CX 4640 Assignment 5

Wesley Ford

September 21st, 2020

1. Chapter 6, Question 7: Find the QR factorization of the general 2×2 matrix

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

For $A = QR$, we know that Q is an orthonormal matrix, that is, the magnitude of the columns of Q is 1, and the inner product of any two columns is 0. We also know that R will be an upper triangular matrix. Rewriting $A = QR$, we get

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix}$$

or

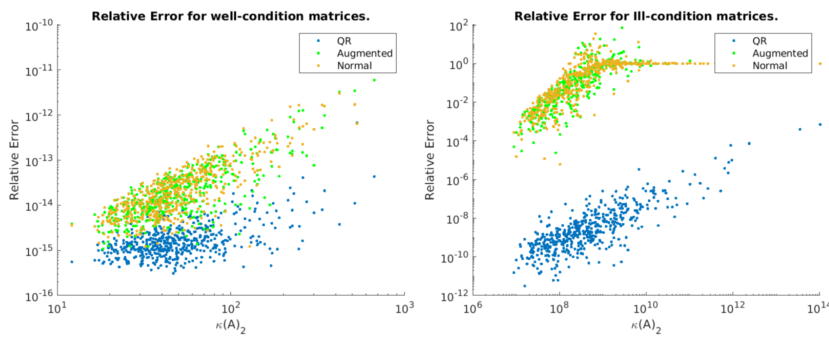
$$\begin{pmatrix} \boldsymbol{a_1} & \boldsymbol{a_2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{q_1} & \boldsymbol{q_2} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix}$$

where $\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{q_1}$ and $\boldsymbol{q_2}$ are columns of their respective matrices.

We know that $\boldsymbol{a_1} = r_{11}\boldsymbol{q_1}$, and $\boldsymbol{a_2} = r_{12}\boldsymbol{q_1} + r_{22}\boldsymbol{q_2}$. We also know that the inner product between the columns of Q, $\langle \boldsymbol{q_1}, \boldsymbol{q_2} \rangle = 0$, while the inner product of a column of Q and itself is $\langle \boldsymbol{q_1}, \boldsymbol{q_1} \rangle = \langle \boldsymbol{q_2}, \boldsymbol{q_2} \rangle = 0$. We satisfy $\boldsymbol{a_1} = r_{11}\boldsymbol{q_1}$ by setting $\boldsymbol{q_1} = \dfrac{\boldsymbol{a_1}}{\|\boldsymbol{a_1}\|}$ and $r_{11} = \|\boldsymbol{a_1}\|$. We can then solve for r_{12} by taking the inner product of both sides $\boldsymbol{a_2} = r_{12}\boldsymbol{q_1} + r_{22}\boldsymbol{q_2}$ and $\boldsymbol{q_1}$ to get $r_{12} = \langle \boldsymbol{a_2}, \boldsymbol{q_1} \rangle$. We also can use the same equation to solve for $\boldsymbol{q_2}$ and r_{22}. By manipulating the equation, we get $r_{22}\boldsymbol{q_2} = \boldsymbol{a_2} - r_{12}\boldsymbol{q_1} = \boldsymbol{a_2} - \langle \boldsymbol{a_2}, \boldsymbol{q_1} \rangle \boldsymbol{q_1}$. $\boldsymbol{q_2}$ must be unit, so we can set $r_{22} = \|\boldsymbol{a_2} - \langle \boldsymbol{a_2}, \boldsymbol{q_1} \rangle \boldsymbol{q_1}\|$ and divide, so we get $\boldsymbol{q_2} = \dfrac{\boldsymbol{a_2} - \langle \boldsymbol{a_2}, \boldsymbol{q_1} \rangle}{\|\boldsymbol{a_2} - \langle \boldsymbol{a_2}, \boldsymbol{q_1} \rangle \boldsymbol{q_1}\|} \boldsymbol{q_1}$.

2. When comparing these three methods for solving $\min_x = \|b - Ax\|_2$, we find that in general, QR factorization performs the best at getting accurate solutions and residuals compared with the Normal Equations method and the Augmented system method, both of which perform similarly to each other. That being said, for well-condition matrices, all three methods perform well, with the discrepancies making a substantial difference primarily for ill-condition matrices. For the purposes of comparison between the methods, we assume that Matlab's left matrix division computes an accurate solution and an accurate residual can be computed from that solution.

