Επιστημονικός Υπολογισμός
Εργαστηριακή Άσκηση

Χειμερινό Εξάμηνο 2023 – 2024

4 Φεβρουαρίου 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Τμήμα Μηχανικών Η / Υ και Πληροφορικής
Πολυτεχνική Σχολή

Ονοματεπώνυμο: Μηλτιάδης Μαντές
Α.Μ.: 1084661
E – mail: up1084661@ac.upatras.gr
Εξάμηνο: 7ο

Διδάσκων: Ευστράτιος Γαλλόπουλος
Τομέας Εφαρμογών και Θεμελιώσεων της Επιστήμης των Υπολογιστών

Περιεχόμενα

https://github.com/miltiadiss/Scientific-Computing

0 Στοιχεία Υπολογιστικού Συστήματος

ΈΝΑΡΞΗ/ΛΗΞΗ ΕΡΓΑΣΙΑΣ	28/01/2024 – 4/02/2024
MODEL	Προσωπικό laptop: DELL Inspiron 15 3528
O/S	Windows 11 Pro 23H2
PROCESSOR NAME	Intel(R) Core(TM) i5-8350U CPU @ 1.70GHz
PROCESSOR SPEED	1.90 GHz
NUMBER OF PROCESSORS	1
TOTAL # CORES	4
TOTAL # THEADS	8
FMA INSTRUCTION	yes
L1 CACHE	128 KB
L2 CACHE	1 MB
L3 CACHE	6 MB
GFLOPS/S	185.7
MEMORY	24 GB
MEMORY BANDWIDTH	DDR4-2400 (1200 MHz)
MATLAB VERSION	23.2.0.2459199 (R2023b) Update 5

Computer Type	LU	FFT	ODE	Sparse	2-D	3-D
Mac mini, Apple M2 @ 3.50 GHz	0.5551	0.1516	0.0767	0.2470	0.1682	0.1430
Windows 11, Intel Core i9-12900 @ 2.4 GHz	0.2516	0.1523	0.0881	0.4521	0.1890	0.2359
Windows 11, AMD Ryzen Threadripper(TM) 3970x @ 3.7 GHz	0.1945	0.1662	0.1723	1.2129	0.1981	0.1274
Windows 11, Intel Core i7-1185G7 @ 3.00 GHz	0.6690	0.3013	0.1433	0.3440	0.2774	0.1461
iMac, macOS 13.2.1, Intel Core i9 @ 3.6 GHz	0.3347	0.2679	0.1336	0.2840	0.6960	0.3816
Debian 11(R), AMD Ryzen Threadripper 2950x @ 3.50 GHz	0.3384	0.2465	0.1597	1.2545	0.2516	0.1971
Windows 11, AMD Ryzen(TM) 5 Pro 6650U @ 2.9 GHz	0.6419	0.3626	0.1273	0.5449	0.3753	0.2346
This machine	0.9610	0.5254	0.2453	0.4877	0.4938	0.4099
MacBook Pro, macOS 11.7.2, Intel(R) Core(TM) i7 @ 2.9 GHz	0.8710	0.5960	0.2006	0.5297	1.7433	0.7368

1 Χρονομετρήσεις

```
% Προεργασία
n = 100:100:2000;
exec_times = zeros(size(n)); % Διάνυσμα όπου αποθηκεύονται οι πραγματικοί χρόνοι εκτέλεσης της εντολής chol(A)

for i = 1:length(n) % Δημιουργία τυχαίων ΣΘΟ μητρώων τάξης n
    dimension = n(i);
    A = randn(dimension, dimension); % Επιλογή τυχαίων τιμών για το τετραγωνικό μητρώο A
    A_symmetric = (A + A') / 2; % Μετατρέπουμε το μητρώο A σε συμμετρικό
    A_symmetric_positive = A_symmetric' * A_symmetric; % Μετατρέπουμε το συμμετρικό μητρώο A σε θετικά ορισμένο
    f = @() chol(A_symmetric_positive); % Χρονομέτρηση της εκτέλεσης της εντολής chol(A) για κάθε n
    exec_times(i) = timeit(f);
end
disp('Πραγματικοί χρόνοι εκτέλεσης για κάθε διάσταση n:')
disp(num2str(exec_times))
```

Αρχικά, ορίζουμε το εύρος διαστάσεων n του τυχαίου Σ.Θ.Ο μητρώου A^{nxn}, δηλαδή το διάστημα [100, 2000] με βήμα 100. Έπειτα, κατασκευάζουμε το διάνυσμα exec_times όπου αποθηκεύουμε τους πραγματικούς χρόνους εκτέλεσης της εντολής **X = chol(A)** για τις διάφορες τιμές της διάστασης n του μητρώου A^{nxn}. Στη συνέχεια, κατασκευάζουμε όλα τα διαφορετικά τυχαία μητρώα A^{nxn} μέσω της **randn( )**, η οποία παράγει τυχαίες τιμές από μια Gaussian κατανομή με μέση τιμή μ = 0 και διασπορά $\sigma^2 = 1$. Κάθε τυχαίο μητρώο το μετατρέπουμε σε Σ.Θ.Ο και καλούμε πάνω σε αυτό τη συνάρτηση **chol( )**. Με μια ανώνυμη συνάρτηση **@( )** ονόματος **f** θα επιχειρήσουμε να χρονομετρήσουμε το χρόνο εκτέλεσης της συνάρτησης **chol( )** περνώντας τη σαν όρισμα στη συνάρτηση **timeit( )**. Κάθε τιμή μιας χρονομέτρησης αποθηκεύεται τελικά στο αρχικό διάνυσμα exec_times.

(1.1) Στο αρχείο **ask1.m** υλοποιούμε τον εξής κώδικα:

```
% Ερώτημα 1
p3 = polyfit(n, exec_times, 3); % Υπολογισμός συντελεστών a3...a0 του πολυωνύμου με σκοπό την ελαχιστοποίηση σφάλματος
Tchol3 = @(n) p3(1)*n.^3 + p3(2)*n.^2 + p3(3)*n + p3(4); % Δημιουργία της κυβικής συνάρτησης Tchol(n)
disp('Κυβικό Πολυώνυμο:') % Εκτύπωση του πολυωνύμου
fprintf('Tchol(n) = %.4fn^3 + %.4fn^2 + %.4fn + %.4f\n', p3(1), p3(2), p3(3), p3(4));
```

Αφού έχουμε υπολογίσει τους χρόνους εκτέλεσης της ζητούμενης εντολής σε όλα τα παραχθέντα μητρώα A^{nxn} χρησιμοποιούμε τη συνάρτηση **polyfit( )** με ορίσματα το διάνυσμα exec_times και τη σταθερά 3, προκειμένου να υπολογίσουμε τους συντελεστές a_0, a_1, a_2 και a_3 του κυβικού πολυωνύμου $T_{chol}(n) = \mathbf{a_3}n^3 + \mathbf{a_2}n^2 + \mathbf{a_1}n + \mathbf{a_0}$. Οι συντελεστές αυτοί είναι τέτοιοι ώστε να ελαχιστοποιούν το μέσο τετραγωνικό σφάλμα πρόβλεψης ανάμεσα στους πραγματικούς και προβλεπόμενους χρόνους εκτέλεσης. Οι συντελεστές αποθηκεύονται στο διάνυσμα p3, το οποίο το ονομάζουμε με αυτό το τρόπο για να συμβολίζει τους συντελεστές του 3ης τάξης πολυωνύμου. Έτσι, η τελική κυβική συνάρτηση θα δίνεται από τον παραπάνω τύπο με την εξής αντιστοιχία:
a_3 → p3(1)
a_2 → p3(2)
a_1 → p3(3)
a_0 → p3(4),
όπου p3(1) ο πρώτος συντελεστής που επιστρέφει η **polyfit( )**, p3(2) ο δεύτερος συντελεστής κ. ο. κ.

(1.2)

```
% Ερώτημα 2
predicted_times3 = arrayfun(Tchol3, n); % Υπολογισμός των προβλεπόμενων χρόνων εκτέλεσης
disp('Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:'); % Εκτύπωση των προβλεπόμενων χρόνων εκτέλεσης
disp(num2str(predicted_times3));
```

Μέσω της συνάρτησης **arryfun( )** και με ορίσματα το πολυώνυμο $T_{chol}(n)$ 3ης τάξης και την αντίστοιχη διάσταση n του μητρώου A^{nxn} εφαρμόζουμε τη πολυωνυμική συνάρτηση T_{chol} σε κάθε διαφορετική διάσταση και αποθηκεύουμε τη πρόβλεψη εκτέλεσης που προκύπτει στο διάνυσμα predicted_times3. Το ονομάζουμε και αυτό με το συγκεκριμένο τρόπο για να δηλώσουμε ότι οι συγκεκριμένες προβλέψεις πραγματοποιούνται από ένα κυβικό πολυώνυμο.

Σημείωση:
Επαναλαμβάνουμε τα συγκεκριμένα ερωτήματα και για εύρος διαστάσεων n = 150:100:1550 όπως μας ζητείται, τροποποιώντας μόνο τη πρώτη γραμμή του κώδικά μας κατά την Προεργασία. Έτσι, θα προκύψουν και οι αντίστοιχοι συντελεστές και οι προβλέψεις για τη συγκεκριμένη περίπτωση.

Εκτέλεση Ερωτημάτων 1, 2:

- Για τη πρώτη περίπτωση λαμβάνουμε το εξής πολυώνυμο και τους 20 διαφορετικούς προβλεπόμενους χρόνους εκτέλεσης στο τερματικό μας παράθυρο:

```
Πραγματικοί χρόνοι εκτέλεσης για κάθε διάσταση n:
3.6058e-05  0.00012469  0.00020714  0.00063697  0.0010517  0.0022742  0.0038561  0.0043774  0.0065598
0.0075515  0.0089922  0.0096987  0.016148  0.019574  0.021814  0.03011  0.034965  0.034926  0.046575
0.050179
Κυβικό Πολυώνυμο:
Tchol(n) = 0.00000000n^3 + 0.00000000n^2 + -0.00000106n + 0.00021334
Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:
0.00014883  0.00018637  0.00035489  0.00068335  0.0012007  0.0019358  0.0029177  0.0041753  0.0057375
0.0076333  0.0098916  0.012541  0.015612  0.019131  0.023129  0.027634  0.032675  0.038282  0.044482
0.051305
```

- Για τη δεύτερη περίπτωση λαμβάνουμε το εξής πολυώνυμο και τους 20 διαφορετικούς προβλεπόμενους χρόνους εκτέλεσης στο τερματικό μας παράθυρο:

```
Πραγματικοί χρόνοι εκτέλεσης για κάθε διάσταση n:
0.00015692  0.00021268  0.00044522  0.0010642  0.0015064  0.0026614  0.0037571  0.0058725  0.0048283
0.007201  0.012818  0.014772  0.015281  0.024379  0.020828
Κυβικό Πολυώνυμο:
Tchol(n) = -0.00000000n^3 + 0.00000002n^2 + -0.00000952n + 0.00156637
Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:
0.00052089  0.00024087  0.00028555  0.00064735  0.0013187  0.0022919  0.0035594  0.0051137  0.0069471
0.009052  0.011421  0.014046  0.01692  0.020035  0.023384
```

(1.3)

```
% Ερώτημα 3
figure; % Γραφική παράσταση των χρονομετρήσεων και των προβλέψεών τους
hold on;
plot(n, exec_times, 'o', 'DisplayName', 'Real Execution Times'); % Χάραξη χρονομετρήσεων με 'o'
plot(n, predicted_times3, 'b', 'DisplayName', 'Predicted Execution Times - 3rd degree'); % Χάραξη
προβλέψεων με συνεχή μπλε γραμμή
title('Comparison of Real and Predicted Execution Times'); % Τίτλος γραφήματος
xlabel('Matrix Size (n)'); % Ονοματοδοσία άξονα x
ylabel('Execution Time (sec)'); % Ονοματοδοσία άξονα y
legend; % Υπόμνημα
grid on;
hold off;
```

Τώρα χαράσσουμε τις γραφικές παραστάσεις των πραγματικών τιμών χρόνου εκτέλεσης και των προβλέψεων όπως ακριβώς μας ζητείται. Με τη συνάρτηση **plot( )** τοποθετούμε στον άξονα x τις διάφορες τιμές του n, στον άξονα y τις διάφορες τιμές των exec_times και predicted_times3 και σχεδιάζουμε τις γραφικές παραστάσεις. Δίνουμε και τα απαραίτητα ονόματα σε γράφημα και άξονες με τις συναρτήσεις **title( )**, **xlabel( )**, **ylabel( )** και τέλος για ευκρίνεια τοποθετούμε και ένα υπόμνημα με τη **legend**.

