Τρίτη Άσκηση
στον Επιστημονικό Υπολογισμό

Κων/νος Αραβανής ΑΜ: *3628*
mail: *arabanis@ceid.upatras.gr*

20 Φεβρουαρίου 2009

1 Χαρακτηριστηκά Συστήματος

Αρχικά θα περιγράψουμε το σύστημα στο οποίο τρέξαμε τα πειράματα μας:
Το σύστημα μας είχε το λειτουργικό Linux/Debian Lenny και ο επεξεργαστής του ήταν ένας Intel Core Duo T2300 στα 1,66GHz. Είχαμε μία ιεραρχεία μνήμης δύο επιπέδων με το πρώτο επίπεδο να αποτελείτε από δύο κρυφές μνήμες, η μία για data (L1 D-Cache) και η άλλη για inst. (L1 D-Cache) και οι δύο των 32KBytes και με τα ακόλουθα χαρακτηριστικά: 8 way set-associative, 64-byte line size. Η κύρια μνήμη (L2 Cache) μας είχε μέγεθος 2048KBytes και ακριβώς τα ίδια χαρακτηριστικά με τις κρυφές. Όσον αφορά το λογισμικό στο οποίο εκτελέστηκαν τα προγράμματα μας ήταν η Matlab $7.6.0(R2008a)$. Να σημειωθεί ότι η αναφορά έγινε σε **LaTeX**.

2 Παραγοντοποίηση QR με Householder και ελάχιστα τετράγωνα

2.1 Κώδικας Παραγοντοποίσης QR

Στο σημείο αυτό μας ζητήθηκε να κατασκευάσουμε κατάλληλη συνάρτηση που να παραγοντοποιεί κατα QR με Householder και ελάχιστα τετράγωνα χρησιμοποιόντας τους αλγόριθμους που μάθαμε από το μάθημα. Συγκεκριμάνα ο κώδικας μας θα πρέπει να επιστρέφει στο κάτω τριγωνικό τμήμα του A ότι χρειάζεται από το διάνυσμα Householder και στο άνω τριγωνικό τμήμα του A το R.

Ο κώδικας που συντάξαμε για αυτό το ερώτημα είναι ο ακόλουθος[1]:

[1] Όσες συναρτήσεις δεν είναι γνωστές από τη Matlab μπορείτε να τις βρείτε στο παράρτημα.

Program 1 QR.m

```
function A = QR(A)
% QR is a function that QR-factorize with Householder a matrix A and
% returns its factorization.

% find the size of the A matrix
size_A = size(A);
% m is the mumber of rows of A matrix
m = size_A(1, 1);
% n is the number of columns of A matrix
n = size_A(1, 2);

for j = 1: n
    u(j : m) = REFL( A(j : m, j) );

    A(j : m, j : n) = REFLROW(A(j : m, j : n), u(j : m)');

    if j < m
        A(j + 1 : m, j) = u(j + 1 : m)';
    end
end
```

2.2 Ελαχιστοποίηση του $\|AX - B\|_{\mathbf{F}}$

Στο σημείο αυτό μας ζητήθηκε να κατασκευάσουμε κατάλληλο αλγόριθμο που δοθέντος του $A \in \Re^{mxn}$ (θεωρούμε ότι $m \succeq n, s$) να επιστρέφει τη λύση $X \in \Re^{mxs}$ που ελαχιτοποιεί το $\| AX - B \|_F$ (νόρμα Frobenius).

Ο κώδικας που συντάξαμε για αυτό το ερώτημα είναι ο ακόλουθος[2]:

[2]Όσες συναρτήσεις δεν είναι γνωστές από τη Matlab μπορείτε να τις βρείτε στο παράρτημα.