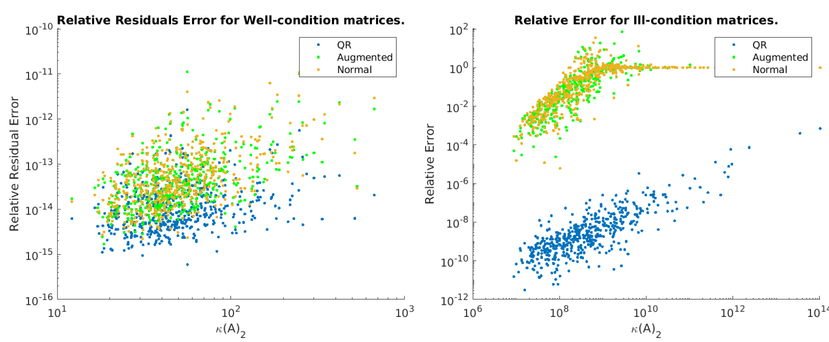
   We first compare the least squares solutions each method produces for a given matrix to the solution given by A\b in Matlab. Doing so for well condition matrices ($\kappa(A)_2 < 10^3$) produces the following results.



   As we see, the relative error between our "true" solution is minimized when using QR factorization, while the Augmented and Normal Equations methods have about a single order of magnitude difference in the relative error of those solutions.

   Doing the same for ill-conditioned Vandermonde matrices ($\kappa(A)_2 > 10^6$), we see that QR factorization is indeed more stable than the other two matrices, performing several orders of magnitude better than the other two methods.

   Measuring the relative residual error for both types of matrices shows the same trend.



   In general, for well-conditioned matrices, all three methods perform well. producing solutions and residuals almost identical to what Matlab computes to be the correct solution. However, for ill-conditioned matrices, QR factorization clearly provides more precision in its solution than either the Normal Equations or the Augmented Matrix system.

   [Code for this problem which generated the graphs is at the end of the document].

3. Chapter 8, question 8: Use the definition of the psudo-inverse matrix A in terms of its singular values and singular vectors to show the flowing relations hold.

   The psudo-inverse of $A = U\Sigma V^T$ is $A^\dagger = V\Sigma^\dagger U^T$, where $\Sigma^\dagger = \begin{cases} 0 & \text{if } \sigma_i = 0 \\ \frac{1}{\sigma_i} & \text{if } \sigma_i \neq 0 \end{cases}$

   - $AA^\dagger A = A$

     $$AA^\dagger A = (U\Sigma V^T)(V\Sigma^\dagger U^T)(U\Sigma V^T) = U\Sigma\Sigma^\dagger\Sigma V^T$$

     The term $\Sigma\Sigma^\dagger\Sigma$ is just Σ, as $\Sigma^\dagger\Sigma$ is almost the identity matrix, except when $\sigma_i = 0$, but in that case, those values in Σ are already 0, so $\Sigma\Sigma^\dagger\Sigma = \Sigma$. We ultimately get $U\Sigma V^T = A$.

   - $A^\dagger AA^\dagger = A^\dagger$

     $$A^\dagger AA^\dagger = (V\Sigma^\dagger U^T)(U\Sigma V^T)(V\Sigma^\dagger U^T) = V\Sigma^\dagger\Sigma\Sigma^\dagger U^T.$$

     Following the same reasoning in the previous question, we can say that $\Sigma^\dagger\Sigma\Sigma^\dagger = \Sigma^\dagger$, and so $A^\dagger AA^\dagger = V\Sigma^\dagger U^T = A^\dagger$.

