εξαιρετική προσπάθεια και παρουσίαση για όσα προσπάθησες. Βλ. σχόλια

Κωνσταντίνος Αναστασόπουλος, 1093320, 4ο έτος. Λείπουν 7, 9β

Table of Contents

Υλικό και Λογισμικό

Σύστημα: Dell OptiPlex 3080

Επεξεργαστής: Intel(R) Core(TM) i3-10100 CPU @ 3.60GHz 3.60 GHz

Κύρια Μνήμη: 8.00 GB RAM

Πυρήνες: 4 Cores

Κρυφή Μνήμη Επιπέδου 1: L1 Cache 32KB/Core (16KB for instructions, 16KB for data) Total Size 128KB

Κρυφή Μνήμη Επιπέδου 2: L2 Cache 256KB/Core Total Size 1MB

Κρυφή Μνήμη Επιπέδου 3: L3 Cache Total Size 6MB

Λειτουργικό Σύστημα: Windows 11 Version 23H2 (OS Build 22631.4317)

Έκδοση MATLAB: MATLAB Version: 9.13.0.2698988 (R2022b) Update 10

Έκδοση BLAS: Intel(R) oneAPI Math Kernel Library Version 2021.3-Product Build 20210611 for Intel(R) 64 architecture applications (CNR branch AVX2)

Άσκηση 1

Το P χωρίζει το διάνυσμα σε δύο ομάδες και αναδιατάσσει τα στοιχεία τοποθετώντας τα με εναλλαγή, διαδοχικά, ένα από κάθε ομάδα. Θα είναι την μορφής P = [e1, e3, e5, e7, e2, e4, e6, e8].

```
x = (1:8)';
P = [1, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 1, 0, 0, 0;
     0, 1, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 1, 0, 0;
     0, 0, 1, 0, 0, 0, 0, 0;
```

```
     0, 0, 0, 0, 0, 0, 1, 0;
     0, 0, 0, 1, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 1];
y = P * x;

disp('Perfect Shuffle Matrix:');
```

```
Perfect Shuffle Matrix:
```

```
disp(P);
```

```
    1     0     0     0     0     0     0     0
    0     0     0     0     1     0     0     0
    0     1     0     0     0     0     0     0
    0     0     0     0     0     1     0     0
    0     0     1     0     0     0     0     0
    0     0     0     0     0     0     1     0
    0     0     0     1     0     0     0     0
    0     0     0     0     0     0     0     1
```

```
disp('Shuffled Vector:');
```

```
Shuffled Vector:
```

```
disp(y);
```

```
    1
    5
    2
    6
    3
    7
    4
    8
```

Άσκηση 2

Σωστό, φαίνεται από τον παρακάτω κώδικα.

```
P = [1, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 1, 0, 0, 0;
     0, 1, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 1, 0, 0;
     0, 0, 1, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 1, 0;
     0, 0, 0, 1, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 1];

I = P*P';
disp('Identity Matrix:');
```

```
Identity Matrix:
```

```
disp(I);
```

```
1   0   0   0   0   0   0   0
0   1   0   0   0   0   0   0
0   0   1   0   0   0   0   0
0   0   0   1   0   0   0   0
0   0   0   0   1   0   0   0
0   0   0   0   0   1   0   0
0   0   0   0   0   0   1   0
0   0   0   0   0   0   0   1
```

Άσκηση 3

Το μητρώο A έχει συμμετρική δομή και αποτελείται από δύο ζώνες, η καθεμία με 2 (υπο/υπερ)διαγωνίους. Ο πολλαπλασιασμός με P από αριστερά κάνει shuffle τις γραμμές του A, βάζοντας πρώτα τις περιττές και μετά τις άρτιες γραμμές και ο πολλαπλασιασμός από δεξιά με το P' (το οποίο κάνει unshuffle αν εφαρμοστεί από αριστερά) κάνει το unshuffling για τις στήλες. Το τελικό μητρώο είναι συμμετρικό αλλά έχει χάσει τη δομή με ζώνες που αρχικά είχε.

