ΑΣΚ 1, 2, 3, 4, 5, 6
Πολύ καλή προσπάθεια (ιδιαίτερα η 6), βλ. σχόλια και αναρτημένες λύσεις.

Θα παρακαλούσα σε επόμενη άσκηση να βάλεις στην αρχή έναν πίνακα περιεχομένων. Επίσης τα στοιχεία σου!

Προαιρετική Άσκηση 1

1.) ***Να περιγράψετε σε μια παράγραφο τα χαρακτηριστικά του συστήματος που χρησιμοποιείτε (υλικό και λογισμικό): Σύστημα, επεξεργαστή, επίπεδα μνήμης και μέγεθός τους.***

Το σύστημα που χρησιμοποιώ είναι ένα **laptop** με λειτουργικό σύστημα **Microsoft Windows 10 Pro Education (έκδοση 10.0.19045)**. Ο επεξεργαστής είναι ένας **AMD Athlon Gold 3150U με Radeon Graphics**, ο οποίος διαθέτει **2 πυρήνες και 4 λογικούς επεξεργαστές με συχνότητα 2.40 GHz**. Η μνήμη **RAM** του συστήματος είναι **4 GB**. Τα επίπεδα της **cache** μνήμης είναι 192 KB (L1), 1 MB (L2) και 4 MB (L3).

<u>*Σχόλιο*</u>: Το σύστημα αυτό είναι επαρκές για βασικές εργασίες σε περιβάλλον MATLAB, αν και ενδέχεται να υπάρξουν περιορισμοί σε πιο απαιτητικά έργα λόγω της χαμηλής μνήμης RAM.

2.) ***(Άσκηση A1.1.2 από GvL) Σε έναν συνήθη πολλαπλασιασμό μητρώων 2 × 2 της μορφής C = AB, υπάρχουν οκτώ πολλαπλασιασμοί: aij βjk , i, j, k = 1 , 2. Να φτιάξετε έναν πίνακα που να δείχνει τη σειρά με την οποία εκτελούνται οι πολλαπλασιασμοί αυτοί από τους αλγόριθμους ijk, jik, kij, ikj, jki, και kji πολλαπλασιασμού μητρώων (βλ. διάλεξη 4).***

Σε έναν συνήθη πολλαπλασιασμό δύο μητρώων 2×2, δηλαδή C=AB, κάθε στοιχείο του πίνακα C υπολογίζεται με βάση τα στοιχεία των πινάκων A και B. Ο πολλαπλασιασμός γίνεται ως εξής:

Για τα στοιχεία του C:

C11=A11B11+A12B21

C12=A11B12+A12B22

C21=A21B11+A22B21

C22=A21B12+A22B22

Αυτός ο πολλαπλασιασμός περιλαμβάνει οκτώ επιμέρους πολλαπλασιασμούς, που είναι οι:

A11B11,A12B21,A11B12,A12B22,A21B11,A22B21,A21B12,A22B22

Ο τρόπος με τον οποίο εκτελούνται αυτοί οι πολλαπλασιασμοί μπορεί να διαφέρει ανάλογα με τη σειρά πρόσβασης στους δείκτες των πινάκων από τους διαφορετικούς αλγόριθμους πολλαπλασιασμού. Οι αλγόριθμοι αυτοί είναι οι: ijk, jik, kij, ikj, jki, και kji. Κάθε αλγόριθμος καθορίζει τη σειρά με την οποία θα διατρέξουμε τα στοιχεία των πινάκων. Παρακάτω δίνεται ο ζητούμενος πίνακας:

