Amanda Lauen

Dr. Karen Braman

MATH 415/515

11th March 2022

Mini Coding Project Write-Up

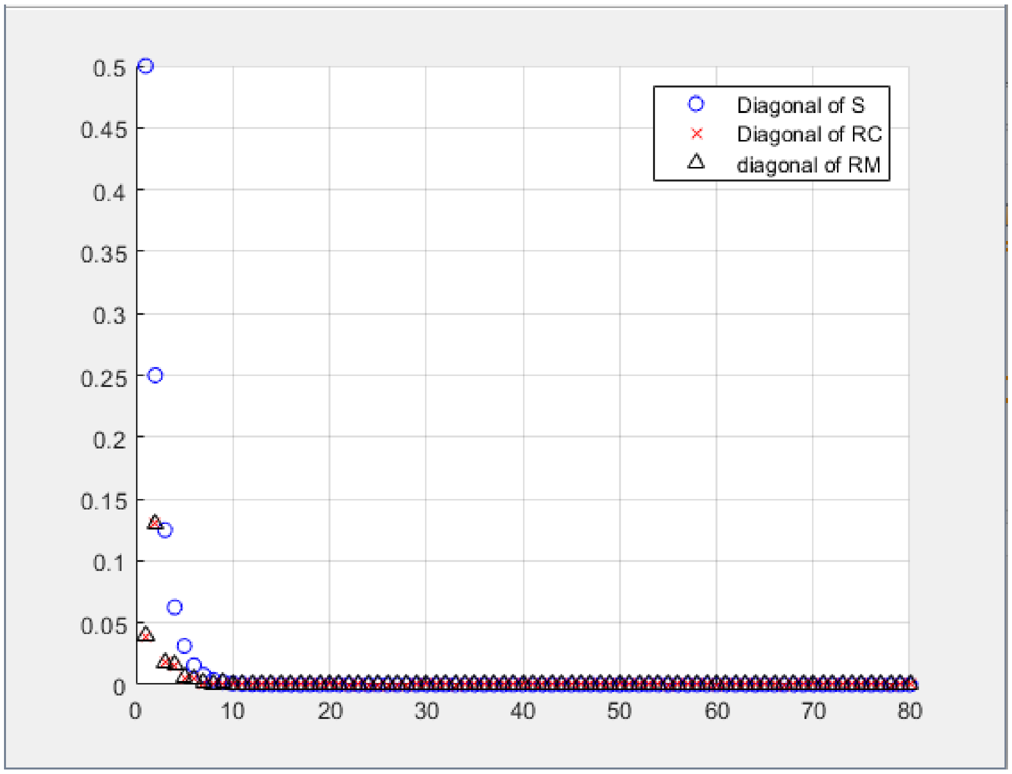
Classical and Modified Gram-Schmidt MATLAB Coding Write-Up

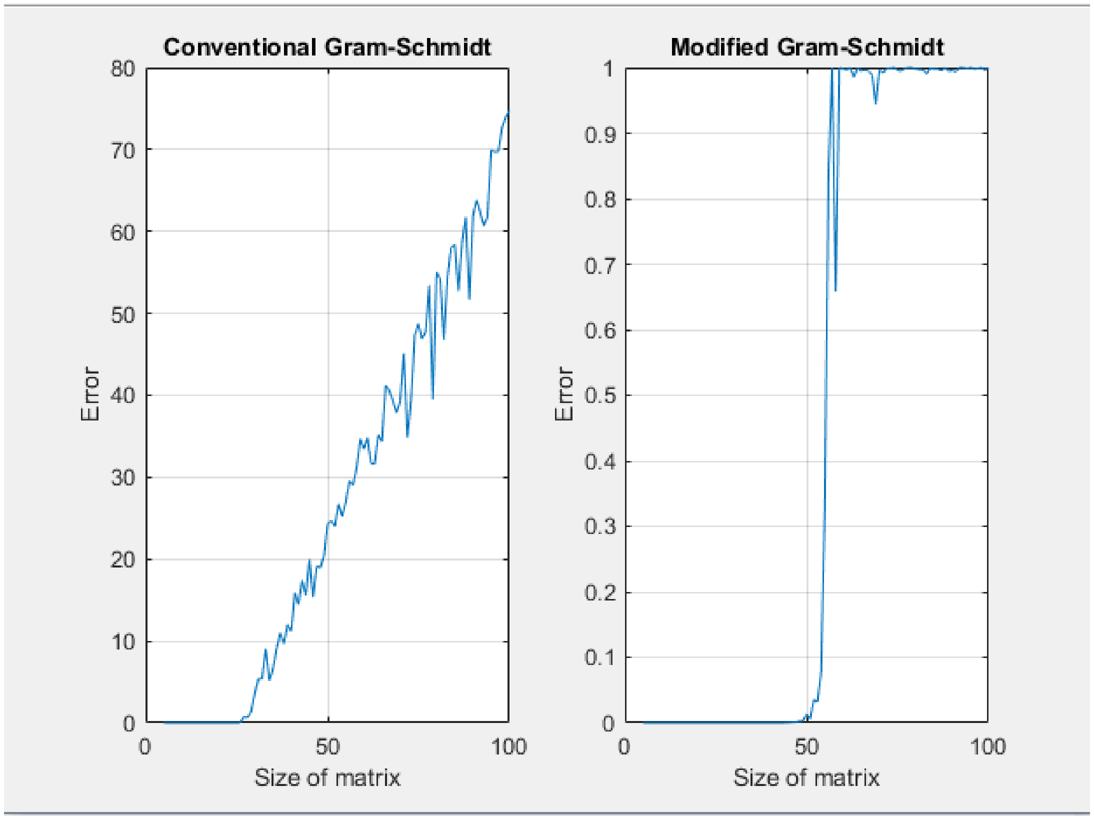
We were tasked to code Classical and Modified Gram-Schmidt Methods in this project. By using two separate functions, I was able to have each function accept an m x n matrix A with $m \geq n$ and have it return a m x n matrix Q with orthogonal columns that spanned the same space as the columns of A and an n x n upper triangular matrix R such that A=QR. All the code for this project will be higlighted in the Official Code section of this document.