Σημείωση:
Για να συγκρίνουμε τις δύο περιπτώσεις και να συμπεράνουμε για ποιες διαστάσεις του μητρώου έχουμε καλύτερη προσέγγιση των πραγματικών τιμών χρόνων εκτέλεσης μπορούμε να υπολογίσουμε το Μέσο Τετραγωνικό Σφάλμα (MSE).

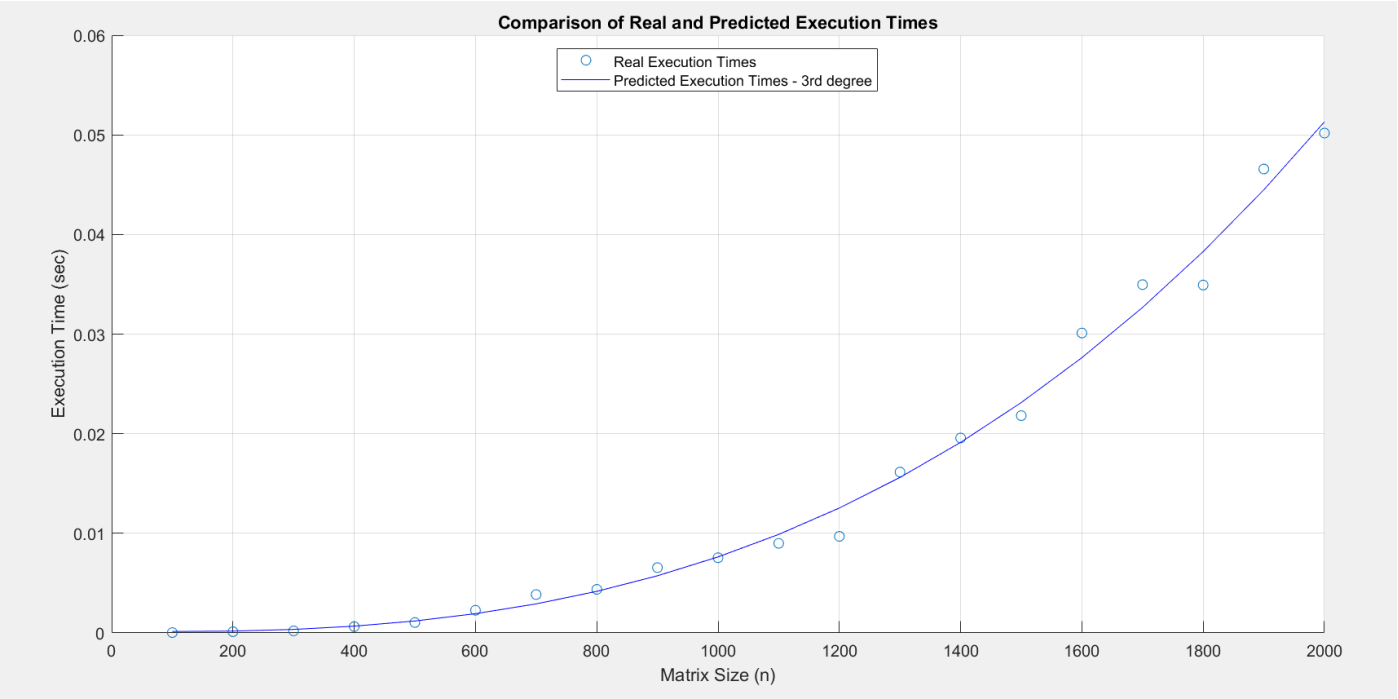
```
mse3 = mean((exec_times - predicted_times3).^2); % Υπολογισμός του Μέσου Τετραγωνικού Σφάλματος (MSE)
fprintf('Μέσο Τετραγωνικό Σφάλμα με κυβικό πολυώνυμο: %.4f\n', mse3);
```

Υπολογίζουμε μέσω της **mean( )** τη μέση τιμή του τετραγώνου της διαφοράς της πραγματικής τιμής και της πρόβλεψής της και αποθηκεύουμε το αποτέλεσμα στο mse3. Το ονομάζουμε έτσι για να γνωρίζουμε ότι το Μέσο Τετραγωνικό Σφάλμα αντιστοιχεί στο κυβικό πολυώνυμο.

Εκτέλεση Ερωτήματος 3:

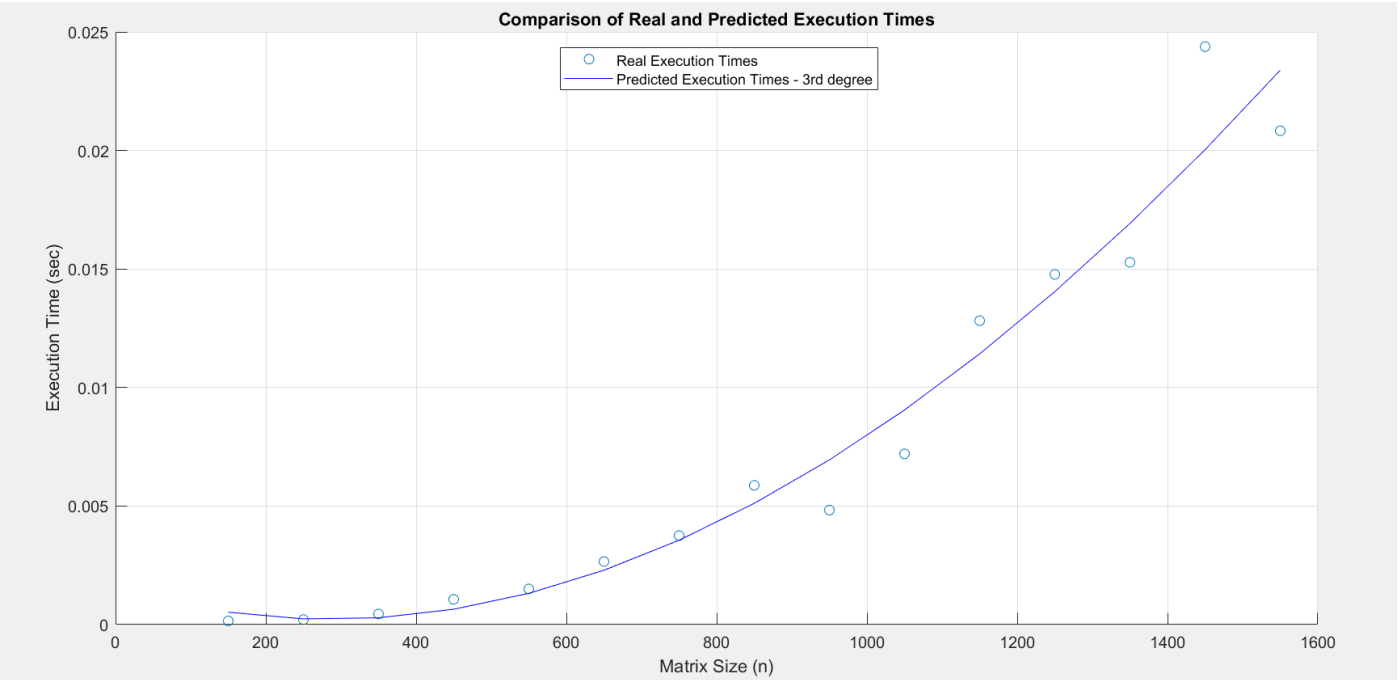
Εικόνα 1: Σύγκριση πραγματικών και πειραματικών χρόνων εκτέλεσης για n = 100:100:2000

```
Μέσο Τετραγωνικό Σφάλμα με κυβικό πολυώνυμο: 0.00000206
```



Εικόνα 2: Σύγκριση πραγματικών και πειραματικών χρόνων εκτέλεσης για n = 150:100:1550

```
Μέσο Τετραγωνικό Σφάλμα με κυβικό πολυώνυμο: 0.00000264
```



(1.4)

```
% Ερώτημα 4
p2 = polyfit(n, exec_times, 2);
Tchol2 = @(n) p2(1)*n.^2 + p2(2)*n + p2(3);
disp('Πολυώνυμο 2ου βαθμού:') % Εκτύπωση του πολυωνύμου
fprintf('Tchol(n) = %.4fn^2 + %.4fn + %.4f\n', p2(1), p2(2), p2(3));
predicted_times2 = arrayfun(Tchol2, n);
disp('Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:'); % Εκτύπωση των προβλεπόμενων χρόνων εκτέλεσης
disp(num2str(predicted_times2));

p4 = polyfit(n, exec_times, 4);
Tchol4 = @(n) p4(1)*n.^4 + p4(2)*n.^3 + p4(3)*n.^2 + p4(4)*n + p4(5);
disp('Πολυώνυμο 4ου βαθμού:') % Εκτύπωση του πολυωνύμου
fprintf('Tchol(n) = %.4fn^4 + %.4fn^3 + %.4fn^2 + %.4fn + %.4f\n', p4(1), p4(2), p4(3), p4(4), p4(5));
predicted_times4 = arrayfun(Tchol4, n);
disp('Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:'); % Εκτύπωση των προβλεπόμενων χρόνων εκτέλεσης
disp(num2str(predicted_times4));

figure; % Γραφική παράσταση των χρονομετρήσεων και των προβλέψεων για όλα τα πολυώνυμα
hold on;
plot(n, exec_times, 'o', 'DisplayName', 'Real Execution Times');
plot(n, predicted_times3, 'b', 'DisplayName', 'Predicted Execution Times - 3rd degree');
plot(n, predicted_times2, 'r', 'DisplayName', 'Predicted Execution Times - 2nd degree');
plot(n, predicted_times4, 'g', 'DisplayName', 'Predicted Execution Times - 4th degree');
title('Comparison of Real and Predicted Execution Times');
xlabel('Matrix Size (n)');
ylabel('Execution Time (sec)');
legend;
grid on;
hold off;

mse2 = mean((exec_times - predicted_times2).^2); % Υπολογισμός του Μέσου Τετραγωνικού Σφάλματος (MSE)
fprintf('Μέσο Τετραγωνικό Σφάλμα με τετραγωνικό πολυώνυμο: %.4f\n', mse2);
mse4 = mean((exec_times - predicted_times4).^2); % Υπολογισμός του Μέσου Τετραγωνικού Σφάλματος (MSE)
fprintf('Μέσο Τετραγωνικό Σφάλμα με πολυώνυμο 4ου βαθμού: %.4f\n', mse4);
```