Αλγόριθμος	1ος	2ος	3ος	4ος	5ος	6ος	7ος	8ος
ijk	A11B11	A12B21	A11B12	A12B22	A21B11	A22B21	A21B12	A22B22
jik	A11B1	A21B11	A12B21	A22B21	A11B12	A21B12	A12B22	A22B22
kij	A11B11	A21B11	A11B12	A21B12	A12B21	A22B21	A12B22	A22B22
ikj	A11B11	A11B12	A12B21	A12B22	A21B11	A21B12	A22B21	A22B22
jki	A11B11	A21B11	A12B21	A22B21	A11B12	A21B12	A12B22	A22B22
kji	A11B11	A21B11	A11B12	A21B12	A12B21	A22B21	A12B22	A22B22

Θα είχε ενδιαφέρον και ο "αντίστροφος" πίνακας, δηλ. αν κάθε στήλη αντιστοιχούσε σε ένα από τα μερικά γινόμενα και η κάθε γραμμή να έδειχνε την σειρά με την οποία εκτελείται το συγκεκριμένο για την εκδοχή που αντιστοιχούσε στη σειρά.

Π.χ. η γραμμή jik να ήταν 1, 5, 2, κλπ υποθέτοντας ότι τα μερικά γινόμενα είχαν την αρίθμηση της εκδοχής ijk.

3.) ***(Άσκηση A1.1.3 από GvL) Να γράψετε έναν αλγόριθμο για τον οποίο* $\Omega = O(n^2)$ για τον υπολογισμό του μητρώου $C = (xy^T)^k$, *όπου τα x και y είναι διανύσματα μεγέθους n και η μεταβλητή k είναι θετικός ακέραιος αριθμός.***

- Κάθε στοιχείο του νέου πίνακα απαιτεί n πολλαπλασιασμούς και n−1 προσθέσεις. Δηλαδή, για κάθε στοιχείο χρειαζόμαστε O(n) πράξεις.

- Δεδομένου ότι έχουμε n^2 στοιχεία στον πίνακα, η συνολική πολυπλοκότητα του πολλαπλασιασμού είναι $O(n^2)\cdot O(n)=O(n^3)$. Ωστόσο, στην περίπτωση του $C=xy^T$, μπορούμε να εκμεταλλευτούμε την ειδική δομή του πίνακα.

Καλύτερα να το υπολογίσεις ακριβώς.

η επιλογή "πίνακας" για τον όρο matrix είναι λανθασμένη.

Η απόδοση "πίνακας" για τον όρο matrix είναι ατυχής και εντέλει λανθασμένη καθώς δημιουργεί σύγχυση. Βλ. π.χ. σχόλια στην εισαγωγή του συγγράμματος (σελ. xii, υποσημ. 6) καθώς και τα σχόλια στο ΑγγλοΕλληνικό λεξικό των θεωρητικών και εφαρμοσμένων Μαθηματικών, του Μεμά Κολαΐτη (εκδ. ΤΕΕ, 1976).

Η δύναμη του πίνακα C μπορεί να υπολογιστεί πιο αποτελεσματικά ως εξής:

1. **Πρώτη Δύναμη**:
   $C=xy^T$

2. **Δεύτερη Δύναμη**:
   $C^2 = C·C = (xy^T)(xy^T) = x(y^Tx)y^T$
   Εδώ, το y^Tx υπολογίζεται ως εσωτερικό γινόμενο του y και του x. Έτσι, το C^2 γίνεται:

   $C^2=x(y^Tx)y^T$

ο υπολογισμός

Γενική Δύναμη:
Ομοίως, για οποιοδήποτε k: $C^k=x(y^Tx)^{k-1}y^T$
Αυτό σημαίνει ότι η εκτίμηση της k-οστής δύναμης του C εξαρτάται μόνο από το εσωτερικό γινόμενο y^Tx και είναι πολύ πιο αποτελεσματική από τον κλασικό υπολογισμό C·C···C k φορές.

αυτό δεν είναι ακριβές

Το κόστος είναι O(n^2) και όχι O(n) που υπονοείς παρακάτω.

Ο υπολογισμός y^Tx απαιτεί O(n) και είναι πολύ πιο γρήγορος από το $O(n^3)$ που θα απαιτούσε ο κλασικός πολλαπλασιασμός πινάκων.

