούμε ότι στην αρχή καλύτερη επίδοση έχει για λίγο η mv_ji αλλά αμέσως μετά πέφτει και γίνεται καλύτερη αυτή της mv_ij. Στην δεύτερη περίπτωση συμβαίνει το ίδιο αλλά σε διαφορετικό σημείο, περίπου εκεί όπου το μέγεθος n είναι 900. Αυτό σχετίζεται με τη μεταβολή στο χρόνο εκτέλεσής τους που εξετάζεται στη συνέχεια, καθώς ο αριθμός των πράξεων είναι και για τις δύο ίδιος.

β. Σαν ένα δεύτερο σχόλιο, όσον αφορά τα διάφορα μεγέθη του προβλήματος, θα λέγαμε ότι για μέγεθος $m \geq 2^8$ ισχύει ότι ο πολλαπλασιασμός ως προς γραμμές είναι πιο αργός από τον πολλαπλασιασμό ως προς στήλες, ενώ και οι δύο είναι πιο αργοί από τον ενδογενή πολλαπλασιασμό.

Τώρα, όσον αφορά τα αποτελέσματα για $2^4 \leq m \leq 2^6$. Παρατηρούμε, αντίστοιχα όπως και στα σχήματα για τα MFlop/s, ότι στο πρώτο γράφημα ο πολλαπλασιασμός mv_ij είναι πιο αργός από τον mv_ji μέχρι περίπου το σημείο όπου $n = 150$, και μετά από αυτό γίνεται γρηγορότερος, ενώ ο χρόνος του δεύτερου αυξάνεται με μεγάλο ρυθμό. Αντίστοιχα, στο δεύτερο γράφημα αυτό συμβαίνει περίπου από το σημείο όπου $n = 870$. Θα μπορούσαμε, λοιπόν, να πούμε ότι ο πολλαπλασιασμός mv_ij είναι ιδανικότερος όταν η διάσταση του μητρώου μας είναι μικρότερη, ενώ όταν αυξάνεται ο mv_ji είναι προτιμότερος. Στην ουσία, ο πρώτος τρόπος βασίζεται στον υπολογισμό του εσωτερικού γινομένου, ενώ ο δεύτερος στο γινόμενο ενός διανύσματος με έναν βαθμωτό (_AXPY). Η επιλογή και τα αποτελέσματα σίγουρα σχετίζονται και με τη χρήση της cache, όπως αναφέρθηκε και παραπάνω.

Όπως και να 'χει, θα λέγαμε ότι σε κάθε περίπτωση *η χρήση της ενδογενούς συνάρτησης είναι μονόδρομος*, μιας και αυτό που μας ενδιαφέρει κυρίως είναι ο χρόνος εκτέλεσης όλων των πραξεων, δεδομένης της ίσης αξιοπιστίας των αποτελεσμάτων μεταξύ των διαφορετικών υλοποιήσεων.

ΕΡΩΤΗΜΑ 4 - Εμπειρική Εκτίμηση Επίδοσης

i) Τα αποτελέσματα που πήραμε από την εκτέλεση της πράξης για τα μητρώα που υποδεικνύονται στην εκφώνηση είναι τα ακόλουθα, πάντα με χρήση της συνάρτησης timeit:

	n=50	**n=100**	**n=200**	**n=400**	**n=800**
sec ($\times 10^{-3}$)	0.0064	0.0149	0.0210	0.0357	0.2983

Το κομμάτι του κώδικα που αφορά την υλοποίηση αυτού του ερωτήματος είναι:

```
% Several matrix sizes.
n = [50;100;200;400;800];
%Initialization of rt_ab to store results.
rt_ab = zeros(5,1);
for i=1:5 % Loop to calculate execution time.
    % Random matrix and vector.
    A = randn(n(i),n(i));
    b = randn(n(i),1);
    % Calculating execution time with timeit.
    x = @() mtimes(A,b);
    rt_ab(i) = timeit(x);
end
```

files/4/ii/script_1_4_ii.m(1)

ii) Στο ερώτημα αυτό κάναμε χρήση της polyfit για να υπολογίσουμε τους συντελεστές των πολυωνύμων, καθώς και της polyval για να υπολογίσουμε τα αποτελέσματα με τη χρήση αυτών των συντελεστών, ώστε να μπορέσουμε να τα συγκρίνουμε με τις αρχικές τιμές. Ο κώδικας που αφορά αυτό το υποερώτημα είναι ο εξής:

```
% Calculating several polynomials.
P1 = polyfit(n,rt_ab,1);
r1 = polyval(P1,n);
P2 = polyfit(n,rt_ab,2);
r2 = polyval(P2,n);
P3 = polyfit(n,rt_ab,3);
r3 = polyval(P3,n);
P4 = polyfit(n,rt_ab,4);
r4 = polyval(P4,n);
```

files/4/ii/script_1_4_ii.m(2)