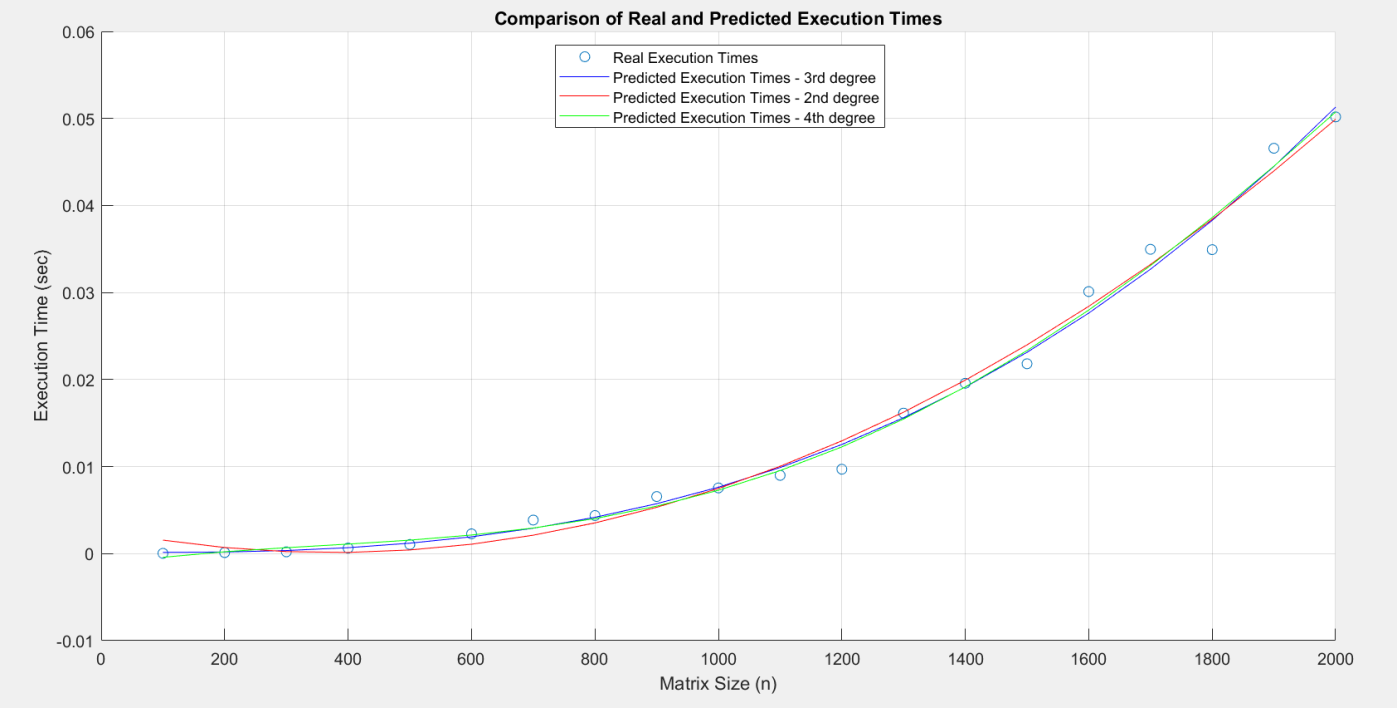
Ακολουθούμε την ίδια διαδικασία με τα Ερωτήματα 1, 2, 3 και υπολογίζουμε τα νέα πολυώνυμα 2ου και 4ου βαθμού, τους νέους χρόνους πρόβλεψης και τις νέες γραφικές τις οποίες τοποθετούμε μαζί με τη γραφική του κυβικού πολυωνύμου προκειμένου να έχουμε μια γρήγορη εικόνα της απόδοσης κάθε πολυωνύμου πριν τα συγκρίνουμε και τα τρία μέσω του MSE.

Εκτέλεση Ερωτήματος 4 (για n = 100:100:2000 και n = 150:100:1550) :

```
Πολυώνυμο 2ου βαθμού:
Tchol(n) = 0.00000002n^2 + -0.00001414n + 0.00277610
Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:
0.001551  0.00070297  0.00023189  0.0001378  0.0004207  0.0010806  0.0021175  0.0035314  0.0053222
0.0074901  0.010035  0.012957  0.016256  0.019931  0.023984  0.028414  0.033221  0.038405  0.043965
0.049903
Πολυώνυμο 4ου βαθμού:
Tchol(n) = -0.00000000n^4 + 0.00000000n^3 + -0.00000002n^2 + 0.00001116n + -0.00134921
Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:
-0.00042113  0.00021637  0.00068487  0.0010957  0.0015498  0.0021379  0.0029404  0.0040274  0.0054587
0.0072839  0.0095423  0.012263  0.015464  0.019154  0.023331  0.027983  0.033088  0.038612
0.044512  0.050735
```

Εικόνα 3: Σύγκριση πραγματικών και πειραματικών χρόνων εκτέλεσης για n = 100:100:2000

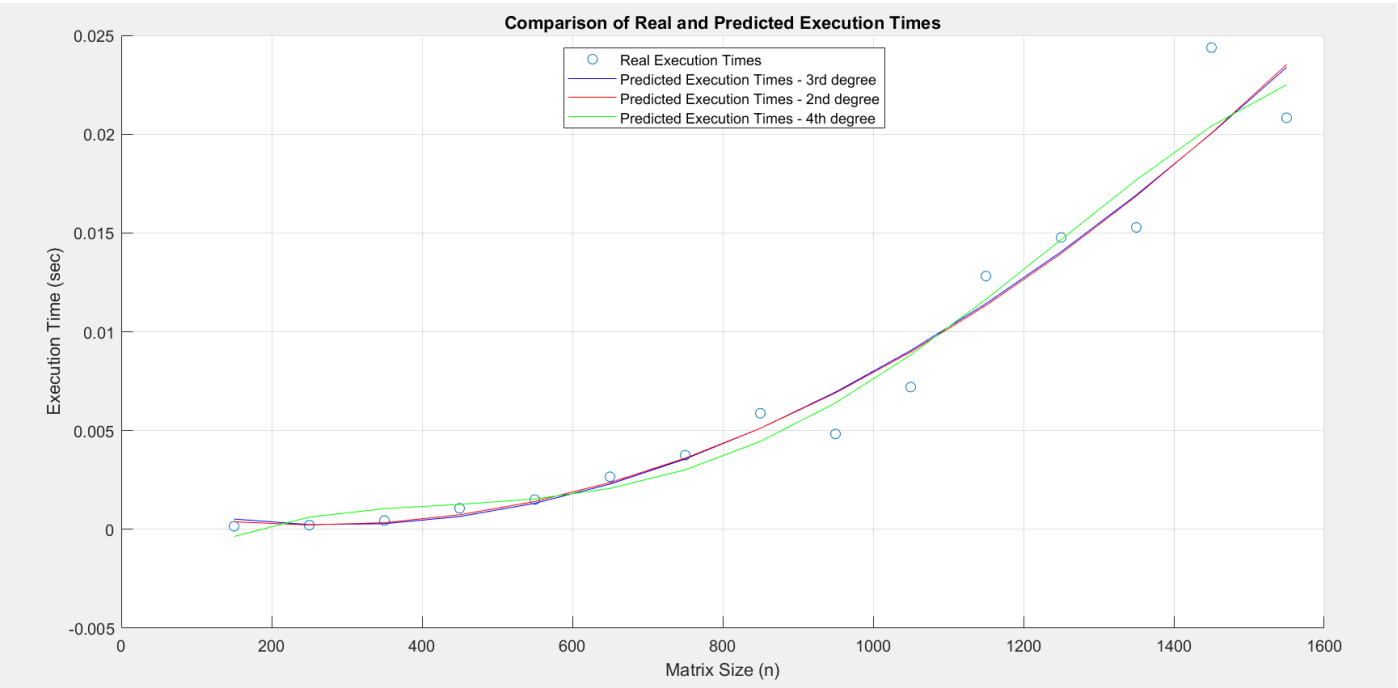
Μέσο Τετραγωνικό Σφάλμα με τετραγωνικό πολυώνυμο: 0.00000257
Μέσο Τετραγωνικό Σφάλμα με πολυώνυμο 4ου βαθμού: 0.00000196



Πολυώνυμο 2ου βαθμού:
Tchol(n) = 0.00000001n^2 + -0.00000720n + 0.00114825
Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:
0.00038263 0.00022112 0.00033873 0.00073547 0.0014113 0.0023663 0.0036004 0.0051137 0.006906 0.0089775 0.011328 0.013958 0.016867 0.020055 0.023522
Πολυώνυμο 4ου βαθμού:
Tchol(n) = -0.00000000n^4 + 0.00000000n^3 + -0.00000008n^2 + 0.00003281n + -0.00371751
Προβλεπόμενοι χρόνοι εκτέλεσης για κάθε διάσταση n:
-0.00035819 0.00061762 0.0010487 0.0012656 0.0015373 0.0020714 0.0030141 0.0044498 0.0064017 0.0088316 0.01164 0.014664 0.017683 0.020412 0.022505

Εικόνα 4: Σύγκριση πραγματικών και πειραματικών χρόνων εκτέλεσης για n = 150:100:1550

Μέσο Τετραγωνικό Σφάλμα με τετραγωνικό πολυώνυμο: 0.00000265
Μέσο Τετραγωνικό Σφάλμα με πολυώνυμο 4ου βαθμού: 0.00000231



Συμπεράσματα:

Όσο αυξάνεται η τάξη του πολυωνύμου $T_{chol}(n)$ τόσο περισσότερο μικραίνει το Ελάχιστο Τετραγωνικό Σφάλμα ανάμεσα στις πραγματικές τιμές και τις προσεγγίσεις. Αυτό είναι αναμενόμενο να συμβαίνει καθώς έχουμε περισσότερους συντελεστές ελαχιστοποίησης σφάλματος, οι οποίοι αν συνδυαστούν κάνουν μια ακόμα πιο ακριβή πρόβλεψη. Γενικά, παρατηρούμε ότι το MSE μειώνεται όσο αυξάνεται η τάξη του πολυωνύμου και όσο μεγαλώνει το εύρος διαστημάτων n. Πράγματι, όταν είναι n = 100:100:2000 έχουμε μικρότερο σφάλμα κατά την εκτέλεση σε σχέση με n = 150:100:1550.