πχ παράδειγμα για το μητρώο της άσκησης 1:

```
n = 4;
B = zeros(n);
A = zeros(2*n, 2*n);

for i = 1:n
    B(i,i) = rand;
end

for i = 1:n-1
    B(i,i+1) = rand();
end

A(1:n, n+1:2*n) = B;
A(n+1:2*n, 1:n) = B';

P = [1, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 1, 0, 0, 0;
     0, 1, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 1, 0, 0;
     0, 0, 1, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 1, 0;
     0, 0, 0, 1, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 1];

A
```

```
A = 8×8
       0         0         0         0    0.8147    0.6324         0         0
       0         0         0         0         0    0.9058    0.0975         0
       0         0         0         0         0         0    0.1270    0.2785
       0         0         0         0         0         0         0    0.9134
```

```
    0.8147         0         0         0         0         0         0         0
    0.6324    0.9058         0         0         0         0         0         0
         0    0.0975    0.1270         0         0         0         0         0
         0         0    0.2785    0.9134         0         0         0         0
```

```
X = P*A*P'
```

```
X = 8×8
         0    0.8147         0    0.6324         0         0         0         0
    0.8147         0         0         0         0         0         0         0
         0         0         0    0.9058         0    0.0975         0         0
    0.6324         0    0.9058         0         0         0         0         0
         0         0         0         0         0    0.1270         0    0.2785
         0         0    0.0975         0    0.1270         0         0         0
         0         0         0         0         0         0         0    0.9134
         0         0         0         0    0.2785         0    0.9134         0
```

Άσκηση 4

Το Kronecker y με x κάνει πολλαπλασιασμό κάθε συνιστώσας του y με όλο το x. Το γινόμενο x επί y' πολλαπλασιάζει όλο το x με κάθε συνιστώσα του y και διατάσσεται ανά στήλες. Το vec(A) φέρνει τις στήλες μία κάτω από την άλλη και το αποτέλεσμα είναι ίδιο με του Kronecker. Φαίνεται και με τρέξιμο του παρακάτω κώδικα.

```
n = input('Δώσε διάσταση για το n: ');
x = rand(n, 1);
y = rand(n, 1);
A = x*y';
k = kron(y, x)
```

```
k = 9×1
    0.0862
    0.1509
    0.1521
    0.5308
    0.9293
    0.9365
    0.5235
    0.9165
    0.9236
```

```
a = A(:)
```

```
a = 9×1
    0.0862
    0.1509
    0.1521
    0.5308
    0.9293
    0.9365
    0.5235
    0.9165
    0.9236
```

Άσκηση 5

Έστω B είναι m1 x n1 και C είναι m2 x n2. Σύμφωνα με το GVL, τα P και Q είναι τα perfect shuffle μητρώα για shuffle m1, m2 και n1, n2 αντίστοιχα. Το P κάνει shuffle τις γραμμές του γινομένου Kronecker των B, C βάζοντας μία από κάθε μία από m2 ομάδες, m1 φορές. Το Q' κάνει κάτι ανάλογο για τις στήλες. Το P*kron(B, C)*Q' τελικά ισούται με kron(C, B). Φαίνεται και στο παρακάτω παράδειγμα.

```
P = [1, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 1, 0, 0, 0;
     0, 1, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 1, 0, 0;
     0, 0, 1, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 1, 0;
     0, 0, 0, 1, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 1];

Q = [1, 0, 0, 0, 0, 0;
     0, 0, 0, 1, 0, 0;
     0, 1, 0, 0, 0, 0;
     0, 0, 0, 0, 1, 0;
     0, 0, 1, 0, 0, 0;
     0, 0, 0, 0, 0, 1];

B = rand(2, 2);
C = rand(4, 3);

permuted_kron = P*kron(B, C)*Q'
```

```
permuted_kron = 8×6
    0.4445    0.1299    0.0173    0.0051    0.3678    0.1075
    0.7328    0.3862    0.0286    0.0151    0.6064    0.3196
    0.3845    0.1124    0.4121    0.1205    0.3607    0.1054
    0.6340    0.3341    0.6795    0.3581    0.5947    0.3134
    0.4657    0.1361    0.4533    0.1325    0.1904    0.0557
    0.7679    0.4047    0.7475    0.3939    0.3139    0.1654
    0.3183    0.0930    0.3294    0.0963    0.3182    0.0930
    0.5248    0.2766    0.5432    0.2863    0.5246    0.2765
```

```
reverse_kron = kron(C, B)
```

```
reverse_kron = 8×6
    0.4445    0.1299    0.0173    0.0051    0.3678    0.1075
    0.7328    0.3862    0.0286    0.0151    0.6064    0.3196
    0.3845    0.1124    0.4121    0.1205    0.3607    0.1054
    0.6340    0.3341    0.6795    0.3581    0.5947    0.3134
    0.4657    0.1361    0.4533    0.1325    0.1904    0.0557
    0.7679    0.4047    0.7475    0.3939    0.3139    0.1654
    0.3183    0.0930    0.3294    0.0963    0.3182    0.0930
    0.5248    0.2766    0.5432    0.2863    0.5246    0.2765
```

