Numerical Linear Algebra

Least square approximation problem

PhD Program in Mathematics and Computer Science

Patricia Díaz de Alba

Contents

1. Introduction

In this work we will try to get the best polynomial approximation of a known function $f(x)$ by a least squares method. It means, we will determinate the polynomial of degree n which minimize the following norm

$$\min_{p_n \in \Pi_n} \|p_n - f\|_2$$

It would be better the infinity norm, but much more difficult so we will use 2-norm.

The method of least squares is a standard approach to the approximated solution of overdetermined systems, i.e., sets of equations in which there are more equations than unknowns:

$$Ax = b$$

where A is a matrix m x n $(m > n)$, $b \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$.

So, we have to solve

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2$$

to get the approximated solution.

We will solve our system by the QR factorization and Cholesky factorization on normal equations.

In general, the QR factorization is better than *Cholesky* on normal equations because the condition number in normal equations $(K_2(A^TA))$ is much bigger than the condition number in QR factorization $(K_2(A))$.

$$K_2(A)^2 = K_2(A^TA)$$

In the next section we will choose one function $f(x)$ and we will construct its best polynomial approximation by least squares solving the system by QR factorization and Cholesky on normal equations.

2. Least squares approximation problem

There are some cases in which is better to do a least squares approximation to get the best polynomial approximation of a function than an interpolation because of the data errors.

In those cases, we get $\{x_0, x_1, ..., x_m\}$, $m+1$ points in the abscissa, and $\{y_0, y_1, ..., y_m\}$ which are the values of the function $f(x)$ in every point, $y_i = f(x_i)$.

Fixed a natural number $n \leq m$, we want to determine $p_n^*(x)$ of degree n which minimize

$$\min_{p_n \in \Pi_n} \|p_n - f\|_2^2$$

Using the canonical basis, we have

$$p_n(x_i) = \sum_{j=0}^{n} a_j x_i^j = (Xa)_i,\ i = 0, ..., m,$$

where $a = (a_0, ..., a_n)^T \in \mathbb{R}^{n+1}$ is the polynomial coefficients vector and X is a Vandermonde matrix of dimensions $(m+1)$ x $(n+1)$

$$X = \begin{pmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & ... & x_m^n \end{pmatrix}$$

Then,

$$\|p_n - f\|_2^2 = \sum_{i=0}^{m} [(Xa)_i - y_i]^2 = \|Xa - y\|_2^2$$

At this moment we can solve

$$Xa = y,$$

by Cholesky and QR factorization.

1. Linear systems and Cholesky factorization on normal equations with Matlab

The Cholesky factorization is a decomposition of a positive-definite matrix, A, into a product $A = R^T R$, where R is an upper triangular matrix.

In this section we can see the Matlab code called ***factcholesky*** to solve a linear system by this decomposition. It also measures performance with *tic toc* and gives the norm $\|Ax - b\|$ and the condition number, K.

```
function [x] = factcholesky(A,b)

tic

R=chol(A'*A);
w=R'\(A'*b);
x=R\w;

toc

NormC=norm(A*x-b)

K=cond(A'*A)

end
```

2. Linear systems and QR factorization with Matlab

The QR factorization is a decomposition of a matrix m x n, A, into a product $A = QR$ of an orthogonal matrix Q m x n and an upper triangular matrix R with the same dimensions of A.

2.1 Solving a linear system with *qr*

In this section, we will show the Matlab code called ***factQR*** to solve a linear system $Ax = b$ using the Matlab function *qr*,

```
function [x] = factQR(A,b)

tic

[rows_A,col_A]=size(A);
[Q,R]=qr(A);
c=Q'*b;
R1=R(1:col_A,1:col_A);
c1=c(1:col_A);
x=R1\c1;

toc

NormQR=norm(A*x-b)

K=cond(A)

end
```

This code also measures performance with *tic toc* and gives us the norm $\|Ax - b\|$ and the condition number, K.

2.2 Solving a linear system constructing the Householder QR factorization

We will show a Matlab code to solve a linear system by QR factorization but, in this case, we will construct the algorithm to get Q and R.

Before that, we have to get the Householder matrix and for that we have the following function, ***Householdermatrix***

```
function [H] = Householdermatrix(x)

[m,n]=size(x);
e1=[1;zeros(1,m-1)'];
sigma=norm(x);
k=-sign(x(1))*sigma;
lambda=sqrt(2*sigma*(sigma+abs(x(1))));
w=(x-k*e1)./lambda;
H=eye(m)-2*w*w';

end
```

Once defined this function we can write the Matlab code to get Q and R, called ***FactorizationQR***

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

[m n]=size(A);
Q=eye(m);

for i=1:n

  h=Householdermatrix(A(i:m,i));

  H=eye(m);
  H(i:m,i:m)=h;
  Q=Q*H;
  A=H*A;

end

R=A;

end
```

After that, we can create a function which solves a linear system using ***FactorizationQR***,called ***factQR2***