2 Ειδικοί Επιλυτές και Αραιά Μητρώα

(2.1) Στο αρχείο **tridiagonal_1084661.m** υλοποιούμε τον εξής κώδικα:

```
function x = tridiagonal_1084661(A, B)

    A0 = A; % Ξεκινάμε διασπώντας το μητρώο A(0) σε D, L, U
    D = diag(diag(A0)); % Κατασκευάζουμε το διαγώνιο μητρώο D
    L = diag(diag(A0, -1),-1); % Κατασκευάζουμε το κάτω τριγωνικό μητρώο L
    U = diag(diag(A0, 1),1); % Κατασκευάζουμε το άνω τριγωνικό μητρώο U

    % Μετασχηματίζουμε τα D, L, U σε D(1), L(1), U(1)
    D1 = D - L * diag(1 ./ diag(D)) * U - U * diag(1 ./ diag(D)) * L;
    L1 = L * diag(1 ./ diag(D)) * L;
    U1 = U * diag(1 ./ diag(D)) * U;
    A1 = D1 - L1 - U1; % Μετασχηματίζουμε το μητρώο A(0) σε A(1)

    % Μετασχηματίζουμε τα διανύσματα b(0) του μητρώου B(0) σε b(1) στο
    % μητρώο B(1) οπότε ολοκληρώνεται το πρώτο βήμα του αλγορίθμου
    B0 = B;
    B1 = B0 + L * diag(1 ./ diag(D)) * B0 + U * diag(1 ./ diag(D)) * B0;

    % Αρχικοποιούμε τα μητρώα A(m) με τις πρώτες τιμές που θα χρησιμοποιήσουμε παρακάτω στον
    % επαναληπτικό αλγόριθμο
    Am = A1;
    Dm = D1;
    Lm = L1;
    Um = U1;
    disp(Am); % Εκτυπώνουμε το πρώτο τροποποιημένο μητρώο A(1) από τα συνολικά m που θα χρειαστούν

    for i = 1 : ceil(log2(size(A) / 4)) + 1 % Αν το μετασχηματισμένο μητρώο δεν έχει φτάσει στη κατάλληλη μορφή
συνεχίζουμε να το μετασχηματίζουμε άλλες i φορές μέχρι να φτάσουμε στο A(m)
        Dmplus1 = Dm - Lm * diag(1 ./ diag(Dm)) * Um - Um * diag(1 ./ diag(Dm)) * Lm; % Βρίσκουμε το D(m+1)
        Lmplus1 = Lm * diag(1 ./ diag(Dm)) * Lm; % Βρίσκουμε το L(m+1)
        Umplus1 = Um * diag(1 ./ diag(Dm)) * Um; % Βρίσκουμε το U(m+1)

        Amplus1 = Dmplus1 - Lmplus1 - Umplus1; % Μετασχηματίζουμε το μητρώο A(m) σε A(m+1)
        disp(num2str(Amplus1));

        Dm = Dmplus1; % Θέτουμε για την επόμενη επανάληψη σαν προηγούμενο μητρώο το D(m+1)
        Lm = Lmplus1; % Θέτουμε για την επόμενη επανάληψη σαν προηγούμενο μητρώο το L(m+1)
        Um = Umplus1; % Θέτουμε για την επόμενη επανάληψη σαν προηγούμενο μητρώο το U(m+1)

        if i == ceil(log2(size(A) / 4)) + 1 % Αν φτάσουμε στο τελευταίο βήμα ο αλγόριθμος τερματίζει και ξεκινάμε
να λύσουμε το σύστημα
            for j = 1:size(B,2)
                x(:, j) =  Amplus1 \ B1(:, j); % Το A(m+1) είναι διαγώνιο και λύνουμε το σύστημα A(m+1)x = B(1)
απευθείας
            end
        end
    end
end
```

Το πρώτο μητρώο με το οποίο ξεκινάμε είναι το αρχικό τριδιαγώνιο A^{nxn}, το οποίο ονομάζουμε $A^{(0)}$ όπως περιγράφεται και στο Σετ διαφανειών 11. Στη συνέχεια, διασπάμε το $A^{(0)}$ σε διαγώνιο μητρώο D^{nxn}, αυστηρά κάτω τριγωνικό L^{nxn} και αυστηρά άνω τριγωνικό U^{nxn}. Για το πρώτο, η εντολή **diag(A0)** δίνει ένα διάνυσμα με τα διαγώνια στοιχεία του $A^{(0)}$ και έπειτα αυτό το διάνυσμα το μετατρέπουμε στο τελικό μητρώο D^{nxn} πάλι μέσω της **diag( )**. Η εντολή **diag(diag(A0, -1),-1)** πάλι παίρνει τα διαγώνια στοιχεία και τα στοιχεία των υποδιαγωνίων του $A^{(0)}$ και τα τοποθετεί στο μητρώο L^{nxn}, κάνοντας όλα τα στοιχεία της υπερδιαγωνίου μηδέν. Όμοια προκύπτει και το μητρώο U^{nxn}. Το μητρώο D^{-1} δίνεται με την εντολή **diag(1 ./ diag(D))**, η οποία τοποθετεί σε ένα διαγώνιο μητρώο τα αντίστροφα διαγώνια στοιχεία του D^{nxn}. Έπειτα, υπολογίζουμε το πρώτο μετασχηματισμό $A^{(1)}$ ως

$D - L - U$, όπως περιγράφεται, μαζί με το μετασχηματισμό $B^{(1)}$ ως $B + L(D^{-1}\mathbf{B}) + U(D^{-1}B)$. Μετά από αυτό το βήμα και με βάση τη διάσταση του εκάστοτε μητρώου αποφασίζουμε αν θα χρειαστεί να εκτελέσουμε άλλα επαναληπτικά βήματα του αλγορίθμου. Αν ναι, τότε βρίσκουμε το μετασχηματισμό $D^{(m+1)}$, $L^{(m+1)}$ και $U^{(m+1)}$ με βάση τους μετασχηματισμούς $D^{(m)}$, $L^{(m)}$ και $U^{(m)}$ που υλοποιήσαμε στο προηγούμενο επαναληπτικό βήμα κάθε φορά μέχρι το τελικό μετασχηματισμένο μητρώο $A^{(m+1)}$ να γίνει διαγώνιο. Μετά από κάθε βήμα αναβαθμίζουμε το μητρώο $A^{(m+1)}$ και ενημερώνουμε τις τιμές των $D^{(m)}$, $L^{(m)}$ και $U^{(m)}$ για την επόμενη επανάληψη. Όταν ο μετρητής που έχουμε θέσει φτάσει στο τελευταίο βήμα του αλγορίθμου, τότε λύνεται απευθείας το σύστημα πολλαπλασιάζοντας το αντίστροφο του $A^{(m+1)}$ με κάθε δεξί μέλος (στήλη) του μητρώου B^{nxn}. Αυτό γίνεται με την πραγματοποίηση της εντολής **x(:, j) = Amplus1 \ B1(:, j)** μέσα σε έναν εσωτερικό επαναληπτικό βρόγχο για κάθε στήλη του B^{nxn}.

Σε ένα νέο αρχείο **ask2.m** τρέχουμε τον εξής κώδικα:

```
% Ερώτημα 1
n = 5; % Ορισμός του μεγέθους των μητρώων A, B διάστασης nxn
A = full(gallery('tridiag', n, randn(1, n-1), randn(1, n), randn(1, n-1)));
B = randn(n, n); % Δημιουργία ενός τυχαίου μητρώου B διάστασης nxn, δηλαδή έχουμε n διανύσματα b nx1 για n διαφορετικά συστήματα Ax = b

disp('Τριδιαγώνιο Μητρώο A:');
disp(num2str(A));

disp('Διανύσματα b (Στήλες Μητρώου B):')
disp(num2str(B));

x = tridiagonal_1084661(A, B); % Επιλύουμε τα συστήματα με τη συνάρτηση που κατασκευάσαμε
disp('Λύσεις συστημάτων Ax = b:')
disp(num2str(x));
```

Ορίζουμε ως διάσταση του μητρώου A 5x5 και ως διάσταση του μητρώου B 5x5, το οποίο περιέχει 5 διανύσματα b διάστασης 5x1. Το μητρώο B^{5x5} το κατασκευάζουμε μέσω της **randn( )**, ενώ το τριδιαγώνιο μητρώο A^{5x5} μέσω της έτοιμης συνάρτησης **full( )** με όρισμα 'tridiag' για να φέρει το μητρώο σε τριδιαγώνια μορφή. Ορίζουμε στο A το πλήθος των διαγώνιων τιμών να είναι 5 και να λαμβάνονται από κανονική κατανομή. Επίσης, ορίζουμε το πλήθος των 1- υπερ/υπο - διαγώνιων τιμών να είναι 4 και να λαμβάνονται και αυτές από κανονική κατανομή. Στο τέλος, καλούμε τη συνάρτηση επίλυσης τριδιαγώνιου συστήματος που κατασκευάσαμε και υπολογίζουμε έτσι τη λύση του συστήματος x^{5x1}.

Εκτέλεση Ερωτήματος 1:

```
Τριδιαγώνιο Μητρώο A:
-0.57669   -0.73482         0         0         0
 0.17853   -0.061528   0.78838         0         0
       0       1.676   0.72559   -1.9654         0
       0           0   -0.3251   -1.2273   -1.9555
       0           0         0   0.10115   0.19093
Διανύσματα b (Στήλες Μητρώου B):
  1.6243    0.30739    1.0478   -0.50723   -1.1181
-0.14942    0.33755    2.0589   -1.4831   -0.59694
 -1.9683     1.067   0.026693   -0.10649   -0.67949
 -1.6861    0.33294   -0.24052    0.24526    0.95505
-0.086922  -0.073484    1.0385    0.14579   -0.1308
  -2.7088         0   -9.4155         0         0
        0   -2.1101         0    2.1355         0
   4.8631         0   22.7216         0    3.1314          A(1)
        0    0.7509         0   -1.0720         0
        0         0   -0.0268         0    0.0298

 -0.69363         0         0         0    1.2976
        0   -0.61413         0         0         0
        0         0    8.6362         0         0          A(2)
```

```
     0         0         0   -0.31199         0
0.0057343         0         0         0   0.033464
-0.91599         0         0         0         0
       0   -0.61413         0         0         0
       0         0    8.6362         0         0              A(3)
       0         0         0   -0.31199         0
       0         0         0         0   0.044192
Λύσεις συστημάτων Ax = b:
  0.174895    -4.73655    -27.9876     19.8904     9.00364
   4.54446    -2.28246    -2.87156     2.34765     1.61058
-0.0692634   -0.879387    -6.53548     4.71101     1.98125
 -0.275641    -1.94719     34.9006     3.84707    -8.33093
   1.17752    -2.28373     23.9478     2.84171    -4.7409
```

Σημείωση:
Εκτός από τα αρχικά μητρώα A^{5x5} και B^{5x5} και τις τελικές λύσεις των συστημάτων, έχουμε βάλει να εκτυπώνεται και κάθε ενδιάμεσο βήμα του αλγορίθμου που μετασχηματίζει το $A^{(0)}$ σε $A^{(3)}$. Ακόμα, το μητρώο B^{5x5} δεν είναι απαραίτητο να είναι συμμετρικό, η υλοποίηση γίνεται σωστά και για μητρώα της μορφής B^{5xm}.

