# Table of Contents



# Introduction

```
%Written by Ian Good on 11/02/2020. For any questions, please reach me
 at: iangood@uw.edu
%{
This Matlab script impliments three different versions of QR
 decompostions
and explores the concepts of stability, conditioning, and compute time
within datasets. We then go on to explore floating point round off and
 the
effects it can have on processing high order polynomials. Finally, we
 look
at ways to better condition data through the addition of small noise
 added
to the data.
%}
clear all;
close all;
clc;
```

# QR Decomposition - Effect of Conditioning on Computation Speed

See the function section at the end of this document for the function implimentations.

```
ASq = rand(200);
tic
[qMGS, rMGS] = MGS(ASq); %Modified Grahm-Schmidt, written by yours
 truly
tocGS = toc
tic
[qHHR,rHHR]  = qrfactor(ASq); %Householder Reflector
tocQR = toc
tic
[qMAT,rMAT]  = qr(ASq); %matlab's built-in QR decomposer
tocMat = toc
disp('-----')
```

```
% note for a real, square matrix, that is well conditioned the fastest
 solve was MGS, followed by Matlab's
% implimentation, with qfactor an order of magnitude slower (n=m=200).

condASq = cond(ASq); %show the condition number of our A matrix
disp(strcat('The condition number is...',{'
 '},num2str(condASq))); %note how small this value is...
ASqIll =ASq; ASqIll(:,end) =ASqIll(:,1);

tic
[qMGSIll, rMGSIll] = MGS(ASqIll); %Modified Grahm-Schmidt, written by
 yours truly
tocGSIll = toc
tic
[qHHRIll,rHHRIll]  = qrfactor(ASqIll); %Householder Reflector
tocQRIll = toc
tic
[qMATIll,rMATIll]  = qr(ASqIll); %matlab's built-in QR decomposer
tocMatIll = toc

condASqIll = cond(ASqIll); %show the condition number of our A matrix
disp(strcat('The condition number is...',{'
 '},num2str(condASqIll))); %note how large this value is...
disp('-----')

% note for a real, square matrix, that is poorly conditioned the
 fastest solve was Matlab's
% implimentation, with MGS an order of magnitude slower, and qfactor
 an order of magnitude slower (n=m=200).
```

# QR Decomposition - Effect of Matrix Shape on Computation Speed

```
ATall = rand(200,100);
tic
[qMGSTall, rMGSTall] = MGS(ATall); %Modified Grahm-Schmidt, written by
 yours truly
tocGST = toc
tic
[qHHRTall,rHHRTall]  = qrfactor(ATall); %Householder Reflector
tocQRT = toc
tic
[qMATTall,rMATTall]  = qr(ATall); %matlab's built-in QR decomposer
tocMatT = toc
disp('-----')
% note for a real, overdetermined matrix, the fastest solve was
 Matlab's
% implimentation, with MGS two orders of magnitude slower, and qfactor
 an additional order of magnitude slower (m=200, n=100).

% try with a triangular matrix
ATri = triu(rand(200));
```

```
tic
[qMGSTri, rMGSTri] = MGS(ATri); %Modified Grahm-Schmidt, written by
 yours truly
tocGSTri = toc
tic
[qHHRTri,rHHRTri]  = qrfactor(ATri); %Householder Reflector
tocQRTri = toc
tic
[qMATTri,rMATTri]  = qr(ATri); %matlab's built-in QR decomposer
 ( based off of Householder Reflector)
tocMatTri = toc
disp('-----')
% note for a real, upper triangular matrix, the fastest solve was
 Matlab's
% implimentation, with MGS one order of magnitude slower, and qfactor
 two orders of magnitude slower (n=m=200).
```

```
tocQRT =

    0.1416


tocMatT =

    0.0010

-----

tocQRTri =

    0.3258


tocMatTri =

    0.0010

-----
```