Έχουμε πει ότι ο όρος "πίνακας" είναι ατυχής και λανθασμένος. Παρακαλώ να χρησιμοποιείτε τους όρους μήτρα ή μητρώο (βλ. σύγγραμμα μαθήματος, Αριθμητικής Ανάλυσης και Γραμμικής Άλγεβρας 1ου έτους - Anton).

Αλγόριθμος σε MATLAB:

```
function C = compute_matrix_power(x, y, k)

    % x: Διάνυσμα στήλη μεγέθους n
    % y: Διάνυσμα στήλη μεγέθους n
    % k: Θετικός ακέραιος

    % Υπολογισμός του εξωτερικού γινομένου xyᵀ ( O(n^2) )
    C = x * y';

    % Επαναληπτικός υπολογισμός της δύναμης του πίνακα C
    for i = 2:k
        C = C * (x * y');
      end
end
```

4.) (Άσκηση A1.1.4 από GvL) Έστω D = ABC, όπου $A \in R^{n1 \times n2}$, $B \in R^{n2 \times n3}$, και $C \in R^{n3 \times n4}$. Να συγκρίνετε το Ω ενός αλγόριθμου που υπολογίζει το D μέσω του τύπου D = (AB) C σε σχέση με το Ω ενός αλγόριθμου που - αξιοποιώντας την προσεταιριστική ιδιότητα για τους πολλαπλασιασμούς μητρώων - υπολογίζει το D χρησιμοποιώντας τον τύπο D = A (BC). Σε ποια περίπτωση η πρώτη διαδικασία είναι πιο αποδοτική από τη δεύτερη ως προς το Ω ;

Υπολογισμός του D=(AB)C

Αρχικά υπολογίζουμε το AB:

- Το A έχει διαστάσεις n1×n2 και το B έχει διαστάσεις n2×n3.
- Ο αριθμός των πράξεων για το πολλαπλασιασμό AB είναι n1·n2·n3.

Στη συνέχεια υπολογίζουμε το (AB)C:

- Το αποτέλεσμα AB έχει διαστάσεις n1×n3, και το C έχει διαστάσεις n3×n4.
- Ο αριθμός των πράξεων για το πολλαπλασιασμό (AB)C είναι n1·n3·n4.

Συνολικός αριθμός πράξεων για τον αλγόριθμο D=(AB)C:
n1·n2·n3+n1·n3·n4 Προσοχή: τα κόστη είναι τα διπλάσια.

2. Υπολογισμός του D=A(BC)

Αρχικά υπολογίζουμε το BC:

- Το B έχει διαστάσεις n2×n3 και το C έχει διαστάσεις n3×n4.
- Ο αριθμός των πράξεων για το πολλαπλασιασμό BC είναι n2·n3·n4.

Στη συνέχεια υπολογίζουμε το A(BC):

- Το αποτέλεσμα BC έχει διαστάσεις n2×n4, και το A έχει διαστάσεις n1×n2.
- Ο αριθμός των πράξεων για το πολλαπλασιασμό A(BC) είναι n1·n2·n4.

Συνολικός αριθμός πράξεων για τον αλγόριθμο D=A(BC):
n2·n3·n4+n1·n2·n4

Προσοχή: τα κόστη είναι τα διπλάσια.

Σύγκριση

Για να είναι η πρώτη διαδικασία πιο αποδοτική από την δεύτερη ως προς το Ω, αναζητούμε τις συνθήκες υπό τις οποίες ισχύει:

$$n1 \cdot n2 \cdot n3 + n1 \cdot n3 \cdot n4 < n2 \cdot n3 \cdot n4 + n1 \cdot n2 \cdot n4$$

Σημειώνεται ότι δεν λήφθηκαν στα παραπάνω υπόψιν οι προσθέσεις, οι οποίες ανήκουν προφανώς στο Ω, λόγω του ότι είναι περίπου ίδιο το πλήθος τους με τους πολλαπλασιασμούς με διαφορά +-1.