(2.2)
Με πολλαπλασιασμούς **Hadamard** τα στοιχεία των δύο πινάκων πολλαπλασιάζονται στοιχείο-προς-στοιχείο, χωρίς την χρήση αθροιστή. Έτσι, ο αλγόριθμος γίνεται πολύ απλός στην υλοποίηση, καθώς απαιτεί μόνο την στοιχειακή πολλαπλασιαστική λειτουργία, επιτρέποντας έτσι την παράλληλη εκτέλεση των πράξεων. Συνεπώς, **θα μειώνει και αισθητά το αριθμητικό κόστος υπολογισμού** σε σύγκριση με τον κλασικό πολλαπλασιασμό μητρώων. Στο κώδικα αυτό επιτυγχάνεται αν αντικαταστήσουμε τις εντολές όπου υπολογίζουμε τα $D^{(1)}$, $L^{(1)}$ και $U^{(1)}$ με τις ακόλουθες:

```
D1 = D - L .* (U ./ D);
L1 = L .* (L ./ D);
U1 = U .* (U ./ D);
A1 = D1 - L1 - U1; % Μετασχηματίζουμε το μητρώο A(0) σε A(1)

% Μετασχηματίζουμε τα διανύσματα b(0) του μητρώου B(0) σε b(1) στο
% μητρώο B(1) οπότε ολοκληρώνεται το πρώτο βήμα του αλγορίθμου
 B0 = B;
 B1 = B0 + L .* (B0 ./ D) + U .* (B0 ./ D);
```

Με τη προσθήκη του μπλοκ εντολών:

```
tic;  % Εκκίνηση χρονομέτρησης

//Κλήση της tridiagonal_1084661( )

cost = toc;  % Συνολικός χρόνος εκτέλεσης
disp('Χρόνος εκτέλεσης:');
disp(num2str(cost));
```

μπορούμε να χρονομετρήσουμε τον πραγματικό χρόνο εκτέλεσης του αλγορίθμου. Αν τρέξουμε το ίδιο μπλοκ εντολών και για τον αλγόριθμο με υλοποίηση Hadamard στο ίδιο παράδειγμα επιβεβαιώνουμε την αύξηση της ταχύτητας και άρα τη μείωση του κόστους.
Ο χρόνος εκτέλεσης φαίνεται παρακάτω για το παράδειγμα που υλοποιήσαμε πιο πάνω:

- **Χωρίς Πολλαπλασιασμούς Hadamard:**

```
Χρόνος εκτέλεσης:
0.011171
```

- **Με Πολλαπλασιασμούς Hadamard:**

```
Χρόνος εκτέλεσης:
0.0020069
```

Γενικά, το αριθμητικό κόστος του αλγορίθμου μας με πολλαπλασιασμούς Hadamard θα εξαρτάται μόνο από τις διαστάσεις n των μητρώων μας.

(2.3)

```
% Ερώτημα 3
A = zeros(n, n);
for i = 1 : n
   for j = 1 : n
      if i~= j
         A(i, j) = randn(); % Τυχαίο γέμισμα μη διαγώνιων κελιών του μητρώου
      end
   end
end

A = A + diag(sum(abs(A), 2) + randn()); % Δημιουργία τυχαίου διαγώνια κυρίαρχου μητρώου
disp('Διαγώνια Κυρίαρχο Μητρώο Α:');
disp(num2str(A));

upper_diag_norm = norm(triu(A), 2); % Υπολογισμός νόρμας υπερδιαγωνίων
disp('Νόρμα L2 υπερδιαγωνίων');
disp(num2str(upper_diag_norm));

lower_diag_norm = norm(tril(A), 2); % Υπολογισμός νόρμας υποδιαγωνίων
disp('Νόρμα L2 υποδιαγωνίων');
disp(num2str(lower_diag_norm));
```

Ένα μητρώο για να είναι αυστηρά διαγώνια κυρίαρχο πρέπει το κάθε στοιχείο της κύριας διαγωνίου του να είναι κατ' απόλυτο μεγαλύτερο από το απόλυτο άθροισμα των στοιχείων της αντίστοιχης γραμμής. Άρα, φτιάχνουμε τώρα ένα τυχαίο μητρώο Α πάλι 5x5 και σε κάθε στοιχείο του πέραν των διαγωνίων (i, i) λαμβάνει μια τυχαία τιμή. Έπειτα, τα διαγώνια στοιχεία (i, i) προκύπτουν από το άθροισμα όλων των στοιχειών της i – γραμμής κατ' απόλυτο μέσω της **sum(abs(A), 2)** συν μια τυχαία τιμή, προκειμένου να εξασφαλίσουμε ότι η διαγώνιος είναι αυστηρά μεγαλύτερη. Τα διαγώνια αυτά στοιχεία προστίθενται στον αρχικό πίνακα και έτσι προκύπτει το τελικό μητρώο. Τέλος, μέσω της συνάρτησης **norm( )** με ορίσματα **triu(A)** ή **tril(A)** αντίστοιχα για υπερ / υπο – διαγωνίους και τη τιμή '2' υπολογίζουμε την νόρμα L2 των υπερδιαγωνίων και υποδιαγωνίων.

Εκτέλεση Ερωτήματος 3:

```
Νόρμα L2 υπερδιαγωνίων
5.1318
Νόρμα L2 υποδιαγωνίων
5.2441
```

Συμπεράσματα:

Καθώς πειραματιζόμαστε με τη παραγωγή τυχαίων αυστηρά διαγώνια κυρίαρχων μητρώων, παρατηρούμε ότι δεν παραμένει η νόρμα κάποιας διαγωνίου σταθερά μεγαλύτερη ή μικρότερη από τη νόρμα της άλλης διαγωνίου, αλλά οι τιμές τους εναλλάσσονται. Γενικά, όταν η νόρμα είναι μεγαλύτερη μπορούμε να αποφανθούμε ποια πλευρά του διαγώνιου μητρώου έχει μεγαλύτερη βαρύτητα στη συμπεριφορά του μητρώου. Στο συγκεκριμένο παράδειγμα, εφόσον η νόρμα της υποδιαγωνίου είναι μεγαλύτερη συμπεραίνουμε ότι η κάτω διαγώνια πλευρά του μητρώου A^{5X5} είναι πιο σημαντική και μπορεί έτσι να επηρεάσει ανάλογα την επίλυση ενός προβλήματος στο οποίο χρησιμοποιείται το συγκεκριμένο μητρώο.

3 Τανυστές

(3.1) Στο αρχείο **ttv_1084661.m** υλοποιούμε τον εξής κώδικα:

```
function Y = ttv_1084661(X, V, N)

   if ~isnumeric(X) || ndims(X) ~= 3 % Ελέγχουμε εάν το X είναι τανυστής 3 τρόπων
      error('Το X πρέπει να είναι τανυστής 3 τρόπων.');
   end

   if ~isvector(V) % Ελέγχουμε εάν το V είναι διάνυσμα(στήλης)
      error('Το V πρέπει να είναι διάνυσμα.');
   end

   if not(any(length(V) == size(X))) %'Ελεγχος αν η διάσταση του διανύσματος ταιριάζει με οποιαδήποτε από
τις διαστάσεις του τανυστή
      error('Η διάσταση του διανύσματος δεν είναι συμβατή με τη διάσταση του τανυστή.');
   end

   I = size(X, 1);
   J = size(X, 2);
   K = size(X, 3);

   if N == 1
      Y = squeeze(sum(bsxfun(@times, X, reshape(V, I, 1, 1)), 1));
   elseif N == 2
      Y = squeeze(sum(bsxfun(@times, X, reshape(V, 1, J, 1)), 2));
   elseif N == 3
      Y = squeeze(sum(bsxfun(@times, X, reshape(V, 1, 1, K)), 3));
   end
end
```

Αρχικά, κάνουμε τους απαραίτητους ελέγχους στον τανυστή X^{IxJxK}, ελέγχοντας αν έχει πλήθος διαστάσεων ίσο με 3 μέσω της **ndims( )**, δηλαδή αν είναι mode – 3. Σε διαφορετική περίπτωση εκτυπώνουμε κατάλληλο ενημερωτικό μήνυμα. Ακόμα, ελέγχουμε αν το στοιχείο *V* αποτελεί διάνυσμα στήλης με την **isvector( )** και αν η παράμετρος N είναι συμβατή με τις διαστάσεις του τρισδιάστατου τανυστή. Πάλι αν οι έλεγχοι δεν επαληθευτούν εκτυπώνουμε αντίστοιχα μηνύματα. Στη συνέχεια, αποθηκεύουμε στις μεταβλητές I, J , K τη $1^η$, $2^η$ και $3^η$ διάσταση αντίστοιχα του τανυστή και διακρίνουμε περιπτώσεις:

1. Αν N = 1 σημαίνει ότι πολλαπλασιάζουμε ως προς τη $1^η$ διάσταση του διανύσματος, οπότε μετασχηματίζουμε το V σε διάσταση Ix1x1. Με τη **bsxfun( )** πολλαπλασιάζουμε στοιχείο προς στοιχείο τα *X, V* με το πολλαπλασιασμό να γίνεται στη πρώτη διάσταση, ενώ με τη **sum( )** υπολογίζουμε το άθροισμα των στοιχείων στη πρώτη διάσταση. Έτσι, προκύπτει ένας τανυστής 1xJxK και για να τον μετατρέψουμε στη τελική μορφή αφαιρούμε τις διαστάσεις με μέγεθος 1 για να καταλήξουμε στη μορφή JxK.

2. Αν N = 2 σημαίνει ότι πολλαπλασιάζουμε ως προς τη $2^η$ διάσταση του διανύσματος, οπότε μετασχηματίζουμε το V σε διάσταση 1xJx1. Με τη **bsxfun( )** πολλαπλασιάζουμε στοιχείο προς στοιχείο τα X, V με το πολλαπλασιασμό να γίνεται στη δεύτερη διάσταση, ενώ με τη sum( ) υπολογίζουμε το άθροισμα των στοιχείων στη δεύτερη διάσταση. Έτσι, προκύπτει ένας τανυστής Ix1xK και για να τον μετατρέψουμε στη τελική μορφή αφαιρούμε τις διαστάσεις με μέγεθος 1 για να καταλήξουμε στη μορφή IxK.

3. Αν N = 3 σημαίνει ότι πολλαπλασιάζουμε ως προς τη $3^η$ διάσταση του διανύσματος, οπότε μετασχηματίζουμε το V σε διάσταση 1x1xK. Με τη bsxfun( ) πολλαπλασιάζουμε στοιχείο προς στοιχείο τα X, V με το πολλαπλασιασμό να γίνεται στη τρίτη διάσταση, ενώ με τη sum( ) υπολογίζουμε το άθροισμα των στοιχείων στη τρίτη διάσταση. Έτσι, προκύπτει ένας τανυστής IxJx1 και για να τον μετατρέψουμε στη τελική μορφή αφαιρούμε τις διαστάσεις με μέγεθος 1 για να καταλήξουμε στη μορφή IxJ.