   - $(AA^\dagger)^T = AA^\dagger$

     $$(AA^\dagger)^T = ((U\Sigma V^T)(V\Sigma^\dagger U^T))^T = (U\Sigma\Sigma^\dagger U^T)^T = U\Sigma^\dagger\Sigma U^T.$$

     We know that $\Sigma\Sigma^\dagger = \Sigma^\dagger\Sigma$, so we can say that $(AA^\dagger)^T = (U\Sigma\Sigma^\dagger U^T)^T = U\Sigma\Sigma^\dagger U^T = (U\Sigma V^T)(V\Sigma^\dagger U^T) = AA^\dagger$

   - $(A^\dagger A)^T = A^\dagger A$

     $$(A^\dagger A)^T = ((V\Sigma^\dagger U^T)(U\Sigma V^T))^T = (V\Sigma^\dagger\Sigma V^T)^T = V\Sigma\Sigma^\dagger V^T$$

     Again, we know that $\Sigma\Sigma^\dagger = \Sigma^\dagger\Sigma$, so we can say that $(A^\dagger A)^T = (V\Sigma^\dagger\Sigma V^T)^T = V\Sigma\Sigma^\dagger V^T = V\Sigma^\dagger\Sigma V^T = (V\Sigma^\dagger U^T)(U\Sigma V^T) = A^\dagger A$

```
%Code for Problem 2 of Assignment 5. All least_squares implementations are as functions

m = 11;
n = 10;

j=500;
condition_num = zeros(1,j);
Vcondition_num = zeros(1,j);
x_norm_errs = zeros(1,j);
x_augmented_errs = zeros(1,j);
x_qr_errs = zeros(1,j);
Vx_norm_errs = zeros(1,j);
Vx_augmented_errs = zeros(1,j);
Vx_qr_errs = zeros(1,j);

r_norm_rel = zeros(1,j);
r_augmented_rel = zeros(1,j);
r_qr_rel = zeros(1,j);
Vr_norm_rel = zeros(1,j);
Vr_augmented_rel = zeros(1,j);
Vr_qr_rel = zeros(1,j);

diff_norm = zeros(1,j);
diff_aug =  zeros(1,j);
diff_qr =  zeros(1,j);

Vdiff_norm = zeros(1,j);
Vdiff_aug =  zeros(1,j);
Vdiff_qr =  zeros(1,j);

for i=1:j
%Calculate random A matrix
A = rand(m,n);
b = rand(m,1);
%Calcualte random V matrix
r = rand(m, 1);
V = (fliplr(vander(r)));
V = V(:,1:n);
Vb = rand(m,1);
%Calcualte condition numbers of matrices
condition_num(i) = cond(A);
Vcondition_num(i) = cond(V);

%Calculate "True" soltuions and residuals using Matlab '\'
x = A\b;
r = b-A*x;
Vx = V\Vb;
Vr = Vb-V*Vx;


%Calculate solutions using normal equations
[x_normal, r_normal] = least_squares_normal(A, b);
[Vx_normal, Vr_normal] = least_squares_normal(V, Vb);
```

```
%Calcualte solutions using Augmented matrix
[x_augmented, r_augmented] = least_squares_augmented(A, b);
[Vx_augmented, Vr_augmented] = least_squares_augmented(V, Vb);

%Calcualte solutions using qr factorization
[x_qr, r_qr] = least_squares_qr(A, b);
[Vx_qr, Vr_qr] = least_squares_qr(V, Vb);

%Calcualte relatve error for each solution
x_norm_errs(i) = norm(x-x_normal) ./ norm(x);
x_augmented_errs(i) = norm(x-x_augmented) ./ norm(x);
x_qr_errs(i) = norm(x-x_qr) ./ norm(x);

Vx_norm_errs(i) = norm(Vx-Vx_normal) ./ norm(Vx);
Vx_augmented_errs(i) = norm(Vx-Vx_augmented) ./ norm(Vx);
Vx_qr_errs(i) = norm(Vx-Vx_qr) ./ norm(Vx);

%Calcualte relative residuals for each solution
r_norm_rel(i) = norm(r_normal);
r_augmented_rel(i) = norm(r_augmented);
r_qr_rel(i) = norm(r_qr);
Vr_norm_rel(i) = norm(Vr_normal);
Vr_augmented_rel(i) = norm(Vr_augmented);
Vr_qr_rel(i) = norm(Vr_qr);

diff_norm(i) = norm(r_normal - r) ./ norm(r);
diff_aug(i) = norm(r_augmented - r) ./ norm(r);