```
function [x] = factQR2(A,b)

tic
```

```
[rows_A,col_A]=size(A);
[Q,R]=FactorizationQR(A);
c=Q'*b;
R1=R(1:col_A,1:col_A);
c1=c(1:col_A);
x=R1\c1;

toc

NormQR2=norm(A*x-b)

end
```

At this moment, we have two functions which give us QR factorization. One of them is a Matlab function, *qr*, and the another one, *FactorizationQR*, which is the one we constructed.

Now, we are going to write another one multiplying QH and HA in another way with less number of operations. Because if we multiply two matrices in this way

$HA = (I - 2ww^T)A$

we are doing n^3 operations, however if we multiply in this way

$HA = (I - 2ww^T)A = A - 2w(w^TA) = A - 2wv^T$, $v = A^Tw$

we are doing only n^2 operations. So this method that we are going to write should be faster.

Firstly, we have to create a function, ***Householdermatrix2***, which gives us the Housholder matrix and the vector w,

```
function [w,H] = Householdermatrix2(x)

[m,n]=size(x);
e1=[1;zeros(1,m-1)'];
sigma=norm(x);
k=-sign(x(1))*sigma;
lambda=sqrt(2*sigma*(sigma+abs(x(1))));
w=(x-k*e1)./lambda;
H=eye(m)-2*w*w';

end
```

Now, we write ***FactorizationQR2*** which calculates Q and R with the function above,

```
function [Q,R]=FactorizationQR2(A)

[m n]=size(A);
Q=eye(m);
```

```
if m>n
    p = n;
else
    p = n-1;
end

for i=1:p

  w=Householdermatrix2(A(i:m,i));

  v=A(i:m,i:n)'*w;
  A(i:m,i:n) = A(i:m,i:n) - 2*w*v';

  u=Q(:,i:m)*w;
  Q(:,i:m)=Q(:,i:m)-2*u*w';


end

R=triu(A);

end
```

Finally, we solve a linear system with ***FactorizationQR2*** with the next function, ***factQR3***,

```
function [x] = factQR3(A,b)

tic

[rows_A,col_A]=size(A);
[Q,R]=FactorizationQR2(A);
c=Q'*b;
R1=R(1:col_A,1:col_A);
c1=c(1:col_A);
x=R1\c1;

toc

NormQR3=norm(A*x-b)

end
```

In the next section we will show an example with every functions we did.

3. Matlab code to solve a least squares approximation problem

Once defined all the functions above, we can make a Matlab code, ***Approssmq***, to solve a least squares approximation problem solving the linear system with every QR

factorizations and Cholesky factorization.

We have chosen $f(x) = e^x cos(x)^2$, 100 points and the degree of p_n, 20. We have also chosen $ds = 10^{-2}$ to get a perturbation.

```
N=100;
n=20;
ds = 1e-2;
t=linspace(-2,2,N+1)';
fun=@(x) exp(x).*cos(x).^2;

X = ones(N+1,n+1);

for i=2:n+1

    X(:,i)= t.*X(:,i-1);

end

ye=fun(t);
y = ye+ds*randn(N+1,1);

aC=factcholesky(X,y)
aQR1=factQR(X,y)
aQR2=factQR2(X,y)
aQR3=factQR3(X,y)

x=linspace(-2,2,200);
f=fun(x);

figure(1)
p1=polyval(fliplr(aC'),x);
plot(x,p1,x,f,'r--',t,y,'o')

figure(2)
p2=polyval(fliplr(aQR1'),x);
plot(x,p2,x,f,'g--',t,y,'o')

figure(3)
p3=polyval(fliplr(aQR2'),x);
plot(x,p3,x,f,'b--',t,y,'o')

figure(4)
p4=polyval(fliplr(aQR3'),x);
plot(x,p4,x,f,'p--',t,y,'o')
```

This code gives us the solution of the linear system $Xa = y$ by Cholesky and every QR factorizations (aC, aQR1, aQR2 and aQR3),

```
Approssmq
```

```
Elapsed time is 0.009747 seconds.

NormC =

    0.0840


K =

   4.5700e+28


aC =

    1.0016
    1.0335
   -0.5348
   -1.2158
    0.3598
    1.7839
   -1.8695
   -3.2130
    3.5732
    3.5069
   -3.6033
   -2.2484
    2.1534
    0.8692
   -0.7832
   -0.1991
    0.1700
    0.0249
   -0.0202
   -0.0013
    0.0010

Elapsed time is 0.003828 seconds.

NormQR =

    0.0840


K =

   1.0186e+09


aQR1 =
```

```
    1.0016
    1.0335
   -0.5349
   -1.2153
    0.3606
    1.7816
   -1.8720
   -3.2086
    3.5772
    3.5021
   -3.6069
   -2.2453
    2.1554
    0.8680
   -0.7839
   -0.1989
    0.1701
    0.0248
   -0.0202
   -0.0013
    0.0010

Elapsed time is 0.039606 seconds.

NormQR2 =

    0.0840


aQR2 =

    1.0016
    1.0335
   -0.5349
   -1.2153
    0.3606
    1.7816
   -1.8720
   -3.2086
    3.5772
    3.5021
   -3.6069
   -2.2453
    2.1554
    0.8680
   -0.7839
   -0.1989
    0.1701
```

```
    0.0248
   -0.0202
   -0.0013
    0.0010

Elapsed time is 0.037391 seconds.

