MSSC 6040 - Homework 2

Henri Medeiros Dos Reis

November 4, 2022

Problem 1 (5 pts)**.** Let S be any subspace of $\mathbb{C}^m$ and let $P \in \mathbb{C}^{m\times m}$ be the orthogonal projector onto S. Given any vector $x \in \mathbb{C}^m$, prove that Px is the vector in S closest to x in Euclidean distance, i.e., prove that the minimizer of

$$\min_{y\in S} \|x-y\|_2^2$$

is $y = Px$. (Hint: Use the identity $x - y = P(x-y) + (I-P)(x-y)$, and recall the Pythagorean Theorem: if u and v are orthogonal vectors then $\|u+v\|_2^2 = \|u\|_2^2 + \|v\|_2^2$.)

Solution 1. Let y be any vector in S, then since the smallest vector is $y = Px$, then

$$\|x-Px\|_2^2 \le \|x-y\|_2^2$$

$$\Rightarrow \|x-y\|_2^2 = \|P(x-y) + (I-P)(x-y)\|_2^2$$

Then let $u = P(x-y)$ and $v = (I-P)(x-y)$. And since u and v are orthogonal, we can use the Pythagorean theorem.

$$\|u+v\|_2^2 = \|u\|_2^2 + \|v\|_2^2 \Rightarrow$$

$$\|P(x-y) + (I-P)(x-y)\|_2^2 = \|P(x-y)\|_2^2 + \|(I-P)(x-y)\|_2^2 \Rightarrow$$
$$\|Px-Py\|_2^2 + \|x-y-Px+Py\|_2^2$$

And since y is in S, then $Py = y$

$$\Rightarrow \|Px-y\|_2^2 + \|x-Px\|_2^2$$

And in order to minimize this equation in respect to y, we need to solve $\|Px-y\|_2^2 = 0 \Rightarrow Px = y$.

Therefore $\min_{y\in S}\|x-y\|_2^2$ occurs where $y = Px$, and such y exists, because it $Px \in range(A)$

Problem 2 (5 pts)**.** T-B: 6.4

Solution 2. .

a-)

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$range(A) = span\left\{ \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\} = span\left\{ \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\} \text{ which is an orthonormal basis}$$

Then let $\hat{U} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \\ \frac{1}{\sqrt{2}} & 0 \end{bmatrix} \Rightarrow P = \hat{U}\hat{U}^* = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} = P$

Then $x^* = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \Rightarrow Px = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$

b-)

$$B = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$range(B) = span\left\{ \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \right\}$$

The first orthonormal vector is easy to find $q_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}$.

For the second, we have $q_2 = b_2 - P_{<q_1>}b_2 = b_2 - (q_1^* b_2)q_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} - \sqrt{2} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$,

which we just need to normalize $\Rightarrow q_2 = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} \end{bmatrix}$

Therefore, $range(B) = span\left\{ \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} \end{bmatrix} \right\}$, which is an orthonormal basis.

Then, let $\hat{U} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \Rightarrow P = \hat{U}\hat{U}^* = \begin{bmatrix} \frac{5}{6} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{3} & \frac{-1}{3} \\ \frac{1}{6} & \frac{-1}{3} & \frac{5}{6} \end{bmatrix}$

Then $x^* = \begin{bmatrix}1\\2\\3\end{bmatrix} \Rightarrow Px = \begin{bmatrix}\frac{5}{6} & \frac{1}{3} & \frac{1}{6}\\ \frac{1}{3} & \frac{1}{3} & \frac{-1}{3}\\ \frac{1}{6} & \frac{-1}{3} & \frac{5}{6}\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix} = \begin{bmatrix}2\\0\\2\end{bmatrix}$

Problem 3 (5 pts)**.** T-B: 7.1

Solution 3. .

a-)

$$A = \begin{bmatrix}1 & 0\\0 & 1\\1 & 0\end{bmatrix}$$

Since we already have normalized columns of A in the previous exercise, we just need to find R.