Στο αρχείο **ask3.m** υλοποιούμε τον εξής κώδικα:

```
% Ερώτημα 1
X = rand(4, 3, 2); % Δημιουργία τυχαίου 4x3x2 τανυστή X
disp('Αρχικός τανυστής X:');
disp(X);

N = 1; % Επιλογή της διάστασης N ως προς την οποία θα γίνει ο πολλαπλασιασμός
```

```
disp(['Επιλεγμένη διάσταση N για τροπικό πολλαπλασιασμό: ', num2str(N)]);

[K, L, M] = size(X);
V = rand(K, 1); % Δημιουργία ενός τυχαίου διανύσματος V διάστασης {4,3,2}x1
disp('Διάνυσμα V:');
disp(V);

Y = ttv_1084661(X, V, N); % Κλήση της συνάρτησης ttv_1084661 για τροπικό πολλαπλασιασμό
disp('Τελικός τανυστής Y:');
disp(Y);
```

Κατασκευάζουμε ένα τυχαίο τανυστή X^{4x3x2} και ένα τυχαίο διάνυσμα V^{4x1} μέσω της rand( ), δηλαδή επιλέγουμε να γίνει ο τροπικός πολλαπλασιασμός ως προς τη πρώτη διάσταση. Μετά καλούμε τη συνάρτηση ttv_1084661(X, V, N) και εκτυπώνουμε τον τελικό τανυστή Y^{3x2}.

Εκτέλεση Ερωτήματος 1:

```
Αρχικός τανυστής X:

(:,:,1) =

   0.4446   0.3866   0.7678
   0.8775   0.4632   0.1232
   0.8707   0.5652   0.2231
   0.4791   0.6954   0.8904


(:,:,2) =

   0.7012   0.7103   0.8783
   0.9972   0.5668   0.2467
   0.6424   0.6153   0.5196
   0.0894   0.0629   0.6447

Επιλεγμένη διάσταση N για τροπικό πολλαπλασιασμό: 1
Διάνυσμα V:
   0.7005
   0.3114
   0.9350
   0.1487

Τελικός τανυστής Y:
   1.4700   1.4156
   1.0469   1.2588
   0.9172   1.2737
```

(3.2) Στο αρχείο **ttt_1084661.m** υλοποιούμε τον εξής κώδικα:

```
function C = ttt_1084661(A, B, mode)
    sizeA = size(A);
    sizeB = size(B);

    if nargin == 3 && strcmp(mode, 'all') % Ελέγχουμε εάν το mode είναι ορισμένο και είναι 'all'
        if sizeA ~= sizeB %'Ελεγχος συμβατότητας
            disp('Οι τανυστές δεν έχουν συμβατά μεγέθη για εσωτερικό γινόμενο.');
        else
            C = dot(A(:), B(:)); % Εσωτερικό γινόμενο των τανυστών
        end
    else
        % Εξωτερικό γινόμενο των τανυστών
```

```
    C = reshape(A, [], 1) * reshape(B, 1, []); % Μετασχηματίζουμε τους τανυστές σε διάνυσμα στήλης και γραμμής με τον αριθμό γραμμών/στηλών να προσαρμόζεται αυτόματα

    % Διαμόρφωση του αποτελέσματος στις σωστές διαστάσεις
    C = reshape(C, [sizeA, sizeB]); % Μετασχηματίζουμε το διδιάστατο μητρώο που περιλαμβάνει το εξωτερικό γινόμενο των στοιχείων των διανυσμάτων πάλι σε τανυστή
  end
end
```

Αρχικά, ελέγχουμε ποιο mode περνιέται σαν όρισμα μαζί με τους τανυστές A, B. Αν κατά τη κλήση της συνάρτησης δεχόμαστε και τρίτη είσοδο η οποία είναι το αλφαριθμητικό 'all', τότε ελέγχουμε αν οι δύο πολυδιάστατοι πίνακες έχουν ίδια μεγέθη και αν όχι τότε εκτυπώνουμε ανάλογο μήνυμα, καθώς το εσωτερικό γινόμενο δεν μπορεί να υπολογιστεί. Διαφορετικά, μεταβαίνουμε κανονικά στον υπολογισμό του μέσω της συνάρτησης dot( ), η οποία πολλαπλασιάζει στοιχείο προς στοιχείο το πρώτο διάνυσμα με το δεύτερο. Τα A(:) και B(:) αντιπροσωπεύουν τα διανύσματα όπου ξεδιπλώνουμε τους τανυστές, καθώς κάθε διάστασή τους είναι μια τιμή του αντίστοιχου τανυστή ανεξάρτητα από τη διάστασή του στην οποία βρισκόμαστε. Στο εσωτερικό γινόμενο, το τελικό αποτέλεσμα που λαμβάνουμε είναι αριθμητική τιμή. Αλλιώς αν κατά τη κλήση της συνάρτησης δε δεχόμαστε τρίτο όρισμα, τότε εκτελούμε το εξωτερικό γινόμενο των τανυστών μετασχηματίζοντας μέσω της reshape( ) τον A σε διάνυσμα στήλης και τον B σε διάνυσμα γραμμής και διατηρώντας την ίδια ακολουθία των στοιχείων. Το εξωτερικό γινόμενο ενός τανυστή N – τρόπων με έναν M – τρόπων είναι ένας τανυστής N + M τρόπων, άρα στη συγκεκριμένη περίπτωση θα είναι ένας τανυστής 6 – τρόπων. Γι' αυτό το λόγο, μετασχηματίζουμε το τελικό αποτέλεσμα πάλι σε έναν τανυστή, μέσω της reshape( ), έτσι ώστε να το επαναφέρουμε στο αρχικό του σχήμα.

Στο αρχείο **ask3.m** προσθέτουμε τον εξής κώδικα:

```
% Ερώτημα 2
A = rand(2,3,4); % Ορισμός τυχαίου 2x3x4 τανυστή A
disp(A);
B = rand(2,3,4); % Ορισμός τυχαίου 2x3x4 τανυστή B
disp(B);

C = ttt_1084661(A, B);
disp('Εξωτερικό γινόμενο των τανυστών A, B:');
disp(C);

t = ttt_1084661(A, B, 'all');
disp('Εσωτερικό γινόμενο των τανυστών A, B:');
disp(num2str(t));
```

Κατασκευάζουμε ένα τυχαίο τανυστή X^{4x3x2} και ένα τυχαίο διάνυσμα V^{4x1} μέσω της rand( ), δηλαδή επιλέγουμε να γίνει ο τροπικός πολλαπλασιασμός ως προς τη πρώτη διάσταση. Μετά καλούμε τη συνάρτηση ttv_1084661(X, V, N) και εκτυπώνουμε τον τελικό τανυστή Y^{3x2}. Επιβεβαιώνουμε την ορθή λειτουργία της συνάρτησης καλώντας έπειτα και την κανονική συνάρτηση ttt( ) του Tensor Toolbox και όντως επιβεβαιώνεται η ορθή λειτουργία της συνάρτησης. Φτιάχνουμε πάλι δυο τυχαίους τανυστές A^{2x3x4} και B^{2x3x4} και καλούμε τη συνάρτηση ttt_1084661(A, B), η οποία επιστρέφει στο C το τελικό τανυστή 6 – τρόπων που θα αποτελεί το εξωτερικό γινόμενο. Αντίστοιχα, καλώντας την ίδια συνάρτηση μαζί με όρισμα 'all' εκτελούμε το εσωτερικό γινόμενο των A, B και το αποθηκεύουμε στη μεταβλητή t. Αντίστοιχα καλούμε και τη συνάρτηση συνάρτηση ttv( ) του Tensor Toolbox, η οποία ξανά θα επαληθεύσει την ορθότητα των αποτελεσμάτων μας.