Αν όλα τα μητρώα είναι nxn δεν έχει διαφορά η σειρά πολλαπλασιασμών και τα κόστη είναι ακριβώς ίσα. Επομένως δεν ¨"τείνει" αλλά είναι ισότητα.

- Όταν όλες οι μεταβλητές είναι ίσες, οι παραπάνω ανισότητα τείνει να μετατραπεί σε ισότητα (το δοκίμασα σε πρόχειρο με συνδυασμό τιμών). Οι αλλαγές στα n1 και n3 έχουν άμεση επίδραση στον αριθμό των πράξεων που απαιτούνται στην αριστερή πλευρά της ανισότητας, κάνοντάς τους πιο ευαίσθητους στην ανισότητα σε σύγκριση με τις άλλες παραμέτρους (με άλλα λόγια, αν αυξήσω σημαντικά τα n1 και n3 σε σχέση με τα n2 και n4, η ανισότητα παύει να ισχύει.

5.) (Άσκηση A1.5.3 από GvL) Δίνεται $A \in R^{n \times n}$ που είναι αποθηκευμένο σε διάταξη ανά στήλη και ότι m = m1M και n = n1N. Θεωρούμε το A ως ένα σύνθετο μητρώο M επί N με μπλοκ m 1 επί n1. Να υλοποιήσετε έναν αλγόριθμο storeInBlocks(A, m1,n1,M,N) σε MATLAB για την αποθήκευση του A σε ένα διάνυσμα A.block(1:mn) με την ιδιότητα ότι κάθε μπλοκ A_{IJ} αποθηκεύεται συνεχόμενα στη διάταξη ανά στήλη. Εξηγήστε τα βήματα του αλγορίθμου και να δείξετε ότι εκτελεί σωστά τη διεργασία χρησιμοποιώντας τα δεδομένα:

```
m1 = 4; M = 2;
m=m1*M; n=m; N=M; n1=m1;
A1 = reshape ( [ 1:m*n ] ,m, n);
A1_block = storeInBlocks (A1, m1, n1 ,M, N);

m1=4; M=2; n1=3; N=3;
m = m1*M; n = n1*N;
A2 = reshape ( [ 1 : n*m] ,m, n) ;
A2_block = storeInBlocks (A2, m1, n1 ,M,N) ;
```

Επιβεβαιώστε ότι οι τιμές στα A?_block είναι σωστές!

Βήματα του αλγόριθμου

Αρχικοποίηση:

1.) Δημιουργούμε ένα διάνυσμα A_block το οποίο θα έχει μέγεθος mxn.

2.) Για κάθε μπλοκ (i,j) όπου i κυμαίνεται από 0 έως M−1 και j από 0 έως N−1, υπολογίζουμε τους δείκτες για το κάθε μπλοκ Aij ως εξής:

```
block_row = i * m1
block_col = j * n1
```

(Αφού προηγουμένως έχω κάνει εφαρμογή της reshape πάνω στη μήτρα A), προκειμένου να εξάγω το κάθε block διαστάσεων m1 x n1 γράφω την παρακάτω γραμμή κώδικα:

```
block = A(block_row + 1:block_row + m1,block_col + 1:block_col + n1);
```

Στη συνέχεια, εφόσον έχω πλέον εξάγει το κατάλληλων διαστάσεων block, οφείλω να το αποθηκεύσω κατά στήλες στις σωστές θέσεις σε ένα διάνυσμα στήλης A_block (τελικό διάνυσμα εξόδου). Αυτό μπορεί να επιτευχθεί με τη χρήση ενός δείκτη ο οποίος κάθε φορά θα ανανεώνεται με βάση την νέα αρχική θέση του κάθε block.

```
A_block(index:index + m1*n1 - 1)
```