$$A = \hat{Q}\hat{R} = \begin{bmatrix}\frac{1}{\sqrt{2}} & 0\\ 0 & 1\\ \frac{1}{\sqrt{2}} & 0\end{bmatrix}\begin{bmatrix}\sqrt{2} & 0\\ 0 & 1\end{bmatrix} \text{ is the reduced QR-factorization}$$

Now for the full QR, we need to add an extra column that makes Q unitary. Which by analyzing Q, we can see that the last column should be the same values as the first column, with one of them being multiplied by -1.

$$Q = \begin{bmatrix}\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}}\\ 0 & 1 & 0\\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}}\end{bmatrix} \text{ and } R = \begin{bmatrix}\sqrt{2} & 0\\ 0 & 1\\ 0 & 0\end{bmatrix}$$

b-)

$$B = \begin{bmatrix}1 & 2\\0 & 1\\1 & 0\end{bmatrix}$$

Again by using the solution from the previous exercise, we have $q_1 = \begin{bmatrix}\frac{1}{\sqrt{2}}\\ 0\\ \frac{1}{\sqrt{2}}\end{bmatrix}, q_2 = \begin{bmatrix}\frac{1}{\sqrt{3}}\\ \frac{1}{\sqrt{3}}\\ -\frac{1}{\sqrt{3}}\end{bmatrix}$

So, we just need to find the R matrix.

Since

$$b_1 = r_{11}q_1 \Rightarrow \begin{bmatrix}1\\0\\1\end{bmatrix} = r_{11}\begin{bmatrix}\frac{1}{\sqrt{2}}\\ 0\\ \frac{1}{\sqrt{2}}\end{bmatrix} \Rightarrow r_{11} = \sqrt{2}$$

$$r_{12} = q_1^* b_2 \Rightarrow r_{12} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}$$

$$b_2 = r_{12}q_1 + r_{22}q_2 \Rightarrow \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} + r_22 \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = r_{22} \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} \end{bmatrix} \Rightarrow r_{22} = \sqrt{3}$$

Therefore $B = \hat{Q}\hat{R} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} \sqrt{2} & \sqrt{2} \\ 0 & \sqrt{3} \end{bmatrix}$ is the reduced QR-factorization.

Now for the full QR, we need to add an extra column to Q that makes it unitary. By looking a Q, we can see that q_{11} and q_{33} have to be the same value, with opposite signs, and q_{23}, is the sum of their absolute values. So let $q_{13} = 1, q_{23} = 2, andq_{33} = -1$. Finally, we just need to normalize the vector q_{33}.

$$B = QR = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{6}} \end{bmatrix} \begin{bmatrix} \sqrt{2} & \sqrt{2} \\ 0 & \sqrt{3} \\ 0 & 0 \end{bmatrix}, \text{ is the full QR-factorization}$$

Problem 4 (5 pts, MATLAB). Write two MATLAB functions, one that implements the Classical Gram-Schmidt algorithm (Algorithm 7.1) and one that implements the Modified Gram-Schmidt algorithm (Algorithm 8.1), to compute reduced QR decomposition $A = \hat{Q}\hat{R}$ of any $m \times n$ matrix A, with $m \geq n$. Run your functions on the matrix:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1.0001 & 1 & 1 \\ 1.0001 & 1.0001 & 1 \end{bmatrix}.$$

Then run MATLAB's internal QR Factorization algorithm with the command `[Q, R] = qr(A)`. Compare the output of the three approaches, using the command `format long` to see more decimal places. In particular, for the $\hat{Q}$ obtained by each approach, check how close $\hat{Q}$ is to being unitary by displaying the entries of $\hat{Q}^*\hat{Q}$. Discuss what you observe. Include a printout/screenshot of your code and the requested output in your writeup.