Άσκηση 6

Επειδή το γινόμενο Kronecker πολλαπλασιάζει κάθε στοιχείο του B με όλο το C, μπορούμε για κάθε υπομητρώο να υπολογίζουμε το συγκεκριμένο στοιχείο β επί το xi * C * xj', για τα κατάλληλα xi και xj. Φαίνεται από τον παρακάτω κώδικα. Το kron και το διπλό άθροισμα σε for δίνουν το ίδιο αποτέλεσμα.

```
p = input('Δώσε διάσταση B: ');
q = input('Δώσε διάσταση C: ');
B = rand(p);
C = rand(q);
x = rand(q*p, 1);

sum_result = 0;
for i = 1:p
    for j = 1:p
        sum_result = sum_result + B(i,j) * (x(1+q*(i-1):q*i)' * C *
x(1+q*(j-1):q*j));
    end
end

sum_result
```

```
sum_result = 3.2946
```

```
kron_result = x'*kron(B, C)*x
```

```
kron_result = 3.2946
```

Άσκηση 8

α)

function A = Hambuild(x, y)

n = length(x);

Βλ. γενικά σχόλια

I = eye(n);

A = [I, x*y'; y*x', -I];

end

β)

```
n = 2.^[6:12];
time_taken = zeros(1, length(n));

for i = 1:length(n)
    x = rand(n(i), 1);
    y = rand(n(i), 1);
    B = rand(2*n(i), 1);
```

```
    A = Hambuild(x, y);
    f = @() mtimes(A, B);
    time_taken(i) = timeit(f);
end

disp(time_taken);
```

Άσκηση 9

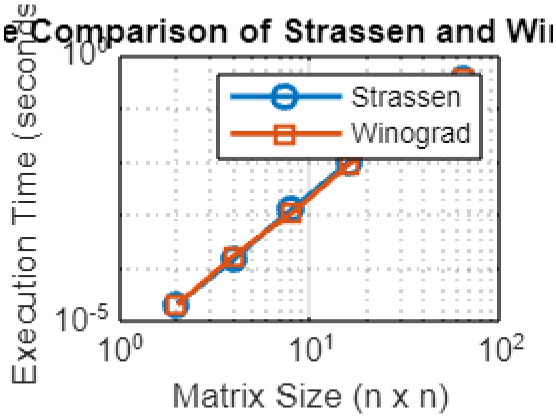
α)

Έστω A1 (**nxm**), A2 (**kxl**), A3 (**pxq**). Τότε το γινόμενο Kronecker των A1, A2 και A3 έχει διάσταση **nkpxmlq**. Προφανώς για να έχει νόημα το γινόμενο του μητρώου αυτού με διάνυσμα x, το διάνυσμα αυτό πρέπει να έχει διάσταση **mlq**x**1**.