# QR Decomposition - Accuracy of Computation

```
% compute the error(L-2 norm) for the square, real well and ill fitted
 matrices
errorSqMGS = norm(qMGS*rMGS-ASq) / norm(ASq);
errorSqMGSIll = norm(qMGSIll*rMGSIll-ASqIll) / norm(ASqIll);
disp(strcat('The L-2 Norm for MGS on a well conditioned square matrix
 is...',{' '},num2str(errorSqMGS)));
disp(strcat('The L-2 Norm for MGS on an ill conditioned square matrix
 is...',{' '},num2str(errorSqMGSIll))); %note how small both of these
 are...

errorSqHHR = norm(qHHR*rHHR-ASq) / norm(ASq);
```

```
errorSqHHRIll = norm(qHHRIll*rHHRIll-ASqIll) / norm(ASqIll);
disp(strcat('The L-2 Norm for HHR on a well conditioned square matrix
 is...',{' '},num2str(errorSqHHR)));
disp(strcat('The L-2 Norm for HHR on an ill conditioned square matrix
 is...',{' '},num2str(errorSqHHRIll))); %note how small both of these
 are...

errorSqMAT = norm(qMAT*rMAT-ASq) / norm(ASq);
errorSqMATIll = norm(qMATIll*rMATIll-ASqIll) / norm(ASqIll);
disp(strcat('The L-2 Norm for MAT on a well conditioned square matrix
 is...',{' '},num2str(errorSqMAT)));
disp(strcat('The L-2 Norm for MAT on an ill conditioned square matrix
 is...',{' '},num2str(errorSqMATIll))); %note how small both of these
 are...
disp('-----')
%{
This shows that the actual error generated through computing the QR
decomposition using any method is in fact, very small. Thus compute
 time
becomes the most important factor. In general, the Modified Grahm-
Schmidt
Algorithm performed better than the QR Factor (Householder Reflector)
function. However both dwarfed in comparison to Matlab's
 implimentation by
at least an order of magnitude for matrices around size n=m=200.
Matlab has done well to develop a tool that is both robust to large
condition numbers and fast to solve matrices. We can rest assured
 calling
qr in matlab will be as good as or better than an personally devloped
algorithms and the users time should instead be spent thinking about
 the
structure of the data and its results.

%}
```

```
    'The L-2 Norm for MGS on a well conditioned square matrix is...
 7.9335e-17'

    'The L-2 Norm for MGS on an ill conditioned square matrix is...
 7.927e-17'

    'The L-2 Norm for HHR on a well conditioned square matrix is...
 4.3729e-16'

    'The L-2 Norm for HHR on an ill conditioned square matrix is...
 4.3533e-16'

    'The L-2 Norm for MAT on a well conditioned square matrix is...
 4.4852e-16'

    'The L-2 Norm for MAT on an ill conditioned square matrix is...
 4.4789e-16'

-----
```

# Looking at Left and Right iterations of Polynomials

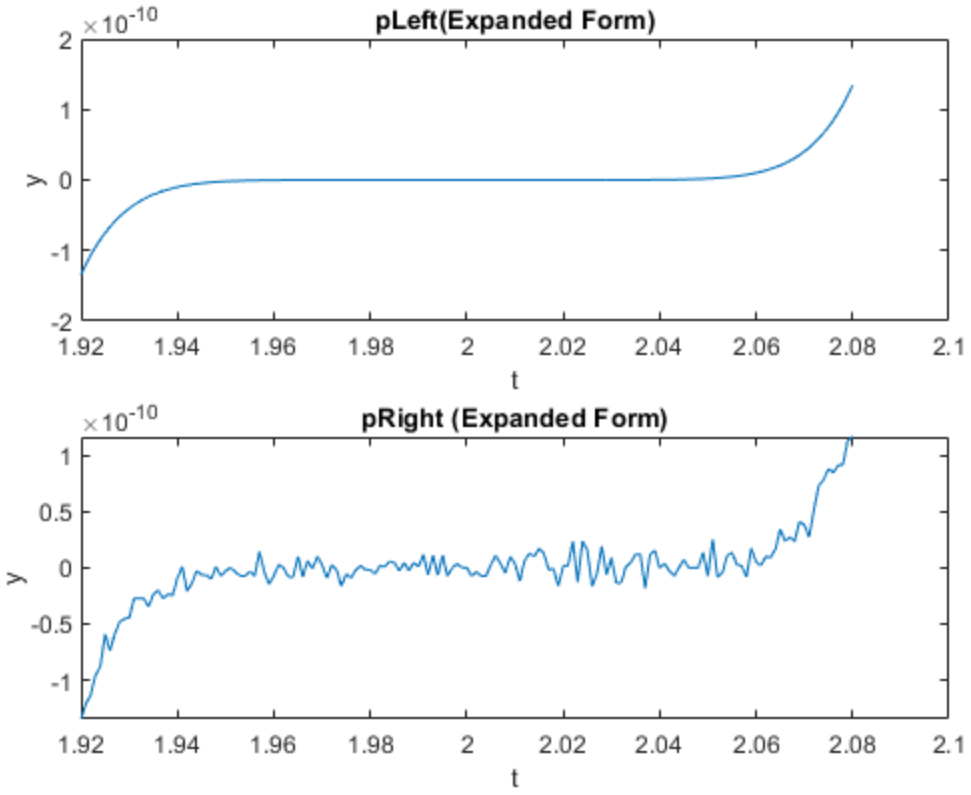
```
xspan = [1.920,2.080];
dx = .001;
xDiscrete = xspan(1):dx:xspan(2);


pRight =
 xDiscrete.^9-18*xDiscrete.^8+144*xDiscrete.^7-672*xDiscrete.^6+2016*xDiscrete.^5-
pLeft = (xDiscrete-2).^9;

figure();hold on;
subplot(2,1,1)
plot(xDiscrete,pLeft)
title('pLeft(Expanded Form)');ylabel('y');xlabel('t')

subplot(2,1,2)
plot(xDiscrete,pRight)
title('pRight (Expanded Form)');ylabel('y');xlabel('t')

%this is a perfect example of how interger roundoff can really destroy
 the
%accuracy of any results generated. It is the job of the keen
 programmer to keep
%a watchful eye both what they implement, but also HOW they impliment
 it.
```