Σε αυτές τις θέσεις λοιπόν αποθηκεύω το κάθε block κατά στήλες, με την εντολή block(:), η οποία θα μετατρέψει το μητρώο/block διαστάσεων m1 x n1 σε διάνυσμα στήλη με τις αρχικές στήλες ‘stacked’ τη μία πάνω στην άλλη. Η τελική εντολή άρα είναι:

```
A_block(index:index + m1*n1 - 1) = block(:);
```

Ενιαίος Κώδικας Συνάρτησης:

```
function A_block = storeInBlocks(A, m1, n1, M, N)

    [m, n] = size(A);
    A_block = zeros(m1 * n1 * M * N, 1);
    index = 1;                          % δείκτης για το διάνυσμα εξόδου A_block

    for i = 0:M-1
        for j = 0:N-1
            block_row = i * m1;
            block_col = j * n1;

            block = A(block_row+1:block_row+m1, block_col+1:block_col+n1);
            A_block(index:index + m1*n1 - 1) = block(:);
            index = index + m1 * n1;          % ανανέωση του δείκτη για το A_block

        end
    end
end
```

6.) Αναθέστε τα τελευταία 4 ψηφία του ΑΜ σας στη μεταβλητή ΑΜ σε περιβάλλον MATLAB και στη συνέχεια εκτελέστε τις παρακάτω εντολές:

```
 rng (AM) ;
p = randperm (14) ; id = p ( 1 : 4 ) ;
```

Στη συνέχεια δείτε την εικόνα/πίνακα με τίτλο "Μετρήσεις Ω και ρυθμού εκτέλεσης" του σετ 3 των διαλέξεων και αριθμώντας ως γραμμή υπ' αριθμ. 1 την δεύτερη (dot product), να επιλέξετε τις συναρτήσεις που αντιστοιχούν στις γραμμές που αντιστοιχούν στο id . Στη συνέχεια, ό,τι ζητάται στο ερώτημα αφορά αποκλειστικά αυτό το υποσύνολο συναρτήσεων.

α) Να χρονομετρήσετε με αξιόπιστο τρόπο τις κλήσεις για n=200 και να υπολογίσετε τα Gflops/s κάνοντας την υπόθεση εργασίας ότι τα Ω είναι ίδια με αυτά που εμφανίζονται στον πίνακα.

β) Βρείτε στη βιβλιογραφία (και να αναφέρετε ακριβώς από πού το βρήκατε), αν υπάρχει, την υπολογιστική πολυπλοκότητα για την συγκεκριμένη πράξη (π.χ. για το DOT είναι 2n.) Σχολιάστε τα αποτελέσματα (αν γνωρίζετε τη θεωρητική τιμή του Ω για γενικό n, θα ήταν καλό να το αναφέρετε.)

Κατάταξη κατά αύξουσα σειρά χρόνου:

chol(A) → 0.000652 < x=A\y → 0.000864 < matrix multiply → 0.000930 < cond(A)

→ 0.008211

Κατάταξη κατά φθίνουσα σειρά Gflops/s:

matrix multiply → 17.21 > x=A\y → 7.21 > chol(A) → 4.91 > cond(A) → 3.29

Πολυπλοκότητες Πράξεων:

x = A\y

Για γενικές πυκνές μήτρες, αναμένεται χρόνος πολυπλοκότητας $O(n^3)$ για τις άμεσες μεθόδους, ενώ οι επαναληπτικές μέθοδοι μπορούν να είναι πιο γρήγορες ανάλογα με την αραιότητα και την κατάσταση της μήτρας.

chol(A) → $(1/3)*n^3 + O(n^2)$

matrix multiply

Naive Method: $O(n^3)$
Strassen's Algorithm: $O(n^{2.81})$
Coppersmith-Winograd Algorithm: $O(n^{2.376})$

cond(A) → $O(n^3)$

Για τον υπολογισμό του δείκτη κατάστασης απαιτείται ο υπολογισμός της L2 νόρμας ($O(n^2)$) και η αντιστροφή του μητρώου ($O(n^3)$).