Άσκηση 10

```
A = rand(8);
B = rand(8);

matrix_sizes = 2.^(1:6);
time_strassen = zeros(1, length(matrix_sizes));
time_winograd = zeros(1, length(matrix_sizes));

for i = 1:length(matrix_sizes)
    n = matrix_sizes(i);
    A = rand(n);
    B = rand(n);

    time_strassen(i) = timeit(@() strassen(A, B));

    time_winograd(i) = timeit(@() winograd(A, B));
end

figure;
plot(matrix_sizes, time_strassen, '-o', 'LineWidth', 1.5, 'DisplayName',
'Strassen');
hold on;
plot(matrix_sizes, time_winograd, '-s', 'LineWidth', 1.5, 'DisplayName',
'Winograd');
hold off;

title('Execution Time Comparison of Strassen and Winograd Algorithms');
xlabel('Matrix Size (n x n)');
ylabel('Execution Time (seconds)');
```

```
legend('show');
grid on;
set(gca, 'XScale', 'log', 'YScale', 'log');
```



Θα μπορούσες να χρονομετρήσεις για μεγαλύτερα μητρώα - ξ διαφορά επίδοσης θα ήταν πιο εμφανής.

Functions

Άσκηση 8

```
function A = Hambuild(x, y)
    n = length(x);
    I = eye(n);
    A = [I, x*y'; y*x', -I];
end
```

Άσκηση 10

```
function C = strassen(A, B)

    [n, m] = size(A);
    [k, l] = size(B);

    if n ~= m || k ~= l || n ~= k
        error('The matrices should be square and of the same size.');
    end

    if n == 1
        C = A * B;
    else
        half = n / 2;

        A11 = A(1:half, 1:half);
        A12 = A(1:half, half+1:end);
        A21 = A(half+1:end, 1:half);
        A22 = A(half+1:end, half+1:end);

        B11 = B(1:half, 1:half);
```

```
        B12 = B(1:half, half+1:end);
        B21 = B(half+1:end, 1:half);
        B22 = B(half+1:end, half+1:end);

        S1 = A12 - A22;
        S2 = A11 + A22;
        S3 = A21 - A11;
        S4 = A11 + A12;
        S5 = A21 + A22;
        S6 = B21 + B22;
        S7 = B11 + B12;
        S8 = B21 - B11;
        S9 = B11 + B22;
        S10 = B12 - B22;

        P1 = strassen(S1, S6);
        P2 = strassen(S2, S9);
        P3 = strassen(S3, S7);
        P4 = strassen(S4, B22);
        P5 = strassen(A11, S10);
        P6 = strassen(A22, S8);
        P7 = strassen(S5, B11);

        C11 = P1 + P2 - P4 + P6;
        C12 = P4 + P5;
        C21 = P6 + P7;
        C22 = P2 + P3 + P5 - P7;
        C = [C11, C12; C21, C22];
    end
end

function C = winograd(A, B)

    [n, m] = size(A);
    [k, l] = size(B);

    if n ~= m || k ~= l || n ~= k
        error('The matrices should be square and of the same size.');
    end

    if n == 1
        C = A * B;
    else
        half = n / 2;

        A11 = A(1:half, 1:half);
        A12 = A(1:half, half+1:end);
        A21 = A(half+1:end, 1:half);
        A22 = A(half+1:end, half+1:end);

        B11 = B(1:half, 1:half);
        B12 = B(1:half, half+1:end);
        B21 = B(half+1:end, 1:half);
        B22 = B(half+1:end, half+1:end);
```

```
        S1 = A21 + A22;
        S2 = S1 - A11;
        S3 = A11 - A21;
        S4 = A12 - S2;
        S5 = B12 - B11;
        S6 = B22 - S5;
        S7 = B22 - B12;
        S8 = S6 - B21;

        M1 = winograd(S2, S6);
        M2 = winograd(A11, B11);
        M3 = winograd(A12, B21);
        M4 = winograd(S3, S7);
        M5 = winograd(S1, S5);
        M6 = winograd(S4, B22);
        M7 = winograd(A22, S8);

        T1 = M1 + M2;
        T2 = T1 + M4;

        C11 = M2 + M3;
        C12 = T1 + M5 + M6;
        C21 = T2 - M7;
        C22 = T2 + M5;
        C = [C11, C12; C21, C22];
    end
end
```

ΑΠΑΝΤΗΣΗ: Εξετάζουμε τους όρους για $k = 2, 3$. Προσοχή - η γνωστή έκφραση για το $(a+b)^k$ δεν μπορεί να χρησιμοποιηθεί καθώς δεν ισχύει η μεταθετική ιδιότητα για τον πολλαπλασιασμό μητρώων, δηλ. $Auv^\top \neq uv^\top A$.