Έτσι, έχουμε τα εξής πολυώνυμα, τάξεως από 1 έως 4:

- $1^{ης}$ τάξης: $(0.0038n - 0.4330) \times 10^{-4}$
- $2^{ης}$ τάξης: $(0.0000n^2 - 0.0024n + 0.2702) \times 10^{-4}$
- $3^{ης}$ τάξης: $(0.0000n^3 - 0.0001n^2 + 0.0270n - 0.4404) \times 10^{-5}$
- $4^{ης}$ τάξης: $(-0.0000n^4 + 0.0000n^3 - 0.0002n^2 + 0.0373n - 0.8352) \times 10^{-5}$

Αξίζει να σημειώσουμε ότι οι συντελεστές που εμφανίζονται ως μηδενικοί δεν είναι ίσοι μεταξύ τους, μιας και η φαινομενική τους ισότητα οφείλεται στην παρουσίαση των αποτελεσμάτων από τη MATLAB. Έτσι, μπορεί π.χ. ο συντελεστής στο τέταρτο πολυώνυμο να φαίνεται ως μηδενικός αλλά να έχει πρόσημο αρνητικό, που σημαίνει ότι δεν είναι στην ουσία ίσος με το μηδέν. Ή μπορεί να φαίνεται απλά ως μηδενικός, αλλά υπάρχει πιθανότητα να μην είναι, απλά να είναι πολύ μικρός για τη σωστή αναπαράστασή του. Στα πλαίσια της άσκησης επιλέχτηκε να παρουσιαστούν έτσι, μιας και αυτό δεν επηρεάζει ιδιαίτερα το ζητούμενο συμπέρασμα.

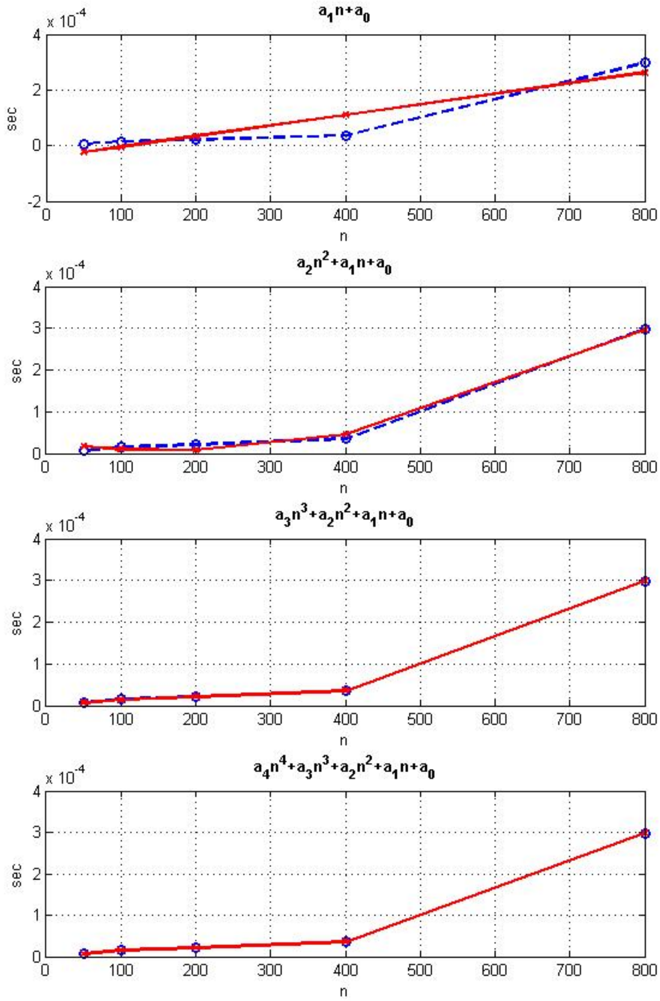
iii) Για τα πολυώνυμα αυτά, τώρα, και με τα αρχικά μας δεδομένα, υπολογίζουμε τους εξής χρόνους:

	n=50	**n=100**	**n=200**	**n=400**	**n=800**
1^{ης} ($\times 10^{-3}$)	-0.0242	-0.0051	0.0332	0.1097	0.2626
2^{ης} ($\times 10^{-3}$)	0.0167	0.0099	0.0073	0.0457	0.2966
3^{ης} ($\times 10^{-3}$)	0.0068	0.0142	0.0214	0.0356	0.2983
4^{ης} ($\times 10^{-3}$)	0.0064	0.0149	0.0210	0.0357	0.2983

Παρατηρούμε ότι όσο ανεβαίνει η τάξη του πολυωνύμου, στα συγκεκριμένα πλαίσια πάντα, η ακρίβεια των εκτιμήσεων γίνεται μεγαλύτερη, δηλαδή «πλησιάζει» όλο και περισσότερο τις αρχικές μας τιμές. Σε μια πρώτη ανάγνωση, λοιπόν, θα λέγαμε ότι το πολυώνυμο 4^{ης} τάξης θα μπορούσε να χρησιμοποιηθεί ως μαθηματικό μοντέλο για τον χρόνο υπολογισμού της πράξης του πολλαπλασιασμού μητρώου επί διάνυσμα. Αν οπτικοποιήσουμε και τα παραπάνω αποτελέσματα, χρησιμοποιώντας τον ακόλουθο κώδικα, έχουμε τις εξής γραφικές παραστάσεις, συναρτήσει και της αρχικής:

```
% Plotting graphs for each polynomial.
figure;
subplot(4,1,1); % Plot for first class.
plot(n,rt_ab,'bo--',n,r1,'rx-','LineWidth',2);
title('a_{1}n+a_{0}','fontWeight','bold');
xlabel('n'); ylabel('sec'); grid on;
subplot(4,1,2); % Plot for second class.
plot(n,rt_ab,'bo--',n,r2,'rx-','LineWidth',2);
title('a_{2}n^{2}+a_{1}n+a_{0}','fontWeight','bold');
xlabel('n'); ylabel('sec'); grid on;
subplot(4,1,3); % Plot for third class.
plot(n,rt_ab,'bo--',n,r3,'rx-','LineWidth',2);
title('a_{3}n^{3}+a_{2}n^{2}+a_{1}n+a_{0}',...
      'fontWeight','bold');
xlabel('n'); ylabel('sec'); grid on;
subplot(4,1,4); % Plot for fourth class.
plot(n,rt_ab,'bo--',n,r4,'rx-','LineWidth',2);
title('a_{4}n^{4}+a_{3}n^{3}+a_{2}n^{2}+a_{1}n+a_{0}',...
      'fontWeight','bold');
xlabel('n'); ylabel('sec'); grid on;
```

files/4/ii/script_1_4_ii.m(3)



Βλέπουμε κι εδώ ότι οι γραφικές παραστάσεις του τρίτου και τέταρτου βαθμού πολυωνύμου προσεγγίζουν καλύτερα την αρχική γραφική παράσταση των μετρήσεών μας.

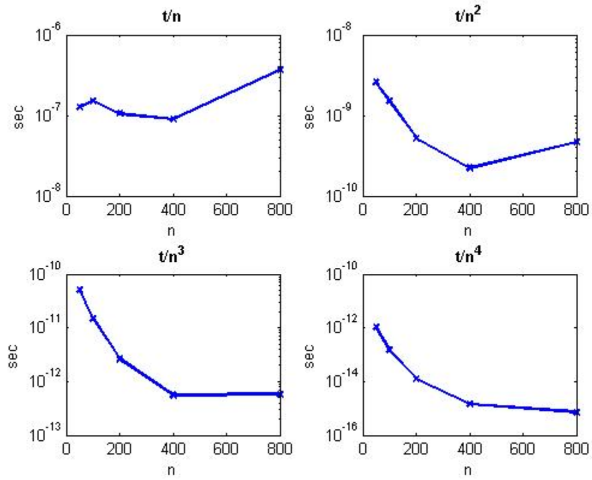
Παρόλ' αυτά, θα ήταν λάθος αν καταλήγαμε στο συμπέρασμα ότι το πολυώνυμο τετάρτου βαθμού είναι αυτό που μπορεί να χρησιμοποιηθεί αξιόπιστα ως μαθηματικό μοντέλο για τον χρόνο υπολογισμου που ψάχνουμε. Κι αυτό γιατί, κατά την εκτέλεση του παρουσιάζεται η εξής προειδοποίηση:

```
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 script_1_4_ii at 21
```