Σχόλια:

Το κόστος επίλυσης είναι μεγαλύτερο καθώς το κυρίαρχο κόστος της επίλυσης είναι η παραγοντοποίηση.

- *Ερμηνεία με βάση τους χρόνους εκτέλεσης:*
  Η παραγοντοποίηση Cholesky είναι η πιο γρήγορη από όλες τις μεθόδους. Αυτή η μέθοδος είναι εξαιρετικά αποδοτική για θετικά ορισμένες μήτρες, κάτι που φαίνεται από τον χαμηλό χρόνο εκτέλεσης. Η επίλυση του συστήματος γραμμικών εξισώσεων είναι ελαφρώς πιο αργή από την παραγοντοποίηση, αλλά παραμένει σε ικανοποιητικό επίπεδο. Αυτό υποδηλώνει (προσωπική μου εκτίμηση) ότι η μήτρα A έχει καλές ιδιότητες. Ο υπολογισμός του γινομένου μητρώων , αν και είναι γρηγορότερος από τον υπολογισμό του δείκτη κατάστασης, είναι λίγο πιο αργός από τις προηγούμενες μεθόδους. Η απόδοση είναι επαρκής και δείχνει την αποδοτικότητα των αλγορίθμων για πολλαπλασιασμό μητρώων. Τέλος, ο υπολογισμός του δείκτη κατάστασης είναι ο πιο αργός και αυτό είναι αναμενόμενο, καθώς περιλαμβάνει την αντιστροφή της μήτρας και τον υπολογισμό της L2 νόρμας.

- *Ερμηνεία με βάση τα Gflops/s:*
  Ο υπολογισμός του γινομένου μητρώων είναι ο πιο αποδοτικός, δείχνοντας την ικανότητα των αλγορίθμων να εκμεταλλεύονται τη υπολογιστική ισχύ. Η λύση του συστήματος γραμμικών εξισώσεων είναι επίσης αποδοτική, με ικανοποιητικά Gflops/s, επιβεβαιώνοντας την αποτελεσματικότητα της μεθόδου. Παρά την υψηλή ταχύτητα χρόνου εκτέλεσης, τα Gflops/s της παραγοντοποίησης Cholesky είναι χαμηλότερα από άλλες μεθόδους. Αυτό μπορεί να οφείλεται στη φύση των υπολογισμών που γίνονται. Τέλος, ο υπολογισμός του δείκτη κατάστασης έχει την πιο χαμηλή απόδοση σε Gflops/s, γεγονός που δείχνει τη δυσκολία και τον χρόνο που απαιτείται για τις διαδικασίες που εμπλέκονται.

Βιβλιογραφικές Πηγές:

Asher-Greif “Εισαγωγή στις Αριθμητικές Μεθόδους” σελ.202 για Cholesky και σελ.221 για δείκτη κατάστασης

https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

exercise 3

```
clear;
clc;

n = 10;
x = rand(n,1);
y = rand(n,1);
k = 10;
C = compute_matrix_power(x,y,k)
```

```
C = 10×10
    0.2461   23.1622    5.8039   28.6613   15.9882   27.5529   38.2726   24.0308 ···
    0.0717    6.7472    1.6907    8.3492    4.6574    8.0263   11.1490    7.0003
    0.4284   40.3249   10.1045   49.8988   27.8351   47.9689   66.6318   41.8372
    0.4437   41.7642   10.4651   51.6798   28.8286   49.6811   69.0100   43.3305
    0.0121    1.1415    0.2860    1.4125    0.7879    1.3579    1.8862    1.1843
    0.5003   47.0934   11.8005   58.2742   32.5072   56.0205   77.8159   48.8595
    0.3890   36.6184    9.1757   45.3123   25.2766   43.5598   60.5073   37.9917
    0.1091   10.2722    2.5740   12.7110    7.0906   12.2194   16.9734   10.6574
    0.2843   26.7614    6.7058   33.1150   18.4726   31.8343   44.2198   27.7650
    0.2590   24.3755    6.1079   30.1627   16.8257   28.9962   40.2775   25.2897
```