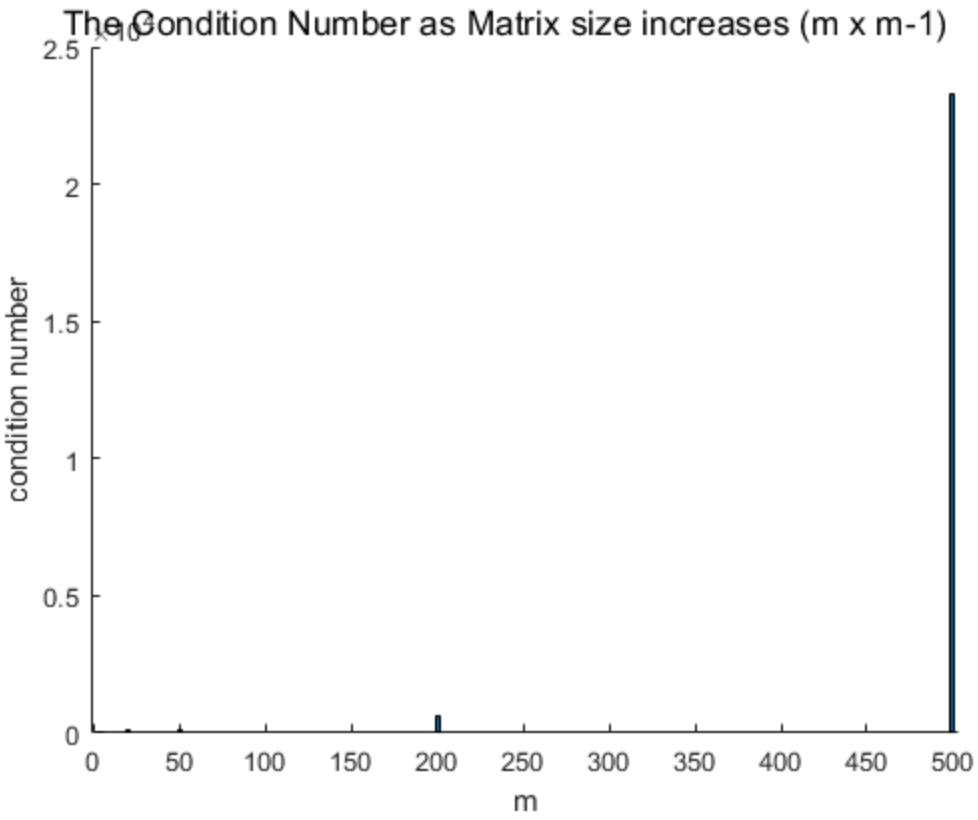



# Exploring Conditioning - Function of size

```
m = [2,5,20,50,200,500]; %pick sizes across multiple orders of
 magnitude
n = m-eye(size(m)); %gets us a matrix that when we add the first
 collumn to the end.

figure(); hold on
sgtitle('The Condition Number as Matrix size increases (m x
 m-1)');ylabel('condition number');xlabel('m')
for iter = 1:length(m)
    ARand= randn(m(iter),n(iter));
    condARand(iter) =cond(ARand);

end
bar(m,condARand) %plot the results as a bar chart
```