I then tested my code with the tests given from the project document and was able to show the norms of these results. The first two norms were a measure of whether A was factored correctly. The second two norms measured whether the Q matrices had orthogonal columns. By doing this test, it helped with building the actual matrix for experimentation, 2×10^{-16}.

The results showed that U and V are random orthogonal matrices, S is a diagonal matrix with decreasing powers of 2 as diagonal elements $(\frac{1}{2}, \frac{1}{4}, \ldots)$, and it built A as a product of those three, allowing us to determine the diagonal elements of the computed R matrices. These should also be approximately the same as the diagonal elements of S.

By getting these values, the best way to represent them is via semiology plots. The following chart represents the semiology plots:





These plots show that the diagonal elements are built as $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \ldots)$. So, as the number of rows/columns increases, the element in the diagonal (n, n) also increases. For example, when N = 80, the last element is $\frac{1}{2}^{80}$, which is zero in this case. So, that is why for all matrices in A, RC, and RM, as the index increases, the element decreases and will decrease until they are zero. The magnitude at which the diagonal elements competed via CGS starts to lose accuracy is when N = 27. The magnitude at which the diagonal elements computed via MGS start to lose accuracy is when N is approximately 50. After these calculations, then the norms have to be evaluated.

The norms that I calculated with A's factors were 7.043e-17 with mycgs code and 7.0763e-17 with mymgs code. I then tested the result from the product of the matrix Q from each method. The result of the product of $Q^T * Q$ should be equal to the identity matrix. With that fact, the norm between the identity and the product of the matrix $Q^T * Q$ from mycgs came out to be 54.8640. Also, the norm between the identity of and the product of the matrix $Q^T * Q$ from mymgs was 0.9976. We can see that the Modified Gram-Schmidt method is more accurate because the norm is smaller than the one calculated using Classical Gram-Schmidt. Then, I tested MATLAB's built-in QR factorization.

When the norms of the built-in QR factorization were calculated, they came out as follows: the norm between A and the factored matrix A constructed with Q*H from the built-in QR function = 1.0754e-16 and the norm between the identity and the $Q^T * Q$ obtained from the built-in QR function = 1.9257e-15. We can see that the built-in method qr(A) and the Modified Gram-Schmidt methods are the most accurate. The reason being is because, for a matrix of size 80x80, the norms are still close to zero (something like 10^{-15} or 10^{-16}). However, for the conventional Gram-Schmidt, this is not the case. We can see that the norms are not zero, and this is because, as the size of the matrix increases, the error also increases. For a matrix of size

10x10 (for example), the error will be close to zero, but the error is significant for this example (a matrix of size 80x80). An idea as to why this phenomenon happens is that since MATLAB uses the Householder Triangularization method versus the regular version of Gram-Schmidt to calculate their values, some of the calculated values can differ. Also, another idea as to why this could happen is because of machine round-off and error. Since the computer is set only to have a specific round-off value (in this case, four decimal places), it could also make the results seem different. Thus, all methods for calculating the QR factorization of a matrix are valid in their ways, and it shows that there is more than one way to find a QR factorization.