Solution 4. First, let's look at the code and the output.

```
A = [1 1 1; 1.0001 1 1; 1.0001 1.0001 1];
format long
```

```
[Qc, Rc] = classicQR(A);
[Qm, Rm] = modifiedQR(A);
[Q,R] = qr(A);

disp(The Q matrix for the three approaches:  )
disp(Using Matlab’s QR: )
disp(Q)
disp(Using the Classic Gram Schmidt:  )
disp(Qc)
disp(Using the Modified Gram Schmidt:  )
disp(Qm)

disp(The R matrix for the three approaches:  )
disp(Using Matlab’s QR: )
disp(R)
disp(Using the Classic Gram Schmidt:  )
disp(Rc)
disp(Using the Modified Gram Schmidt:  )
disp(Rm)

disp(Now, check if they are unitary:  )
disp(Using Matlab’s QR: )
disp(Q'*Q)
disp(Using the Classic Gram Schmidt:  )
disp(Qc'*Qc)
disp(Using the Modified Gram Schmidt:  )
disp(Qm'*Qm)

function [Q, R] = classicQR(A)
    for j=1:size(A)
        vj = A(:,j);
        for i=1:j-1
            R(i,j) = Q(:,i)'*A(:,j);
            vj=vj-R(i,j)*Q(:,i);
        end
        R(j,j)=norm(vj);
        Q(:,j)=vj/R(j,j);
    end
end

function [Q, R] = modifiedQR(A)
    for i=1:size(A)
```

```
        V(:,i) = A(:,i);
    end
    for i=1:size(A)
        R(i,i) = norm(V(:,i));
        Q(:,i) = V(:,i)/R(i,i);
        for j=i+1:size(A)
            R(i,j) = Q(:,i)'*V(:,j);
            V(:,j) = V(:,j)-R(i,j)*Q(:,i);
        end
    end
end
%-------------------Output------------------------
%>> hw4_q4
%The Q matrix for the three approaches:
%Using Matlab's QR:
%  -0.577311781096138   0.408241484793693  -0.707142133874828
%  -0.577369512274247  -0.816482973671955   0.000000000001479
%  -0.577369512274247   0.408282308944734   0.707071426730676
%Using the Classic Gram Schmidt:
%   0.577311781096138   0.408241484795121   0.707142138976441
%   0.577369512274247  -0.816482973672249  -0.000000010199954
%   0.577369512274247   0.408282308942718  -0.707071421628554
%Using the Modified Gram Schmidt:
%   0.577311781096138   0.408241484795121   0.707142133875418
%   0.577369512274247  -0.816482973672249   0.000000000002093
%   0.577369512274247   0.408282308942718  -0.707071426730087
%The R matrix for the three approaches:
%Using Matlab's QR:
%  -1.732166279547088  -1.732108542595860  -1.732050805644633
%                   0   0.000081648297367   0.000040820066473
%                   0                   0  -0.000070707142673
%Using the Classic Gram Schmidt:
%   1.732166279547088   1.732108542595860   1.732050805644633
%                   0   0.000081648297367   0.000040820065589
%                   0                   0   0.000070707142673
%Using the Modified Gram Schmidt:
%   1.732166279547088   1.732108542595860   1.732050805644633
%                   0   0.000081648297367   0.000040820066473
%                   0                   0   0.000070707142673
%Now, check if they are unitary:
%Using Matlab's QR:
%   1.000000000000000                   0                   0
%                    0   1.000000000000000                   0
%                    0                   0   1.000000000000000
%Using the Classic Gram Schmidt:
%   1.000000000000000  -0.000000000000510   0.000000000002743
%  -0.000000000000510   1.000000000000000   0.000000012495113
%   0.000000000002743   0.000000012495113   1.000000000000000
%Using the Modified Gram Schmidt:
%   1.000000000000000  -0.000000000000510   0.000000000002743
%  -0.000000000000510   1.000000000000000                   0
%   0.000000000002743                   0   1.000000000000000
```

It is possible to see that $\hat{Q}$ for all the approaches are almost the same for the three approaches, if we look at their absolute values. The one generated by the Matlab's function has some negative values, which is different than the others, however, the $\hat{R}$ matrix makes up for it, since we have negatives values there too, while the other approaches have positive.