Εκτέλεση Ερωτήματος 2:

```
Εξωτερικό γινόμενο των τανυστών A, B:

(:,:,1,1,1,1) =

   0.6002   0.6534   0.7646
   0.5966   0.3255   0.1380


(:,:,2,1,1,1) =
```

```
   0.3801   0.2810   0.1689
   0.1584   0.2998   0.1125


(:,:,3,1,1,1) =

   0.2893   0.3179   0.0856
   0.2769   0.7174   0.7106


(:,:,4,1,1,1) =

   0.7607   0.6437   0.3295
   0.8089   0.1174   0.0514


(:,:,1,2,1,1) =

   0.2715   0.2955   0.3458
   0.2699   0.1472   0.0624


(:,:,2,2,1,1) =

   0.1719   0.1271   0.0764
   0.0717   0.1356   0.0509


(:,:,3,2,1,1) =

   0.1308   0.1438   0.0387
   0.1252   0.3245   0.3214


(:,:,4,2,1,1) =

   0.3441   0.2912   0.1490
   0.3659   0.0531   0.0232


(:,:,1,1,2,1) =

   0.3149   0.3428   0.4012
   0.3130   0.1708   0.0724


(:,:,2,1,2,1) =

   0.1994   0.1474   0.0886
   0.0831   0.1573   0.0590


(:,:,3,1,2,1) =

   0.1518   0.1668   0.0449
   0.1453   0.3764   0.3728


(:,:,4,1,2,1) =
```

```
   0.3991   0.3377   0.1729
   0.4244   0.0616   0.0269


(:,:,1,2,2,1) =

   0.6412   0.6980   0.8168
   0.6374   0.3477   0.1474


(:,:,2,2,2,1) =

   0.4061   0.3002   0.1804
   0.1693   0.3203   0.1202


(:,:,3,2,2,1) =

   0.3090   0.3396   0.0915
   0.2958   0.7664   0.7591


(:,:,4,2,2,1) =

   0.8126   0.6876   0.3520
   0.8641   0.1255   0.0549


(:,:,1,1,3,1) =

   0.4256   0.4633   0.5422
   0.4231   0.2308   0.0979


(:,:,2,1,3,1) =

   0.2695   0.1992   0.1197
   0.1123   0.2126   0.0798


(:,:,3,1,3,1) =

   0.2051   0.2254   0.0607
   0.1963   0.5087   0.5038


(:,:,4,1,3,1) =

   0.5394   0.4564   0.2336
   0.5736   0.0833   0.0364


(:,:,1,2,3,1) =

   0.0343   0.0373   0.0437
   0.0341   0.0186   0.0079


(:,:,2,2,3,1) =
```

```
   0.0217   0.0161   0.0096
   0.0091   0.0171   0.0064


(:,:,3,2,3,1) =

   0.0165   0.0182   0.0049
   0.0158   0.0410   0.0406


(:,:,4,2,3,1) =

   0.0435   0.0368   0.0188
   0.0462   0.0067   0.0029


(:,:,1,1,1,2) =

   0.2852   0.3104   0.3633
   0.2835   0.1546   0.0656


(:,:,2,1,1,2) =

   0.1806   0.1335   0.0802
   0.0753   0.1425   0.0535


(:,:,3,1,1,2) =

   0.1374   0.1510   0.0407
   0.1316   0.3409   0.3376


(:,:,4,1,1,2) =

   0.3614   0.3058   0.1566
   0.3843   0.0558   0.0244


(:,:,1,2,1,2) =

   0.6821   0.7425   0.8689
   0.6780   0.3699   0.1568


(:,:,2,2,1,2) =

   0.4319   0.3193   0.1919
   0.1800   0.3407   0.1279


(:,:,3,2,1,2) =

   0.3287   0.3612   0.0973
   0.3146   0.8153   0.8075


(:,:,4,2,1,2) =
```

```
   0.8645   0.7315   0.3744
   0.9192   0.1335   0.0584


(:,:,1,1,2,2) =

   0.0787   0.0856   0.1002
   0.0782   0.0427   0.0181


(:,:,2,1,2,2) =

   0.0498   0.0368   0.0221
   0.0208   0.0393   0.0148


(:,:,3,1,2,2) =

   0.0379   0.0417   0.0112
   0.0363   0.0940   0.0931


(:,:,4,1,2,2) =

   0.0997   0.0844   0.0432
   0.1060   0.0154   0.0067


(:,:,1,2,2,2) =

   0.0936   0.1018   0.1192
   0.0930   0.0507   0.0215


(:,:,2,2,2,2) =

   0.0592   0.0438   0.0263
   0.0247   0.0467   0.0175


(:,:,3,2,2,2) =

   0.0451   0.0495   0.0133
   0.0432   0.1118   0.1108


(:,:,4,2,2,2) =

   0.1186   0.1003   0.0514
   0.1261   0.0183   0.0080


(:,:,1,1,3,2) =

   0.5669   0.6171   0.7222
   0.5635   0.3074   0.1304


(:,:,2,1,3,2) =
```

```
   0.3590   0.2654   0.1595
   0.1496   0.2832   0.1063


(:,:,3,1,3,2) =

   0.2732   0.3002   0.0809
   0.2615   0.6776   0.6711


(:,:,4,1,3,2) =

   0.7185   0.6080   0.3112
   0.7640   0.1109   0.0485


(:,:,1,2,3,2) =

   0.3708   0.4036   0.4723
   0.3685   0.2010   0.0853


(:,:,2,2,3,2) =

   0.2348   0.1736   0.1043
   0.0979   0.1852   0.0695


(:,:,3,2,3,2) =

   0.1787   0.1964   0.0529
   0.1710   0.4431   0.4389


(:,:,4,2,3,2) =

   0.4699   0.3976   0.2035
   0.4997   0.0725   0.0317


(:,:,1,1,1,3) =

   0.5987   0.6517   0.7626
   0.5951   0.3246   0.1377


(:,:,2,1,1,3) =

   0.3791   0.2803   0.1684
   0.1580   0.2991   0.1123


(:,:,3,1,1,3) =

   0.2885   0.3171   0.0854
   0.2762   0.7156   0.7087


(:,:,4,1,1,3) =
```

```
   0.7588   0.6420   0.3287
   0.8068   0.1171   0.0512


(:,:,1,2,1,3) =

   0.4227   0.4601   0.5385
   0.4202   0.2292   0.0972


(:,:,2,2,1,3) =

   0.2677   0.1979   0.1189
   0.1116   0.2112   0.0793


(:,:,3,2,1,3) =

   0.2037   0.2239   0.0603
   0.1950   0.5052   0.5004


(:,:,4,2,1,3) =

   0.5357   0.4533   0.2320
   0.5696   0.0827   0.0362


(:,:,1,1,2,3) =

   0.2578   0.2807   0.3285
   0.2563   0.1398   0.0593


(:,:,2,1,2,3) =

   0.1633   0.1207   0.0725
   0.0681   0.1288   0.0483


(:,:,3,1,2,3) =

   0.1243   0.1366   0.0368
   0.1189   0.3082   0.3052


(:,:,4,1,2,3) =

   0.3268   0.2765   0.1415
   0.3475   0.0504   0.0221


(:,:,1,2,2,3) =

   0.1637   0.1782   0.2086
   0.1628   0.0888   0.0376


(:,:,2,2,2,3) =
```

```
   0.1037   0.0767   0.0461
   0.0432   0.0818   0.0307


(:,:,3,2,2,3) =

   0.0789   0.0867   0.0234
   0.0755   0.1957   0.1938


(:,:,4,2,2,3) =

   0.2075   0.1756   0.0899
   0.2207   0.0320   0.0140


(:,:,1,1,3,3) =

   0.3319   0.3613   0.4228
   0.3299   0.1800   0.0763


(:,:,2,1,3,3) =

   0.2102   0.1554   0.0934
   0.0876   0.1658   0.0622


(:,:,3,1,3,3) =

   0.1600   0.1758   0.0473
   0.1531   0.3967   0.3929


(:,:,4,1,3,3) =

   0.4206   0.3559   0.1822
   0.4473   0.0649   0.0284


(:,:,1,2,3,3) =

   0.2359   0.2568   0.3005
   0.2345   0.1279   0.0542


(:,:,2,2,3,3) =

   0.1494   0.1104   0.0664
   0.0623   0.1178   0.0442


(:,:,3,2,3,3) =

   0.1137   0.1249   0.0337
   0.1088   0.2820   0.2793


(:,:,4,2,3,3) =
```

```
   0.2990   0.2530   0.1295
   0.3179   0.0462   0.0202


(:,:,1,1,1,4) =

   0.6641   0.7229   0.8459
   0.6601   0.3601   0.1527


(:,:,2,1,1,4) =

   0.4206   0.3109   0.1868
   0.1753   0.3317   0.1245


(:,:,3,1,1,4) =

   0.3201   0.3517   0.0947
   0.3063   0.7937   0.7862


(:,:,4,1,1,4) =

   0.8417   0.7122   0.3646
   0.8950   0.1299   0.0568


(:,:,1,2,1,4) =

   0.1920   0.2091   0.2446
   0.1909   0.1041   0.0442


(:,:,2,2,1,4) =

   0.1216   0.0899   0.0540
   0.0507   0.0959   0.0360


(:,:,3,2,1,4) =

   0.0926   0.1017   0.0274
   0.0886   0.2295   0.2273


(:,:,4,2,1,4) =

   0.2434   0.2060   0.1054
   0.2588   0.0376   0.0164


(:,:,1,1,2,4) =

   0.4590   0.4996   0.5847
   0.4562   0.2489   0.1055


(:,:,2,1,2,4) =
```

```
   0.2907   0.2149   0.1291
   0.1212   0.2293   0.0861


(:,:,3,1,2,4) =

   0.2212   0.2431   0.0655
   0.2117   0.5486   0.5434


(:,:,4,1,2,4) =

   0.5817   0.4922   0.2520
   0.6185   0.0898   0.0393


(:,:,1,2,2,4) =

   0.5525   0.6014   0.7038
   0.5492   0.2996   0.1270


(:,:,2,2,2,4) =

   0.3499   0.2586   0.1554
   0.1458   0.2760   0.1036


(:,:,3,2,2,4) =

   0.2663   0.2926   0.0788
   0.2549   0.6603   0.6540


(:,:,4,2,2,4) =

   0.7002   0.5925   0.3033
   0.7445   0.1081   0.0473


(:,:,1,1,3,4) =

   0.5853   0.6371   0.7456
   0.5818   0.3174   0.1346


(:,:,2,1,3,4) =

   0.3707   0.2740   0.1647
   0.1545   0.2924   0.1097


(:,:,3,1,3,4) =

   0.2821   0.3100   0.0835
   0.2700   0.6996   0.6929


(:,:,4,1,3,4) =
```

```
   0.7418   0.6277   0.3213
   0.7888   0.1145   0.0501


(:,:,1,2,3,4) =

   0.0580   0.0632   0.0739
   0.0577   0.0315   0.0133


(:,:,2,2,3,4) =

   0.0367   0.0272   0.0163
   0.0153   0.0290   0.0109


(:,:,3,2,3,4) =

   0.0280   0.0307   0.0083
   0.0268   0.0694   0.0687


(:,:,4,2,3,4) =

   0.0735   0.0622   0.0319
   0.0782   0.0114   0.0050

Εσωτερικό γινόμενο των τανυστών A, B:
5.9533
```