In sum, when testing Classical and Modified Gram-Schmidt methods, it can be determined that Modified Gram-Schmidt is more accurate between the two. These results can be verified when comparing norms and the built-in MATLAB QR function. Overall, this project showed the differences between the two different Gram-Schmidt methods and how they are calculated using MATLAB and Trefethen-Beau's methods.

Official Code

mycgs.m

```
function [QC, RC] = mycgs(A)
% Get size of A
    [m, n] = size(A);
    % Create QC and RC
    QC = zeros(m,n);
    RC = zeros(n,n);

    % Algorithm 7.1 Classical Gram Schmidt
    for j = 1:n
        vj = A(:, j); % column j
        for i = 1:j - 1
            RC(i,j) = QC(:,i)'*A(:,j);
            vj = vj - RC(i,j)*QC(:,i);
        end
        RC(j,j) = norm(vj);
        QC(:,j) = vj/RC(j,j);
    end
end
```

mymsg.m

```
function [QM, RM] = mymgs(A)
% Get size of A
    [m, n] = size(A);
    % Create QC and RC
    QM = zeros(m,n);
    RM = zeros(n,n);

    V = A;

    % Algoritm 8.1 Modified Gram-Schmidt
    % for i = 1:n
    %     vi = ai
    % end
    for i = 1:n
        RM(i,i) = norm(V(:,i));
        QM(:,i) = V(:,i)/RM(i,i);
        for j = i + 1:n
            RM(i,j) = QM(:,i)'*V(:,j);
            V(:,j) = V(:,j) - RM(i,j)*QM(:,i);
        end
    end
end
```

NewTest.m

```
clc, clear all, close all

% Build actual matrix
N = 80;
[U,X] = qr(randn(N));
[V,X] = qr(randn(N));
```

```
S = diag(2.^(-1:-1:-N));
A = U*S*V;

% Test mycgs
[QC, RC] = mycgs(A);
[QM, RM] = mymgs(A);

% Calculate norms
fprintf("The following norms test that the factorization of A by each method
is correct\nAs the norm is smaller, the method is more accurate\n");
fprintf("Norm between A and QC*RC (result from mycgs)\n");
norm(A-QC*RC)
fprintf("\nNorm between A and QM*RM (result from mymgs)\n");
norm(A-QM*RM)

fprintf("Now we test the result from the product of matrix Q from each
method\n");
fprintf("The result of the product of Q'*Q should be equal to the Identity
matrix\n");
fprintf("So, we will compute the norm between that product and a Identity
Matrix\n");
fprintf("\nNorm between the identity and the product of matrix Q'*Q from
mycgs\n");
norm(eye(N,N)-QC'*QC)
fprintf("\nNorm between the identity and the product of matrix Q'*Q from
mymgs\n");
norm(eye(N,N)-QM'*QM)
fprintf("\nWe see that the Modified Gram-Schmidt is better because the norm
is smaller\n");

%% produce a plot of the diagonal elements of S, RC and RM
diagS = diag(S);
diagRC = diag(RC);
diagRM = diag(RM);

figure
hold on
semilogy(diagS, 'bo')
semilogy(diagRC,'rx')
semilogy(diagRM,'k^')
legend('Diagonal of S', 'Diagonal of RC', 'diagonal of RM')
grid on

%% Use built-in QR factorization
fprintf("Now, we test MATLAB's built-in function for QR factorization\n");
[QH,RH] = qr(A);
% Check norms
fprintf("\nNorm between A and the factored matrix A constructed with Q*H from
the built-in qr function\n");
norm(A-QH*RH)
fprintf("\nNorm between the Identity and Q'*Q obtained from the built-in qr
function\n");
norm(eye(N,N)-QH'*QH)

err1 = [];
```

```
err2 = [];
i = 1;
for N = 5:100
    [U,X] = qr(randn(N));
    [V,X] = qr(randn(N));
    S = diag(2.^(-1:-1:-N));
    A = U*S*V;

    [QC, RC] = mycgs(A);
    [QM, RM] = mymgs(A);

    % Calculate norms
    norm1 = norm(eye(N,N)-QC'*QC);
    norm2 = norm(eye(N,N)-QM'*QM);

    err1(i) = norm1;
    err2(i) = norm2;
    i = i + 1;
end

figure
subplot(1,2,1)
plot(5:100, err1)
xlabel('Size of matrix')
ylabel('Error')
title('Conventional Gram-Schmidt')
grid on
subplot(1,2,2)
plot(5:100, err2)
grid on
xlabel('Size of matrix')
title('Modified Gram-Schmidt')

ylabel('Error')
```