diff_qr(i) = norm(r_qr - r) ./ norm(r);

Vdiff_norm(i) = norm(Vr_normal - Vr) ./ norm(Vr);
Vdiff_aug(i) = norm(Vr_augmented -Vr) ./ norm(Vr);
Vdiff_qr(i) = norm(Vr_qr - Vr) ./ norm(Vr);

end

%sort data by conditon number
[sorted_cond, order] = sort(condition_num);
[Vsorted_cond, Vorder] = sort(Vcondition_num);

clf
hold on
scatter(sorted_cond, diff_qr(order), '.');
scatter(sorted_cond, diff_aug(order), 'g.');
scatter(sorted_cond, diff_norm(order), '.');
legend("QR","Augmented", "Normal")
title("Relative Residuals Error for Well-condition matrices.")
ylabel("Relative Residual Error")
xlabel("\kappa(A)_2")
set(gca, 'xscale', 'log');
set(gca, 'yscale', 'log');
hold off
```

```
scatter(Vsorted_cond, Vdiff_qr(Vorder), '.');

hold on
scatter(Vsorted_cond, Vdiff_aug(Vorder), 'g.');
scatter(Vsorted_cond, Vdiff_norm(Vorder), '.');
legend("QR","Augmented", "Normal")
title("Relative Residuals for Ill-condition matrices.")
ylabel("Relative Residual Error")
xlabel("\kappa(A)_2")
set(gca, 'xscale', 'log');
set(gca, 'yscale', 'log');
hold off

scatter(sorted_cond, x_qr_errs(order), '.')
hold on
scatter(sorted_cond, x_augmented_errs(order),'g.')
scatter(sorted_cond, x_norm_errs(order),'.')
title("Relative Error for well-condition matrices.")
ylabel("Relative Error")
xlabel("\kappa(A)_2")
set(gca, 'xscale', 'log');
set(gca, 'yscale', 'log');
legend("QR","Augmented", "Normal")
hold off

scatter(Vsorted_cond, Vx_qr_errs(Vorder), '.')
hold on
title("Relative Error for Ill-condition matrices.")
ylabel("Relative Error")
xlabel("\kappa(A)_2")
scatter(Vsorted_cond, Vx_augmented_errs(Vorder), 'g.')
scatter(Vsorted_cond, Vx_norm_errs(Vorder), '.')
set(gca, 'xscale', 'log');
set(gca, 'yscale', 'log');
legend("QR","Augmented", "Normal")
hold off

scatter(Vsorted_cond, Vdiff_qr(Vorder), '.');

hold on
scatter(Vsorted_cond, Vdiff_aug(Vorder), 'g.');
scatter(Vsorted_cond, Vdiff_norm(Vorder), '.');
legend("QR","Augmented", "Normal")
title("Relative Residuals for Ill-condition matrices.")
ylabel("Relative Residual Error")
xlabel("\kappa(A)_2")
set(gca, 'xscale', 'log');
set(gca, 'yscale', 'log');
hold off
```

```
function [x,r] = least_squares_normal(A,b)
%Solves least squares using normal equations
B = A'* A;
y = A' * b;
%Use lu factorization to avoid errors with Vermonde matrices not being
%SPD because of finite precision
[L,U,P] = lu(B);
z = L\(P*y);
x = (U\z);
%       G = chol(B)';
%       z = G\y;
%       x = G'\z;
r = b-A*x;
end

function [x,r] = least_squares_augmented(A,b)
%Solves least squares using augmented matrix method
[m,n] = size(A);
aug_matrix = [eye(m,m), A; A', zeros(n,n)];
aug_vec = [b; zeros(n,1)];
[L,U, P] = lu(aug_matrix);
y = L \ (P * aug_vec);
aug_sol = (U\y);
%Residual is first m terms, x is last n temrs
r = aug_sol(1:m);
x = aug_sol(m+1:end);
end

function [x, r] = least_squares_qr(A,b)
%Sovles least squares using qr factorization
[Q,R] = qr(A,0);
c = Q'*b;
x = R\c;
r = b-A*x;
end
```