Program 2 askisi_1_2.m

```
function X = askisi_1_2(A, b)
% askisi_1_2 is a function that takes as arguments the matrixes A and b and
% returns the solution of the problem that minimize the ||A*X - b||(F)
% (Forbenius norm)

% find the size of the A matrix
size_A = size(A);
% m is the mumber of rows of A matrix
m = size_A(1, 1);
% n is the number of columns of A matrix
n = size_A(1, 2);

% find the size of the b matrix
size_b = size(b);
% s is the number of columns of b matrix
s = size_b(1, 2);

% QR factorization of matrix A
A = QR(A);

% find the U matrix and the R matrix
[Q,R,U] = find_Q_R_U(A);


for i = 1: s
    temp = b(:,i);
    for j = 1: n
        temp(j : m, 1) = REFLROW( temp(j : m, 1), U(j : m,j) );
    end

    X(:,i) = R(1 : n, 1 : n) \ temp(1 : n, 1);
end

end
```

2.3 Κώδικας παραγωγής του αντίστροφου αποτελέσματος της Παραγοντοποίσης QR

Στο σημείο αυτό μας ζητήθηκε να κατασκευάσουμε τον 'αντίστροφο' αλγόριθμο που να επιτυγχάνει το αντίθετο αποτέλεσμα από αυτό του αλγορίθμου στην ενότητα 2.1. Δηλαδή, δοθέντος του αποτελέσματος της QR της ενότητας 2.11 με τ διανύσματα Householder και το R αποθηκευμένα όπως πριν, να υπολογίζει και να επιστρέφει το αντίστοιχο μητρώο A.

Για τον υπολογισμό του $QR = H_1H_2H_3...H_nR = A$ χρησιμοποιήσαε την REFLROW μιάς και απαιτεί λιγότερες πράξεις.

Ο κώδικας που συντάξαμε για αυτό το ερώτημα είναι ο ακόλουθος[3]:

[3]Όσες συναρτήσεις δεν είναι γνωστές από τη Matlab μπορείτε να τις βρείτε στο παράρτημα.

Program 3 reversed_QR.m

```
function B = reversed_QR(A)
% reversed_QR is the "reversed" function of QR. It takes the QR-factorized
% A matrix and returns it without being QR factorized processed.

% find the size of the A matrix
size_A = size(A);
% m is the mumber of rows of A matrix
m = size_A(1, 1);
% n is the number of columns of A matrix
n = size_A(1, 2);

% the R matrix is the up-triangular part of A
R = A;

% we zero the down-triangular part of R (== A) matrix
for i = 1: m
    for j = 1: n
        if i > j
            R(i, j) = 0;
        end
    end
end

% find the A matrix without the QR factorization process, by using the
% reflrow for the multiplications in order to make less multiplies.
B = REFLROW(R, [ zeros(n-1,1) ; 1 ; A(n+1:m,n) ] );

for i = n-1 : -1 : 1
    u = [ zeros(i-1, 1) ; 1 ; A(i+1:m, i) ];
    B = REFLROW(B, u);
end

end
```

2.4 Πειραματική Διαδικασία

Κατά την πειραματική διαδικασία μας ζητήθηκε να επιλέξουμε 4 μητρώα (για τα οποία $n \succeq 20$) και να επαληθεύσουμε τον κάθε κώδικα με την παρακάτω διαδικασία:

1. *για την QR*: $\|A - QR\|_F$
2. *για την reversed_QR*: $\|A - \hat{A}\|_F$ (όπου $\hat{A}$ είναι το υπολογισμένο A)
3. *για τα συστήματα*: επαληθεύοντας την τιμή του $\|B - AX\|_F$ ώστε να είναι περίπου ίδια με την τιμή του $\|B - A(A \setminus B)\|_F$

Τα 4 μητρώα που επιλέξαμε είναι τα ακόλουθα:

- *A1 = rand(25, 20)*: Το μητρώο αυτό το επιλέξαμε γιατί είναι η οριακή περίτωση, καθώς είναι πολύ μικρό σε σχέση με αυτά που αναένετε να χρησιμοποιηθούν και περιμένουμε τα σφάλματα που θα λάβουν να είναι πάρα πολύ μικρά.

- *A2 = rand(100, 75)*: Αντίθετα αυτό είναι ένα σχετικά μεγάλο μητρώο που σε σύγκριση με τα αποτελέσματα από το προηγούμενο μητρώο θα μπορέσουμε να κατανοήσουμε την συμπεριφορά των αλγορίθμων μας σε σχέση με την διακύμανση του μεγέθους.