Το μητρώο αποτέλεσμα δεν χρειάζεται γιατί δεν αναδεικνύει κάτι ιδιαίτερο.

exercise 5

```
clear;
clc;


% Δεδομένα όπου το μητρώο A και τα block είναι τετραγωνικά
m1 = 4; M = 2;
m = m1 * M; n = m; N = M; n1 = m1;
A1 = reshape(1:m*n, m, n)
```

```
A1 = 8×8
     1     9    17    25    33    41    49    57
     2    10    18    26    34    42    50    58
     3    11    19    27    35    43    51    59
     4    12    20    28    36    44    52    60
     5    13    21    29    37    45    53    61
     6    14    22    30    38    46    54    62
     7    15    23    31    39    47    55    63
     8    16    24    32    40    48    56    64
```

```
A1_block = storeByBlocks(A1, m1, n1, M, N);

disp('A1_block:');
```

A1_block: Θα μπορούσες να τα δείξεις ως γραμμές.

```
disp(A1_block);
```

```
     1
     2
     3
     4
     9
    10
    11
```

```
    12
    17
    18
    19
    20
    25
    26
    27
    28
    33
    34
    35
    36
    41
    42
    43
    44
    49
    50
    51
    52
    57
    58
    59
    60
     5
     6
     7
     8
    13
    14
    15
    16
    21
    22
    23
    24
    29
    30
    31
    32
    37
    38
    39
    40
    45
    46
    47
    48
    53
    54
    55
    56
    61
    62
    63
    64
```

```
% Δεδομένα όπου το μητρώο A και τα block είναι ορθογώνια
m1 = 4; M = 2; n1 = 3; N = 3;
m = m1 * M; n = n1 * N;
A2 = reshape(1:n*m, m, n)
```

```
A2 = 8×9
     1     9    17    25    33    41    49    57    65
```

```
     2    10    18    26    34    42    50    58    66
     3    11    19    27    35    43    51    59    67
     4    12    20    28    36    44    52    60    68
     5    13    21    29    37    45    53    61    69
     6    14    22    30    38    46    54    62    70
     7    15    23    31    39    47    55    63    71
     8    16    24    32    40    48    56    64    72
```

```
A2_block = storeByBlocks(A2, m1, n1, M, N);

disp('A2_block:');
```

```
A2_block:
```

```
disp(A2_block);
```

```
     1
     2
     3
     4
     9
    10
    11
    12
    17
    18
    19
    20
    25
    26
    27
    28
    33
    34
    35
    36
    41
    42
    43
    44
    49
    50
    51
    52
    57
    58
    59
    60
    65
    66
    67
    68
     5
     6
     7
     8
    13
    14
    15
    16
    21
    22
    23
    24
    29
    30
    31
```

```
32
37
38
39
40
45
46
47
48
53
54
55
56
61
62
63
64
69
70
71
72
```