# Exploring Conditioning - The Power of Adding Noise

```
mNoise = 20;
nNoise = mNoise-1; %gets us a matrix that when we add the first
 collumn to the end
ARandClean = rand(mNoise,nNoise);
ARandClean = [ARandClean ARandClean(:,1)]; %Adds the first collumn to
 the end of matrix.
condAClean = cond(ARandClean);
detAClean = det(ARandClean);

disp(strcat('The condition number without noise is...',{'
 '},num2str(condAClean))); %note how big this value is...
disp(strcat('The determinate without noise is...',{'
 '},num2str(detAClean)));
disp('-----')
noiseOrder = [1E-14,1E-12,1E-10,1E-8,1E-6]; %order of magnitude for
 the noise to be added. Note, we do not go above 1E-6 since that is
 the accuracy of ode45
randNoise = rand(mNoise,1);
ARandNoise = ARandClean;
for iter = 1:length(noiseOrder)
    noiseV = noiseOrder(iter).*randNoise;
```

```
    ARandNoise(:,end) = ARandClean(:,end).*noiseV;
    condANoise = cond(ARandNoise);
    disp(strcat('The condition number with',{'
 '},num2str(noiseOrder(iter)),{' '},'noise added is...',{'
 '},num2str(condANoise))); %note how small this value is... these
 algorithms really are impressive

end
disp('-----')
%{
We see a massive downturn in the condition number of the matrix when
 applying even a small ammount of noise.
During one run with 1E14 & 1E-12 noise added, we returned the
 following results from a matrix size 200 x 200:

The condition number without noise is... 1.577355513332925e+17
The determinate without noise is... -3829618042.7004
The condition number with 1e-14 noise added is... 2.856762955227868e
+16
The condition number with 1e-12 noise added is... 2.856762955228167e
+14
...

This is a large reduction in the condition number but not quite what
 we
were hoping for in reduction of size...
If instead we applied the noise to the whole set of data we would
 get...
%}
randNoiseMatrix = randn(mNoise); %note the randN here. From my
 testing, this resulted in a larger reduction in the condition number
for iter = 1:length(noiseOrder)
    ARandNoiseFull = ARandClean+noiseOrder(iter)*randNoiseMatrix; %A
 plus full noise matrix
    condANoiseFull = cond(ARandNoiseFull);
    disp(strcat('The condition number with',{'
 '},num2str(noiseOrder(iter)),{' '},'noise added to the whole matrix
 is...',{' '},num2str(condANoiseFull)));

end
disp('-----')
%{
This shows an even greater reduction of the condition number... For
 the
same example as above but with a different random matrix the condition
 number becomes...

The condition number with 1e-14 noise added is... 3.099590613475094e
+15
The condition number with 1e-12 noise added is... 3.104291126738605e
+13
%}
```

```
    'The condition number without noise is... 5.81081369824321e+16'
```

```
    'The determinate without noise is... 1.9751e-17'

-----
    'The condition number with 1e-14 noise added is...
 4.166875341718062e+16'

    'The condition number with 1e-12 noise added is...
 416687534171878.8'

    'The condition number with 1e-10 noise added is...
 4166875341718.177'

    'The condition number with 1e-08 noise added is...
 41668753417.192'

    'The condition number with 1e-06 noise added is... 416687534.1723'

-----
    'The condition number with 1e-14 noise added to the whole matrix
 is... 3713967647680690'

    'The condition number with 1e-12 noise added to the whole matrix
 is... 44373317881715.45'

    'The condition number with 1e-10 noise added to the whole matrix
 is... 442920738961.7635'

    'The condition number with 1e-08 noise added to the whole matrix
 is... 4429138281.8761'

    'The condition number with 1e-06 noise added to the whole matrix
 is... 44288714.2049'

-----
```

# Defining Functions - QR Implimentation (Grahm-Schmidt)

```
function [Q,R]=MGS(A)

[m,n]=size(A); R=zeros(n);
Q=A;
for k=1:n
  for ii=1:k-1
    R(ii,k)=Q(:,ii)'*Q(:,k); %solve step for R
    Q(:,k)=Q(:,k)-R(ii,k)*Q(:,ii); %use that to get Q

  end
  R(k,k)=norm(Q(:,k)); Q(:,k)=Q(:,k)/R(k,k); %build full R

end
```

```
end
```

```
tocGS =

    0.0470


tocGSIll =

    0.0218


tocGST =

    0.0068


tocGSTri =

    0.0189
```

# Defining Functions - Class QR Implimentation (Householder reflection)

```
function [Q,R] = qrfactor(A)

[m,n] = size(A);
Q=eye(m);
for k = 1:n
    % Find the HH reflector
    z = A(k:m,k);
    v = [ -sign(z(1))*norm(z) - z(1); -z(2:end) ];
    v = v / sqrt(v'*v);   % remoce v'*v in den

    % Apply the HH reflection to each column of A and Q
    for j = 1:n
        A(k:m,j) = A(k:m,j) - v*( 2*(v'*A(k:m,j)) );
    end
    for j = 1:m
        Q(k:m,j) = Q(k:m,j) - v*( 2*(v'*Q(k:m,j)) );
    end

end

Q = Q';
R = triu(A);  % exact triangularity
end
```

```
tocQR =

    0.4003


tocMat =

    0.0084

-----
    'The condition number is... 3235.2567'


tocQRIll =

    0.3567


tocMatIll =

    0.0017

    'The condition number is... 1.809721693615064e+17'

-----
```

*Published with MATLAB® R2019b*