- *A3 = toeplitz([1; rand(99, 1)], [1, rand(1, 74)])*: Το toeplitz το πήραμε λόγω του ότι είναι ένα απο τα πιο συνηθισμένα μητρώα σε πολλές εφαρμογές και θα θέλαμε να μελετήσουμε για αυτό το λόγο τη συμπεριφορά του. Και για να το συγκρίνουμε με τα άλλα επιλεξαε να έχει το ίδιο μέγεθος με το *A2*.

- *A4 = vand(100,rand(75, 1))*: Το μητρώο vandermonde το συμπεριλάβαμε στις μετρήσεις μας καθως αποτελεί μία αρκετά ιδιαίτερη περίπτωση μητρώου πράγμα που το κάνει αρκετά ενδιαφέρον στη μελέτη της συμπεριφοράς του. Τα vandermonde μητρώα έχουν πολύ μεγάλο δείκτη κατάστασης με αποτέλεσμα οι πραξεις στις οποίες χρησιμοποιείται να παράγει μεγάλα σφάλματα. Τις διαστάσεις και αυτού του μητρώου τις επιλέξαμε για τον ίδιο ακριβώς λόγο όπως και παραπάνω.

Συνοπτικά τα αποτελέσατα που λάβαμε φαίνονται στον πίνακα 1.

	$\|A - QR\|_F$	$\|A - \hat{A}\|_F$	$abs(\|B - AX\|_F - \|B - A(A \setminus B)\|_F)$
A1	5.5958e-15	6.8362e-15	4.4409e-16
A2	5.5491e-14	5.7246e-14	5.3291e-15
A3	4.6193e-14	4.0260e-14	1.7764e-15
A4	1.8773e-14	2.0028e-14	4.4786e+05

Πίνακας 1: Αποτελέσματα πειράματος

Από τα παραπάνω εύκολα μπορούμε να δούμε ότι όσο αυξάνεται το μέγεθος των διαστάσεων του μητρώου τόσο αυξάνονται και τα διάφορα σφάλματα που συμβαίνουν πράγμα αναμενόμενο λόγω του ότι αυξάνεται και η πολυπλοκότητα. Επίσης μπορούμε να δούμε ότι τα σφάλματα όταν χρησιμοποιούμε μητρώα toeplitz σχεδόν δεκαπλασιάζονται, αλλά συνεχίζουν να παραμένουν υπερβολικά μικρά με αποτέλεσμα να μην ενοχλούμαστε (φυσικά αυτό παιζει βέβαια και ρόλο και από την ανοχή που πρέπει να δείξουμε, αν και η τάξη του σφάλματος είναι 10^14 μικρότερη από τα δεδομένα των πινάκων μας). Σχετικά με το vandermonde μητρώο

παρατηρούμε ότι τα σφάλματα για την QRκαι την reversed_QR παραμένουν πολύ μικρά αλλά για την Ελαχιστοποίηση του $\|AX - B\|_{\mathbf{F}}$ είναι πολύ ψηλά (όπως είπαμε άλλοστε και παραπάνω: Τα vandermonde μητρώα έχουν πολύ μεγάλο δείκτη κατάστασης με αποτέλεσμα οι πραξεις στις οποίες χρησιμοποιείται να παράγει μεγάλα σφάλματα - για τα οποία μας ενημέρωσε και η Matlab με αρκετά *Warnings*). Τέλος να πούμε ότι από τα παραπάνω καταλήξαμε στο ότι ο αλγόριθμος QR και ο reversed_QR που φτιάξαμε, ανεξάρτητα από το είδος του μητρώου που τους δώσουμε σαν είσοδο παράγουν ορθά αποτελέσματα, ενώ ο αλγόριθμος που φτιάξαμε για την ελαχιστοποίηση του $\|AX - B\|_{\mathbf{F}}$ είναι σωστός μόνο για μητρώα με μικρό δείκτη κατάστασης. Αντίθετα για αυτά με μεγάλο δείκτη κατάστασης παράγει αποτελέσματα με τεράστια σφάλματα.

Ο κώδικας που χρησιμοποιήθηκε για τα παραπάνω φαίνεται παρακάτω[4]:

Program 4 askisi1_script.m

```
% A1 martix
A1 = rand(25, 20);
[a1,b1,c1] = prove(A1)

% A2 martix
A2 = rand(100, 75);
[a2,b2,c2] = prove(A2)

% A3 martix
A3 = toeplitz([ 1; rand(99, 1) ], [ 1, rand(1, 74) ]);
[a3,b3,c3] = prove(A3)

% A4 martix
A4 = vand(100,rand(75, 1));
[a4,b4,c4] = prove(A4)
```

3 Παράρτημα

Στο παράρτημα παραθέτουμε τις συναρτήσεις που χρησιμποιήθηκαν παραπάνω από άλλες συναρτήσεις και ο κώδικας τους δεν φαινόταν.