exercise 6

Πολύ καλή απάντηση.

```
clear;
clc;


AM = 3316;
rng(AM);
p = randperm(14);
id = p(1:4);
disp(id);       % 6 4 9 12
```

```
     6     4     9    12
```

```


% question a)

n = 200;
C = randn(n,n);
A = C'*C;         % το A είναι ΣΘΟ  ( δες Asher-Greif σελ. 203 Παράδειγμα
5.13)
y = rand(n,1);


%% Πράξη x = A\y
f_backslash = @() A\y;
time_backslash = timeit(f_backslash);

%% Πράξη chol(A)
f_chol = @() chol(A);
time_chol = timeit(f_chol);

%% Πράξη matrix multiply
```

```
B = rand(n);
f_matrix_multiply = @() A * B;
time_matrix_multiply = timeit(f_matrix_multiply);

% 4. Πράξη cond(A)
f_cond = @() cond(A);
time_cond = timeit(f_cond);

% Τιμές των Ω από τον πίνακα
omega_backslash = 6228144;
omega_chol = 3201127;
omega_matrix_multiply = 16000012;
omega_cond = 27000896;

% Υπολογισμός των Gflops/s
gflops_backslash = omega_backslash / (time_backslash * 1e9);
gflops_chol = omega_chol / (time_chol * 1e9);
gflops_matrix_multiply = omega_matrix_multiply / (time_matrix_multiply * 1e9);
gflops_cond = omega_cond / (time_cond * 1e9);

% Εμφάνιση αποτελεσμάτων
fprintf('Gflops/s για x = A\\y: %.2f\n', gflops_backslash);
```

```
Gflops/s για x = A\y: 10.34
```

```
fprintf('Gflops/s για chol(A): %.2f\n', gflops_chol);
```

```
Gflops/s για chol(A): 13.40
```

```
fprintf('Gflops/s για matrix multiply: %.2f\n', gflops_matrix_multiply);
```

```
Gflops/s για matrix multiply: 23.19
```

```
fprintf('Gflops/s για cond(A): %.2f\n', gflops_cond);
```

```
Gflops/s για cond(A): 5.71
```

```


fprintf('Χρόνος για x = A\\y: %f sec\n', time_backslash);
```

```
Χρόνος για x = A\y: 0.000602 sec
```

```
fprintf('Χρόνος για chol(A): %f sec\n', time_chol);
```

```
Χρόνος για chol(A): 0.000239 sec
```

```
fprintf('Χρόνος για matrix multiply: %f sec\n', time_matrix_multiply);
```

```
Χρόνος για matrix multiply: 0.000690 sec
```

```
fprintf('Χρόνος για cond(A): %f sec\n', time_cond);
```

```
Χρόνος για cond(A): 0.004732 sec
```

EXERCISE 3 FUNCTION DEFINITION

```
function C = compute_matrix_power(x, y, k)

    % x: Διάνυσμα στήλη μεγέθους n
    % y: Διάνυσμα στήλη μεγέθους n
    % k: Θετικός ακέραιος

    % Υπολογισμός του εξωτερικού γινομένου xyT ( O(n^2) )
    C = x * y';

    % Επαναληπτικός υπολογισμός της δύναμης του πίνακα C
    for i = 2:k
        C = C * (x * y');
    end
end
```

EXERCISE 5 FUNCTION DEFINITION

```
function A_block = storeByBlocks(A, m1, n1, M, N)
    % A: μητρώο A μεγέθους m x n
    % m1: πλήθος γραμμών του κάθε block
    % n1: πλήθος στηλών του κάθε block
    % M:  αριθμός block στις γραμμές του A
    % N: αριθμός block στις στήλες του A

    [m, n] = size(A);
    A_block = zeros(m1 * n1 * M * N, 1); % Αρχικοποιώ το διάνυσμα εξόδου/
αποτελέσματος
    index = 1;                           % Αρχικοποίηση δείκτη για το A_block

    for i = 0:M-1
        for j = 0:N-1
            % Υπολογίζω για κάθε block τους δείκτες αρχής γραμμής και
            % στήλης
            block_row = i * m1;
            block_col = j * n1;

            % Εξάγω το κάθε block
            block = A(block_row+1:block_row+m1, block_col+1:block_col+n1);

            % Και το αποθηκεύω column_wise στο τελικό διάνυσμα-στήλη A_block
            A_block(index:index + m1*n1 - 1) = block(:); % Column-wise αποθήκευση
            index = index + m1 * n1;                      % Ενημερώνω τον δείκτη
για το επόμενο block
        end
    end
end
```