Αν $k = 2$ και θέσουμε E_2 το μητρώο αναστροφής, δηλ. το αντιταυτοτικό μητρώο $E_2 = [0, 1; 1, 0]$ θα έχουμε:

$$\begin{aligned}(A+uv^\top)^2 &= A^2 + Auv^\top + uv^\top A + (v^\top u)uv^\top \\ &= A^2 + \\ &= A^2 + [u, Au][(A^\top + vu^\top)v, v]^\top\end{aligned}$$

Ισχυριζόμαστε ότι

$$\begin{aligned}(A+uv^\top)^k &= A^k + U_k V_k^\top, \text{ όπου} \\ U_k &= [u, Au, ..., A^{k-1}u]E_k, V_k = [v, (A^\top + vu^\top)v, ..., (A^\top + vu^\top)^{k-1}v]\end{aligned}$$

Είναι προφανώς σωστό αν $k = 1$ και έστω ότι ισχύει για $k = 2, ..., s$. Τότε

$$\begin{aligned}(A+uv^\top)^{s+1} &= (A+uv^\top)(A+uv^\top)^s = A(A^s + U_s V_s^\top) + uv^\top(A+uv^\top)^s \\ &= A^{s+1} + A[u, Au, ..., A^{s-1}u]E_s V_s^\top + uv^\top(A+uv^\top)^s\end{aligned}$$

Επικεντρωνόμενοι στους τελευταίους 4 όρους:

$$\begin{aligned}&= [Au, A^2u, ..., A^s u]E_s V^\top + uv^\top(A+uv^\top)^s \\ &= [u, Au, ..., A^s u]E_{s+1}[v, (A^\top + vu^\top)v, ..., (A^\top + vu^\top)^s v]^\top \\ &= U_{s+1}V_{s+1}^\top, \text{ αποδεικνύοντας το ζητούμενο.}\end{aligned}$$

Στη συνέχεια γίνεται η μέτρηση του κοστους υπολογισμού των παραπάνω. Το βασικό βέβαια (που δεν αναφέρεται στην εκφώνηση) είναι ότι για να έχει νόημα η χρήση του τύπου, θα πρέπει το A να έχει κάποια ειδική δομή που επιτρέπει τον ταχύ υπολογισμό του γινομένου του με διάνυσμα, π.χ με κόστος $O(p(n))$ όπου $p(n) = O(n)$ ή $O(n \log n)$ αντί για $O(n^2)$. Πρόκειται για περιπτώσεις αντίστοιχες με εκείνες της χρήσης του τύπου SMW. Ο τύπος που αποδείξαμε έχει ιδιαίτερο ενδιαφέρον όχι τόσο για τον ρητό υπολογισμό του μητρώου $(A^k + UV^\top)$ αλλά για την κατασκευή των όρων U, V έτσι ώστε να είναι διαθέσιμοι για την εφαρμογή του $(A^k + UV^\top)$ σε διάνυσμα, κ.λπ. Επιπλέον, αντί για επανειλημμένη χρήση BLAS-1, αυτό μπορεί να γίνει με BLAS-2. Ο υπολογισμός των U, V μπορεί να γίνει αναδρομικά. Π.χ. για $k = 2$, υπολογίζονται τα

I) $U_2 = [u, Au]$

II) $v^\top A + (v^\top u)v^\top = v^\top(A + uv^\top)$ και άρα $V_2 = [v, (A^\top + vu^\top)v]$.

με συνολικό κόστος $O(p(n))$. Για $k = 3$ προκύπτει εύκολα ότι αν έχουμε ήδη υπολογίσει τα U_2, V_2, τότε με επιπλέον κόστος $O(p(n))$ υπολογίζονται οι επιπλέον όροι για τα U_3, V_3, κ.ο.κ. Το συνολικό κόστος για τους όρους U_k, V_k επομένως είναι $\Omega = O(kp(n))$. Ανάλογα με την περίπτωση, μπορεί να χρειάζεται και ο υπολογισμός του A^k.