Now, when we check to see weather the matrices are unitary or not, we can see that, the Matlab's function is the only one that gives a perfect unitary matrix. Both the Modified Gram Schmidt and the Classical Gram Schmidt give matrices that are very close to being unitary, but due to rounding errors and the instability of the algorithms, the matrices are not exactly unitary.

Furthermore, if we compare just the modified approach with the classical approach, we can see that the modified one, does the job a little bit better, since the entries in $(q^*q)_{23}$ and $(q^*q)_{32}$ are exactly 0, while in the classical approach they are not. Which was expected, since the modified version is a little more stable than the classical one.

Problem 5 (5 pts, MATLAB)**.** This problem extends "Experiment 1" on pg. 64 of T-B.

(a) First, read through the experiment and run the two indicated code blocks. Include the generated plot in your write-up.

(b) Approximate the function $f(x) = \cos(\pi x)$ on the interval $[-1, 1]$ as a linear combination of the first four Legendre polynomials by projecting the vector `y = cos(pi*x);` onto the range of `A` (Hint: the matrix `Q` defined in the first code block might be helpful for this). Plot both $f(x)$ and its approximation by Legendre polynomials on the same graph. Repeat this experiment for the first six Legendre polynomials, and again for the first eight Legendre polynomials.

(c) Redo part (b) for the "Heaviside function" given by

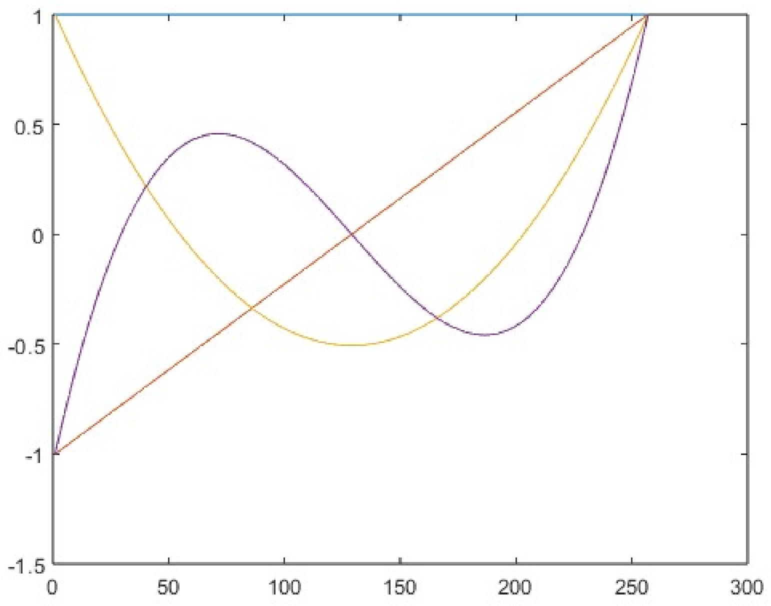
$$h(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

In MATLAB, you can use the built-in function `heaviside(x)`. What do you observe in terms of the ability of Legendre polynomials to approximate $h(x)$?

Solution 5. .

a-)