[4] Όσες συναρτήσεις δεν είναι γνωστές από τη Matlab μπορείτε να τις βρείτε στο παράρτημα.

Program 5 REFL.m

```
function u = REFL(x)
% REFL is a function that takes the x arguent - vector and returns the
% Householder Vector

% n is the size of x vector
n = length(x);

% m is the norm2 of vector x a.k.a. the Euclidean length
m = norm(x, 2);

u = x;

% if the Euclidean length of vector x isn't zero
if m ~= 0
    b = x(1) + sign( x(1) ) * m;
    u(2 : n) = u(2 : n) / b;
end

u(1) = 1;
end
```

Program 6 REFLROW.m

```
function B = REFLROW(A, u)
% REFLROW function: A <- H * A

b = -2 / (u' * u);
w = b * A' * u;
B = A + u * w';
end
```

Program 7 find_Q_R_U.m

```
function [Q, R, U] = find_Q_R_U(A)
% find_Q_R_U is a function that takes as an argument that QR-factorized A
% matrix and returns the R, Q and U matrixes.

% find the size of the A matrix
size_A = size(A);
% m is the mumber of rows of A matrix
m = size_A(1, 1);
% n is the number of columns of A matrix
n = size_A(1, 2);

% the R matrix is the up-triangular part of A
R = A;

% we zero the down-triangular part of R (== A) matrix
for i = 1: m
    for j = 1: n
        if i > j
            R(i, j) = 0;
        end
    end
end

% the U matrix is the down-triangular part of A with ones in its diagonal
U = A - R + eye(m, n);

% I is a matrix with ones
I = eye(m, m);

% in this part we find the Q matrix by processing the U matrix
Q = I - 2 * U(:, 1) * U(:, 1)' / (U(:, 1)' * U(:, 1));

for i = 2: n
    H = I - 2 * U(:, i) * U(:, i)' / (U(:, i)' * U(:, i));
    Q = Q * H;
end

end
```

Program 8 vand.m from Matrix Computation Toolbox

```
function V = vand(m, p)
%VAND   Vandermonde matrix.
%       V = VAND(P), where P is a vector, produces the (primal)
%       Vandermonde matrix based on the points P, i.e. V(i,j) = P(j)^(i-1).
%       VAND(M,P) is a rectangular version of VAND(P) with M rows.
%       Special case: If P is a scalar then P equally spaced points on [0,1]
%                     are used.

%       Reference:
%       N. J. Higham, Accuracy and Stability of Numerical Algorithms,
%       Second edition, Society for Industrial and Applied Mathematics,
%       Philadelphia, PA, 2002; chap. 22.

if nargin == 1, p = m; end
n = length(p);

%  Handle scalar p.
if n == 1
   n = p;
   p = linspace(0,1,n);
end

if nargin == 1, m = n; end

p = p(:).';                    % Ensure p is a row vector.
V = ones(m,n);
for i=2:m
    V(i,:) = p.*V(i-1,:);
end
```

Program 9 prove.m

```
function [prf_QR, prf_reversed_QR, prf_askisi_1_2] = prove(A)
%function that is used to prove the rightness of the algorithms QR,
%reversed_QR and askisi_1_2

% proving the QR function
A_qr = QR(A);
[Q,R,U] = find_Q_R_U(A_qr);
prf_QR = norm(A - Q*R, 'fro');

% proving the reversed_QR function
A_computed = reversed_QR(A_qr);
prf_reversed_QR = norm(A - A_computed, 'fro');


% proving the askisi_1_2 function

% find the size of the A matrix
size_A = size(A);

% m is the mumber of rows of A matrix
m = size_A(1, 1);

B = rand(m);
X = askisi_1_2(A,B);
prf_askisi_1_2 = abs(norm(B - A*X, 'fro') - norm(B - A * (A \ B), 'fro'));

end
```