Πηγαίνοντας, λοιπόν, στη γραμμή 21 του κώδικά μας, παρατηρούμε ότι εκεί γίνεται ο υπολογισμός των συντελεστών του τετάρτης τάξεως πολυωνύμου. Η προειδοποίηση λέει ότι το πολυώνυμο είναι κακώς ορισμένο και μας προτείνει 2-3 τρόπους για να διορθώσουμε την κατάσταση. Ο πρώτος τρόπος αφορά στην προσθήκη περισσότερων σημείων με διακριτές τιμές ως προς το x. Ο τρίτος τρόπος αφορά στην κανονικοποίηση και κεντροποίηση όπως περιγράφεται στην επεξήγηση της χρησιμοποιούμενης εντολής. Έτσι, αν ακολουθήσουμε τον εξής τρόπο εκτέλεσης, η προειδοποίηση παύει να εμφανίζεται:

[P4,S,MU] = polyfit(n,rt_ab,4);

Όμως, στην ουσία, το πρόβλημά μας παραμένει. Η τελευταία πρόταση, η δεύτερη στη σειρά, μας λέει να δοκιμάσουμε ένα πολυώνυμο μικρότερου βαθμού για να γίνει ορθότερο ο υπολογισμός. Συνεπώς, κρατάμε ότι το πολυώνυμο τρίτης τάξης που έχουμε υπολογίσει πιθανόν να είναι πιο αξιόπιστο. Αυτή την υπόθεση μπορούμε να την επικυρώσουμε αν χρησιμοποιήσουμε τη μέθοδο που υποδείχθηκε και κατά το εργαστήριο, διαιρώντας το χρόνο με το μέγεθος του μητρώου, υψωμένου κάθε φορά στην αντίστοιχη δύναμη. Αν οπτικοποιήσουμε τα αποτελέσματα αυτής της διαδικασίας, έχουμε τις εξής γραφικές παραστάσεις:



Ο κώδικας που αφορά τις παραπάνω γραφικές παραστάσεις είναι ο ακόλουθος:

```
45 figure; % Plotting graphs to find complexity.
46 subplot(2,2,1); % Plot for first class.
47 semilogy(n,(rt_ab./n),'x-','LineWidth',2);
48 title('t/n','fontWeight','bold');
49 xlabel('n'); ylabel('sec');
50 subplot(2,2,2); % Plot for second class.
51 semilogy(n,(rt_ab./(n.^2)),'x-','LineWidth',2);
52 title('t/n^2','fontWeight','bold');
53 xlabel('n'); ylabel('sec');
54 subplot(2,2,3); % Plot for third class.
55 semilogy(n,(rt_ab./(n.^3)),'x-','LineWidth',2);
56 title('t/n^3','fontWeight','bold');
57 xlabel('n'); ylabel('sec');
58 subplot(2,2,4); % Plot for fourth class.
59 semilogy(n,(rt_ab./(n.^4)),'x-','LineWidth',2);
```

```
title('t/n^4','fontWeight','bold');
xlabel('n'); ylabel('sec');
```

files/4/ii/script_1_4_ii.m(4)

Αν τις εξετάσουμε, λοιπόν, παρατηρούμε ότι ο βαθμός του πολυωνύμου στον οποίο φαίνεται ότι σταθεροποιείται η γραφική παράσταση είναι ο τρίτος, δηλαδή ένα πολυώνυμο τρίτης τάξεως. Έτσι, επιβεβαιώνεται η προηγούμενή μας υπόθεση, ότι *το πολυώνυμο τρίτου βαθμού είναι το πιο κατάλληλο ως μαθηματικό μοντέλο* για τον χρόνο υπολογισμού του πολλαπλασιασμού μας.