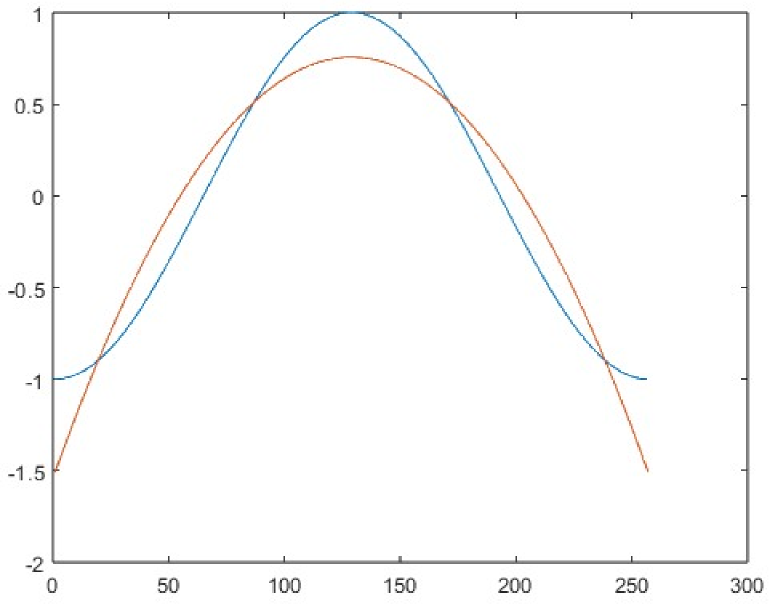
```
x = (-128:128)'/128;
A = [x.^0 x.^1 x.^2 x.^3];
[Q,R] = qr(A,0);

scale = Q(257, :);
Q = Q*diag(1 ./scale);
plot(Q)
```



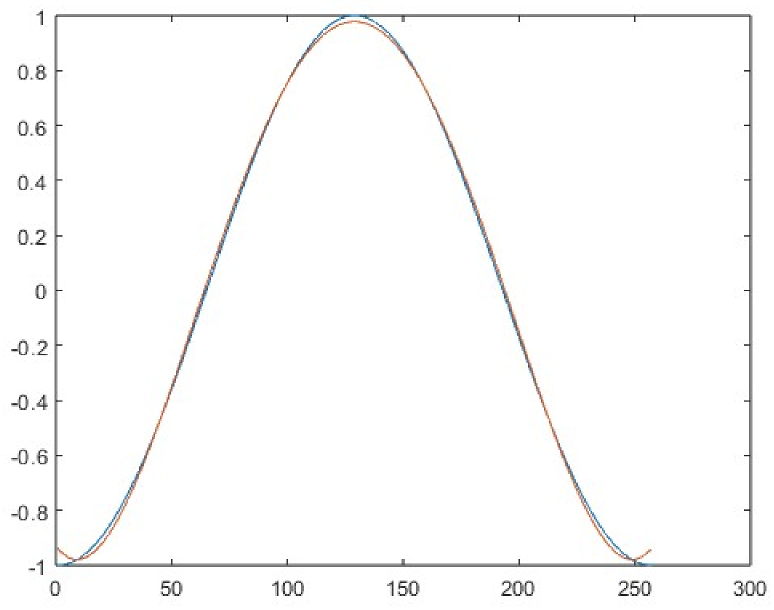
b-)

```
%% 4 polynomials
x = (-128:128)'/128;
A = [x.^0 x.^1 x.^2 x.^3];
[Q,R] = qr(A,0);

scale = Q(257, :);
Q = Q*diag(1 ./scale);

P = Q*(Q'*Q)^(-1)*Q';
y = cos(pi*x);
py = P*y;
figure()
plot(y)
```

