

CS594 - Homework # 10

(due April 26th)

1 Introduction, Motivation, Recall

This homework is on the study of the Gram-Schmidt algorithm and its variants. The Gram-Schmidt algorithm is an orthogonalization algorithm used in most of the nonsymmetric iterative methods (solving a linear system or eigenproblem). Starting from a nonsingular m -by- n matrix A , the goal is to produce its QR -factorization.

The Gram-Schmidt algorithm is suitable when the next column of the matrix A (or block of columns) depends on the previous columns of the matrix Q (think Arnoldi, where A is (v_0, AV_j) and $Q = V_{j+1}$).

The three main costs of an iterative methods are preconditioning, matrix-vector products and orthogonalization. The Gram-Schmidt algorithm plays a role in the latter. The cost of the orthogonalization might be nonnegligible in some iterative methods.

In this homework, we are first concerned with numerical issues. A short study in Matlab is asked. Then we are concerned with scalability issues. The goal is to compare two Gram-Schmidt algorithm, namely modified Gram-Schmidt (MGS) and classical Gram-Schmidt (CGS).

Below is three matlab functions to explain what are MGS and CGS. Just copy-paste the verbatim in a file call test_cgs_mgs.m and run it in Matlab.

```
function test_cgs_mgs
    m=100; n=20;
    A=randn(m,n);
    [Q1,R1]=cgs(A);
    [Q2,R2]=mgs(A);
    fprintf('cgs: norm(A-Q*R)/norm(A) = %2.2e\n',norm(A-Q1*R1)/norm(A))
    fprintf('      norm(I-Q'*Q)          = %2.2e\n',norm(eye(n)-Q1'*Q1))
    fprintf('mgs: norm(A-Q*R)/norm(A) = %2.2e\n',norm(A-Q2*R2)/norm(A))
    fprintf('      norm(I-Q'*Q)          = %2.2e\n',norm(eye(n)-Q2'*Q2))
end
function [Q,R]=mgs(A)
    [m,n]=size(A);
    Q = A;
    R = zeros(n);
    for j=1:n,
        for i=1:j-1,
            R(i,j) = Q(:,i)'*Q(:,j);
            Q(:,j) = Q(:,j)-Q(:,i)*R(i,j);
        end
    end
end
```

```

        end
        R(j,j) = norm(Q(:,j));
        Q(:,j) = Q(:,j)/R(j,j);
    end
end
function [Q,R]=cgs(A)
    [m,n]=size(A);
    Q = A;
    R = zeros(n);
    for j=1:n,
        R(1:j-1,j) = Q(:,1:j-1)'*Q(:,j);
        Q(:,j) = Q(:,j)-Q(:,1:j-1)*R(1:j-1,j);
        R(j,j) = norm(Q(:,j));
        Q(:,j) = Q(:,j)/R(j,j);
    end
end
end

```

2 Stability

Make a demo code that shows that the loss of orthogonality of Q ($\|I - Q^T Q\|_2$) depends on the condition number of the matrix A ($\kappa(A)$). Any language accepted. (I.e. Matlab is fine). What seems to be the link between $\kappa(A)$ and $\|I - Q^T Q\|_2$ for MGS? What seems to be the link between $\kappa(A)$ and $\|I - Q^T Q\|_2$ for CGS? Which algorithm seems the best from the stability point of view? (No explanation needed, just observe and deduce, no need to prove anything, this is really hard, explanations will be given in class.)

Provide your code, your results and your deduction.

3 Scalability

Starting from a random matrix A spread on the processors by rows (i.e. processor #1 has the first $n/nproc$ rows, ...) you have below the code for MGS algorithm using MPI and BLAS:

```

    for ( j = 0; j < n; j++) {
        for ( i = 0; i < j; i++) {
            alpha = cblas_ddot(mloc,&(A[i*mloc]),1,&(A[j*mloc]),1) ;
            MPI_Allreduce( &alpha, &(R[j*n+i]), 1, MPI_DOUBLE,
                MPI_SUM, MPI_COMM_WORLD);
            cblas_daxpy(mloc,-R[j*n+i],&(A[i*mloc]),1,&(A[j*mloc]),1);
        }
        alpha = cblas_dnorm2(mloc,&(A[j*mloc]),1);
        alpha = alpha*alpha ;
        MPI_Allreduce( &alpha, &(R[j*n+j]), 1, MPI_DOUBLE,
            MPI_SUM, MPI_COMM_WORLD);
    }

```

```
        R[j*n+j] = sqrt(R[j*n+j]);  
        cblas_dscal(mloc, 1/R[j*n+j], &(A[j*mloc]), 1);  
    }
```

A complete demo code has been provided to you. It initializes MPI, the matrices, run and time MGS and then check the result.

The goal is first to code the CGS variant and then to do a scalability analysis of both algorithms. Which algorithm seems the best from the scalability point of view? (Explanations welcome.)

Provide your code, your results and your explanation.

4 Bonus

1. if columns of the matrix A are available by blocks, do you see any other faster schemes to perform the QR -factorization of the matrix A ? (stability and scalability of the solution.)
2. if the whole matrix A is available from the beginning, do you see any other faster schemes to perform the QR -factorization of the matrix A ? (stability and scalability of the solution.)