Επαληθεύουμε την ορθή λειτουργία των συναρτήσεων που υλοποιησαμε καλώντας τις αντίστοιχες συναρτήσεις ttt( ) και ttv( ) του Tensor Toolbox, η οποίες όντως επαληθεύουν την ορθότητα των αποτελεσμάτων μας.

```
 B = ttv(tensor(X), V, N);
disp(B);

D = ttt(tensor(A), tensor(B));
disp(D);
```

```
ans is a tensor of size 3 x 2
        ans(:,:) =
           1.4700   1.4156
           1.0469   1.2588
           0.9172   1.2737

ans is a tensor of size 2 x 3 x 4 x 2 x 3 x 4
        ans(:,:,1,1,1,1) =
           0.6002   0.6534   0.7646
           0.5966   0.3255   0.1380
        ans(:,:,2,1,1,1) =
           0.3801   0.2810   0.1689
           0.1584   0.2998   0.1125
        ans(:,:,3,1,1,1) =
           0.2893   0.3179   0.0856
           0.2769   0.7174   0.7106
        ans(:,:,4,1,1,1) =
           0.7607   0.6437   0.3295
           0.8089   0.1174   0.0514
```

```
ans(:,:,1,2,1,1) =
   0.2715   0.2955   0.3458
   0.2699   0.1472   0.0624
ans(:,:,2,2,1,1) =
   0.1719   0.1271   0.0764
   0.0717   0.1356   0.0509
ans(:,:,3,2,1,1) =
   0.1308   0.1438   0.0387
   0.1252   0.3245   0.3214
ans(:,:,4,2,1,1) =
   0.3441   0.2912   0.1490
   0.3659   0.0531   0.0232
ans(:,:,1,1,2,1) =
   0.3149   0.3428   0.4012
   0.3130   0.1708   0.0724
ans(:,:,2,1,2,1) =
   0.1994   0.1474   0.0886
   0.0831   0.1573   0.0590
ans(:,:,3,1,2,1) =
   0.1518   0.1668   0.0449
   0.1453   0.3764   0.3728
ans(:,:,4,1,2,1) =
   0.3991   0.3377   0.1729
   0.4244   0.0616   0.0269
ans(:,:,1,2,2,1) =
   0.6412   0.6980   0.8168
   0.6374   0.3477   0.1474
ans(:,:,2,2,2,1) =
   0.4061   0.3002   0.1804
   0.1693   0.3203   0.1202
ans(:,:,3,2,2,1) =
   0.3090   0.3396   0.0915
   0.2958   0.7664   0.7591
ans(:,:,4,2,2,1) =
   0.8126   0.6876   0.3520
   0.8641   0.1255   0.0549
ans(:,:,1,1,3,1) =
   0.4256   0.4633   0.5422
   0.4231   0.2308   0.0979
ans(:,:,2,1,3,1) =
   0.2695   0.1992   0.1197
   0.1123   0.2126   0.0798
ans(:,:,3,1,3,1) =
   0.2051   0.2254   0.0607
   0.1963   0.5087   0.5038
ans(:,:,4,1,3,1) =
   0.5394   0.4564   0.2336
   0.5736   0.0833   0.0364
ans(:,:,1,2,3,1) =
   0.0343   0.0373   0.0437
   0.0341   0.0186   0.0079
ans(:,:,2,2,3,1) =
   0.0217   0.0161   0.0096
   0.0091   0.0171   0.0064
ans(:,:,3,2,3,1) =
   0.0165   0.0182   0.0049
   0.0158   0.0410   0.0406
ans(:,:,4,2,3,1) =
   0.0435   0.0368   0.0188
   0.0462   0.0067   0.0029
```

```
ans(:,:,1,1,1,2) =
   0.2852   0.3104   0.3633
   0.2835   0.1546   0.0656
ans(:,:,2,1,1,2) =
   0.1806   0.1335   0.0802
   0.0753   0.1425   0.0535
ans(:,:,3,1,1,2) =
   0.1374   0.1510   0.0407
   0.1316   0.3409   0.3376
ans(:,:,4,1,1,2) =
   0.3614   0.3058   0.1566
   0.3843   0.0558   0.0244
ans(:,:,1,2,1,2) =
   0.6821   0.7425   0.8689
   0.6780   0.3699   0.1568
ans(:,:,2,2,1,2) =
   0.4319   0.3193   0.1919
   0.1800   0.3407   0.1279
ans(:,:,3,2,1,2) =
   0.3287   0.3612   0.0973
   0.3146   0.8153   0.8075
ans(:,:,4,2,1,2) =
   0.8645   0.7315   0.3744
   0.9192   0.1335   0.0584
ans(:,:,1,1,2,2) =
   0.0787   0.0856   0.1002
   0.0782   0.0427   0.0181
ans(:,:,2,1,2,2) =
   0.0498   0.0368   0.0221
   0.0208   0.0393   0.0148
ans(:,:,3,1,2,2) =
   0.0379   0.0417   0.0112
   0.0363   0.0940   0.0931
ans(:,:,4,1,2,2) =
   0.0997   0.0844   0.0432
   0.1060   0.0154   0.0067
ans(:,:,1,2,2,2) =
   0.0936   0.1018   0.1192
   0.0930   0.0507   0.0215
ans(:,:,2,2,2,2) =
   0.0592   0.0438   0.0263
   0.0247   0.0467   0.0175
ans(:,:,3,2,2,2) =
   0.0451   0.0495   0.0133
   0.0432   0.1118   0.1108
ans(:,:,4,2,2,2) =
   0.1186   0.1003   0.0514
   0.1261   0.0183   0.0080
ans(:,:,1,1,3,2) =
   0.5669   0.6171   0.7222
   0.5635   0.3074   0.1304
ans(:,:,2,1,3,2) =
   0.3590   0.2654   0.1595
   0.1496   0.2832   0.1063
ans(:,:,3,1,3,2) =
   0.2732   0.3002   0.0809
   0.2615   0.6776   0.6711
ans(:,:,4,1,3,2) =
   0.7185   0.6080   0.3112
   0.7640   0.1109   0.0485
```

```
ans(:,:,1,2,3,2) =
   0.3708   0.4036   0.4723
   0.3685   0.2010   0.0853
ans(:,:,2,2,3,2) =
   0.2348   0.1736   0.1043
   0.0979   0.1852   0.0695
ans(:,:,3,2,3,2) =
   0.1787   0.1964   0.0529
   0.1710   0.4431   0.4389
ans(:,:,4,2,3,2) =
   0.4699   0.3976   0.2035
   0.4997   0.0725   0.0317
ans(:,:,1,1,1,3) =
   0.5987   0.6517   0.7626
   0.5951   0.3246   0.1377
ans(:,:,2,1,1,3) =
   0.3791   0.2803   0.1684
   0.1580   0.2991   0.1123
ans(:,:,3,1,1,3) =
   0.2885   0.3171   0.0854
   0.2762   0.7156   0.7087
ans(:,:,4,1,1,3) =
   0.7588   0.6420   0.3287
   0.8068   0.1171   0.0512
ans(:,:,1,2,1,3) =
   0.4227   0.4601   0.5385
   0.4202   0.2292   0.0972
ans(:,:,2,2,1,3) =
   0.2677   0.1979   0.1189
   0.1116   0.2112   0.0793
ans(:,:,3,2,1,3) =
   0.2037   0.2239   0.0603
   0.1950   0.5052   0.5004
ans(:,:,4,2,1,3) =
   0.5357   0.4533   0.2320
   0.5696   0.0827   0.0362
ans(:,:,1,1,2,3) =
   0.2578   0.2807   0.3285
   0.2563   0.1398   0.0593
ans(:,:,2,1,2,3) =
   0.1633   0.1207   0.0725
   0.0681   0.1288   0.0483
ans(:,:,3,1,2,3) =
   0.1243   0.1366   0.0368
   0.1189   0.3082   0.3052
ans(:,:,4,1,2,3) =
   0.3268   0.2765   0.1415
   0.3475   0.0504   0.0221
ans(:,:,1,2,2,3) =
   0.1637   0.1782   0.2086
   0.1628   0.0888   0.0376
ans(:,:,2,2,2,3) =
   0.1037   0.0767   0.0461
   0.0432   0.0818   0.0307
ans(:,:,3,2,2,3) =
   0.0789   0.0867   0.0234
   0.0755   0.1957   0.1938
ans(:,:,4,2,2,3) =
   0.2075   0.1756   0.0899
   0.2207   0.0320   0.0140
```

```
ans(:,:,1,1,3,3) =
   0.3319   0.3613   0.4228
   0.3299   0.1800   0.0763
ans(:,:,2,1,3,3) =
   0.2102   0.1554   0.0934
   0.0876   0.1658   0.0622
ans(:,:,3,1,3,3) =
   0.1600   0.1758   0.0473
   0.1531   0.3967   0.3929
ans(:,:,4,1,3,3) =
   0.4206   0.3559   0.1822
   0.4473   0.0649   0.0284
ans(:,:,1,2,3,3) =
   0.2359   0.2568   0.3005
   0.2345   0.1279   0.0542
ans(:,:,2,2,3,3) =
   0.1494   0.1104   0.0664
   0.0623   0.1178   0.0442
ans(:,:,3,2,3,3) =
   0.1137   0.1249   0.0337
   0.1088   0.2820   0.2793
ans(:,:,4,2,3,3) =
   0.2990   0.2530   0.1295
   0.3179   0.0462   0.0202
ans(:,:,1,1,1,4) =
   0.6641   0.7229   0.8459
   0.6601   0.3601   0.1527
ans(:,:,2,1,1,4) =
   0.4206   0.3109   0.1868
   0.1753   0.3317   0.1245
ans(:,:,3,1,1,4) =
   0.3201   0.3517   0.0947
   0.3063   0.7937   0.7862
ans(:,:,4,1,1,4) =
   0.8417   0.7122   0.3646
   0.8950   0.1299   0.0568
ans(:,:,1,2,1,4) =
   0.1920   0.2091   0.2446
   0.1909   0.1041   0.0442
ans(:,:,2,2,1,4) =
   0.1216   0.0899   0.0540
   0.0507   0.0959   0.0360
ans(:,:,3,2,1,4) =
   0.0926   0.1017   0.0274
   0.0886   0.2295   0.2273
ans(:,:,4,2,1,4) =
   0.2434   0.2060   0.1054
   0.2588   0.0376   0.0164
ans(:,:,1,1,2,4) =
   0.4590   0.4996   0.5847
   0.4562   0.2489   0.1055
ans(:,:,2,1,2,4) =
   0.2907   0.2149   0.1291
   0.1212   0.2293   0.0861
ans(:,:,3,1,2,4) =
   0.2212   0.2431   0.0655
   0.2117   0.5486   0.5434
ans(:,:,4,1,2,4) =
   0.5817   0.4922   0.2520
   0.6185   0.0898   0.0393
```

```
ans(:,:,1,2,2,4) =
   0.5525   0.6014   0.7038
   0.5492   0.2996   0.1270
ans(:,:,2,2,2,4) =
   0.3499   0.2586   0.1554
   0.1458   0.2760   0.1036
ans(:,:,3,2,2,4) =
   0.2663   0.2926   0.0788
   0.2549   0.6603   0.6540
ans(:,:,4,2,2,4) =
   0.7002   0.5925   0.3033
   0.7445   0.1081   0.0473
ans(:,:,1,1,3,4) =
   0.5853   0.6371   0.7456
   0.5818   0.3174   0.1346
ans(:,:,2,1,3,4) =
   0.3707   0.2740   0.1647
   0.1545   0.2924   0.1097
ans(:,:,3,1,3,4) =
   0.2821   0.3100   0.0835
   0.2700   0.6996   0.6929
ans(:,:,4,1,3,4) =
   0.7418   0.6277   0.3213
   0.7888   0.1145   0.0501
ans(:,:,1,2,3,4) =
   0.0580   0.0632   0.0739
   0.0577   0.0315   0.0133
ans(:,:,2,2,3,4) =
   0.0367   0.0272   0.0163
   0.0153   0.0290   0.0109
ans(:,:,3,2,3,4) =
   0.0280   0.0307   0.0083
   0.0268   0.0694   0.0687
ans(:,:,4,2,3,4) =
   0.0735   0.0622   0.0319
   0.0782   0.0114   0.0050
```

Μπορούμε να επαληθεύσουμε επίσης και τη σωστή λειτουργία τους μέσω τη συνάρτησης **test_tensor( )**.

```
function test_tensor_1084661
   % Αρχικοποίηση
   tol = 1e-8; nd = 3; rng(0); err = zeros(1, nd+2);
   ndim = [2, 3, 4]; Atemp = randi(5, ndim); Btemp = randi(4, ndim);
   X = randi([-1, 1], max(ndim), 1); A = tensor(Atemp); B = tensor(Btemp);

   % Έλεγχος των συναρτήσεων
   try
      for k = 1:nd
         err(k) = norm(ttv_1084661(A, X(1:ndim(k), 1), k) - double(ttv(A, X(1:ndim(k), 1), k)));
      end
      assert(max(err) < tol, 'ttm modal multiplication fails');
   catch ME1
   end

   try
      err(nd+1) = norm(tensor(ttt_1084661(A, B)) - ttt(A, B));
      assert(err(nd+1) < tol, 'ttt outer multiplication fails');
   catch ME2
   end

   try
```

```
        err(nd+2) = abs(ttt_1084661(A, B, 'all') - double(ttt(A, B, 1:nd)));
        assert(err(nd+2) < tol, 'ttt inner product fails');
    catch ME3
    end

    % Εκτύπωση τυχόν σφαλμάτων
    if (exist('ME1', 'var'))
        disp(ME1.message);
    end
    if (exist('ME2', 'var'))
        disp(ME2.message);
    end
    if (exist('ME3', 'var'))
        disp(ME3.message);
    end
end
```