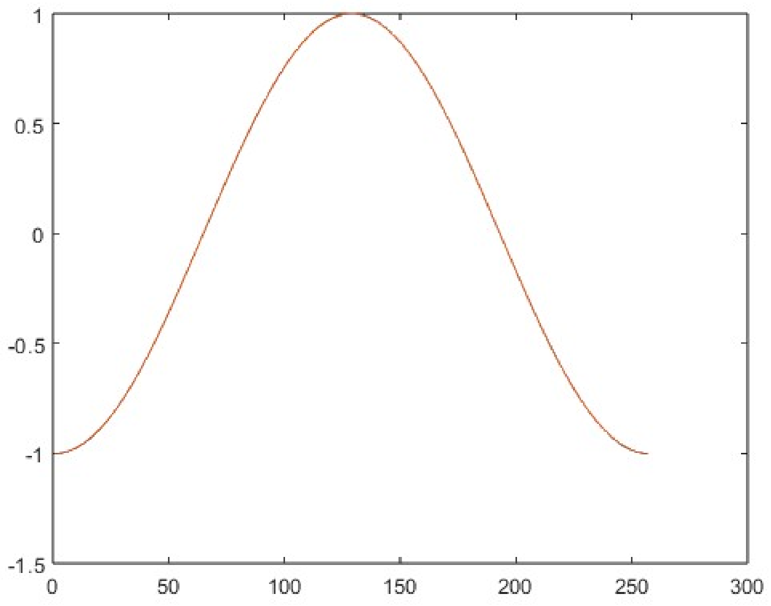
```
hold on
plot(py)
hold off
```



```
%% 6 polynomials

x = (-128:128)'/128;
A = [x.^0 x.^1 x.^2 x.^3 x.^4 x.^5];
[Q,R] = qr(A,0);

scale = Q(257, :);
Q = Q*diag(1 ./scale);

P = Q*(Q'*Q)^(-1)*Q';
y = cos(pi*x);
py = P*y;
figure()
plot(y)
hold on
plot(py)
hold off
```



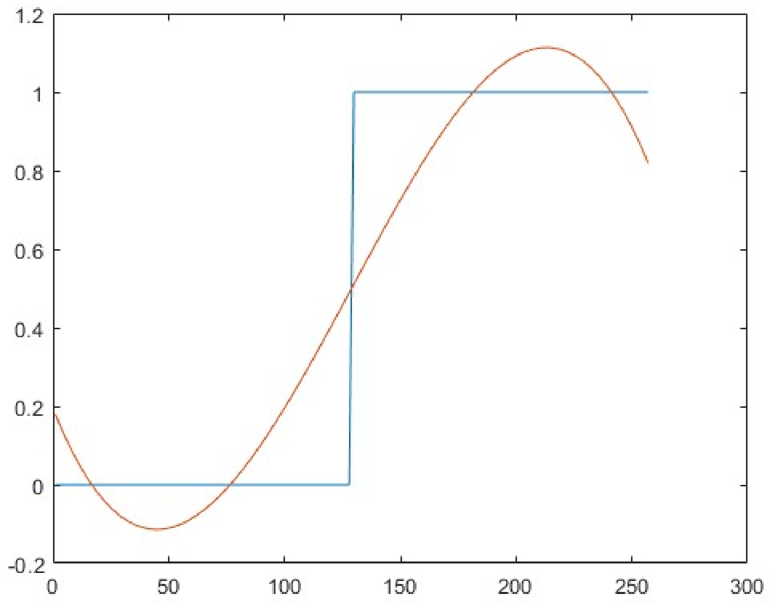
```
%% 8 polynomials

x = (-128:128)'/128;
A = [x.^0 x.^1 x.^2 x.^3 x.^4 x.^5 x.^6 x.^7];
[Q,R] = qr(A,0);

scale = Q(257, :);
Q = Q*diag(1 ./scale);

P = Q*(Q'*Q)^(-1)*Q';
y = cos(pi*x);
py = P*y;
figure()
plot(y)
hold on
plot(py)
hold off
```



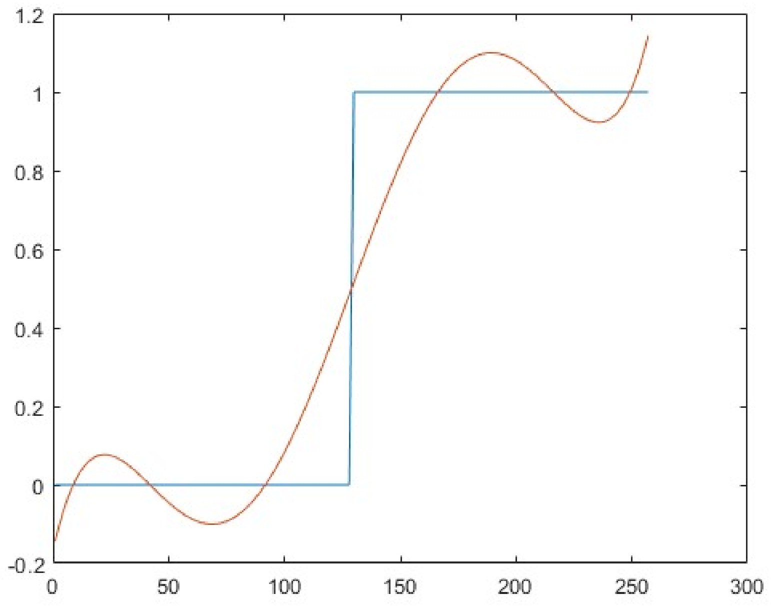
It is possible to see that the 4 Legendre polynomials, does not approximate the function that well. However, the 6 Legendre polynomials already does a great job, and using 8 perfectly approximates it.

c-)

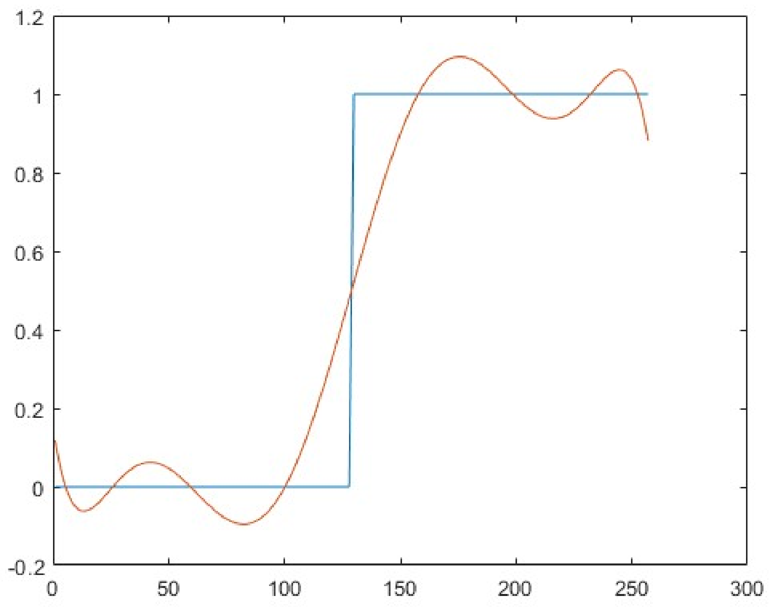
```
%% 4 polynomials
x = (-128:128)'/128;
A = [x.^0 x.^1 x.^2 x.^3];
[Q,R] = qr(A,0);

scale = Q(257, :);
Q = Q*diag(1 ./scale);

P = Q*(Q'*Q)^(-1)*Q';
y = heaviside(x);
py = P*y;
figure()
plot(y)
hold on
plot(py)
hold off
```



```
%% 6 polynomials

x = (-128:128)'/128;
A = [x.^0 x.^1 x.^2 x.^3 x.^4 x.^5];
[Q,R] = qr(A,0);

scale = Q(257, :);
Q = Q*diag(1 ./scale);

P = Q*(Q'*Q)^(-1)*Q';
y = heaviside(x);
py = P*y;
figure()
plot(y)
hold on
plot(py)
hold off
```



```
%% 8 polynomials
x = (-128:128)'/128;
A = [x.^0 x.^1 x.^2 x.^3 x.^4 x.^5 x.^6 x.^7];
[Q,R] = qr(A,0);

scale = Q(257, :);
Q = Q*diag(1 ./scale);

P = Q*(Q'*Q)^(-1)*Q';
y = heaviside(x);
py = P*y;
figure()
plot(y)
hold on
plot(py)
hold off
```



In this case, trying to approximate $h(x)$ with Legendre polynomials does not approximate it well, not even by increasing the number of polynomials we are using. I suspect that the main reason for this is because the heaviside function is not continuous, while all the Legendre polynomials are.