NormQR3 =

    0.0840


aQR3 =

    1.0016
    1.0335
   -0.5349
   -1.2153
    0.3606
    1.7816
   -1.8720
   -3.2086
    3.5772
    3.5021
   -3.6069
   -2.2453
    2.1554
    0.8680
   -0.7839
   -0.1989
    0.1701
    0.0248
   -0.0202
   -0.0013
    0.0010
```

It also shows every graphic solutions,

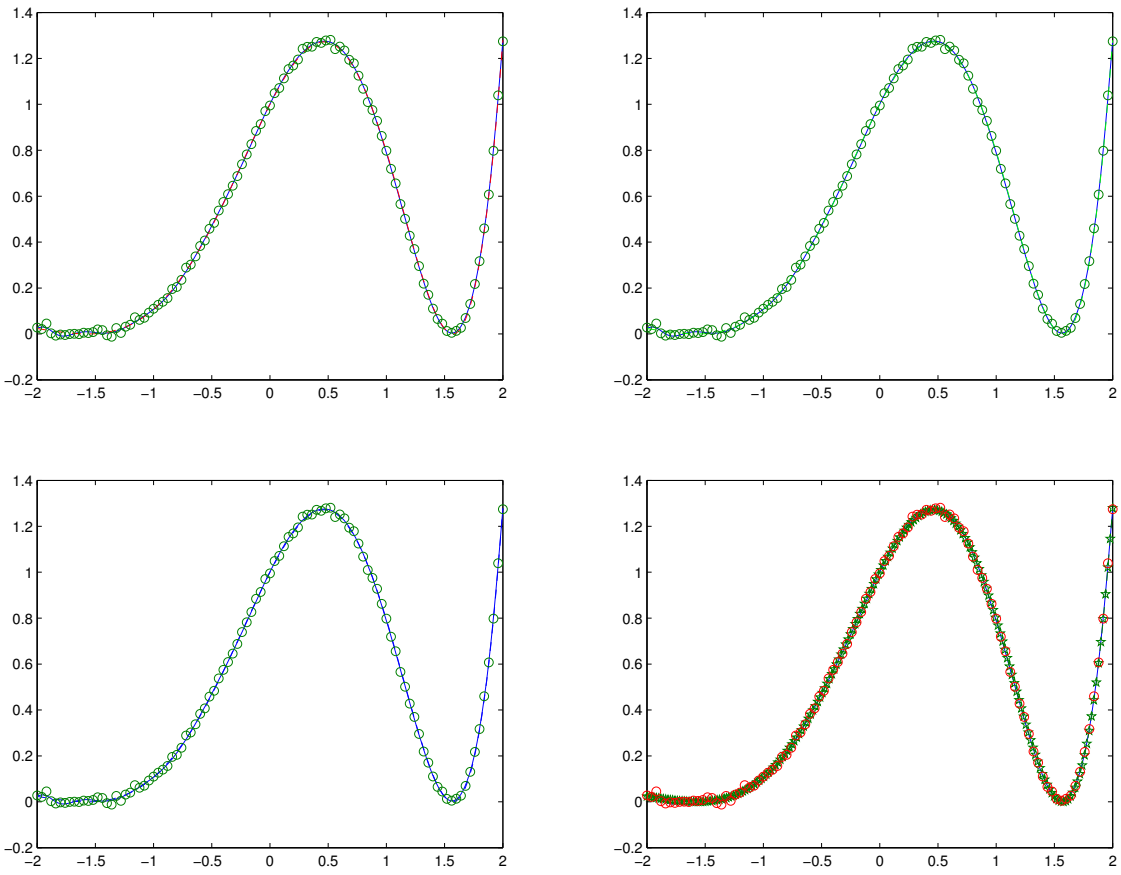

Figure 1: Approximation polynomial of f ($n = 20$, $ds = 10^{-2}$) by Cholesky and every QR factorizations

The first one represents the function $f(x) = e^x cos(x)^2$ with a red dashed line, its polynomial approximation, $p1$, by the function we constructed before ***factcholesky*** with a blue solid line and the values of f in every component of the vector t with green circles.

The second one also represents the function f with a green dashed line, the values of f in every component of the vector t with green circles and its polynomial approximation but in this case we solved the linear system by the Matlab function ***factQR*** that we constructed before.

The third one shows the function f with a blue dashed line, the values of f in every component of the vector t with green circles and its polynomial approximation but in this case we solved the linear system by the Matlab function ***factQR2*** that we constructed before.

And the last one represents the function f, the values of f in every component of the vector t with green circles and its polynomial approximation but in this case we solved the linear system by the Matlab function ***factQR3***.

All of them with $ds = 10^{-2}$.

In this example we can't appreciate a big difference between them because the degree (n=20) is not so high but, in general, QR factorization is better than Cholesky factorization.

If the perturbation is bigger, $ds = 10^{-1}$, we can see that the approximation is worse.

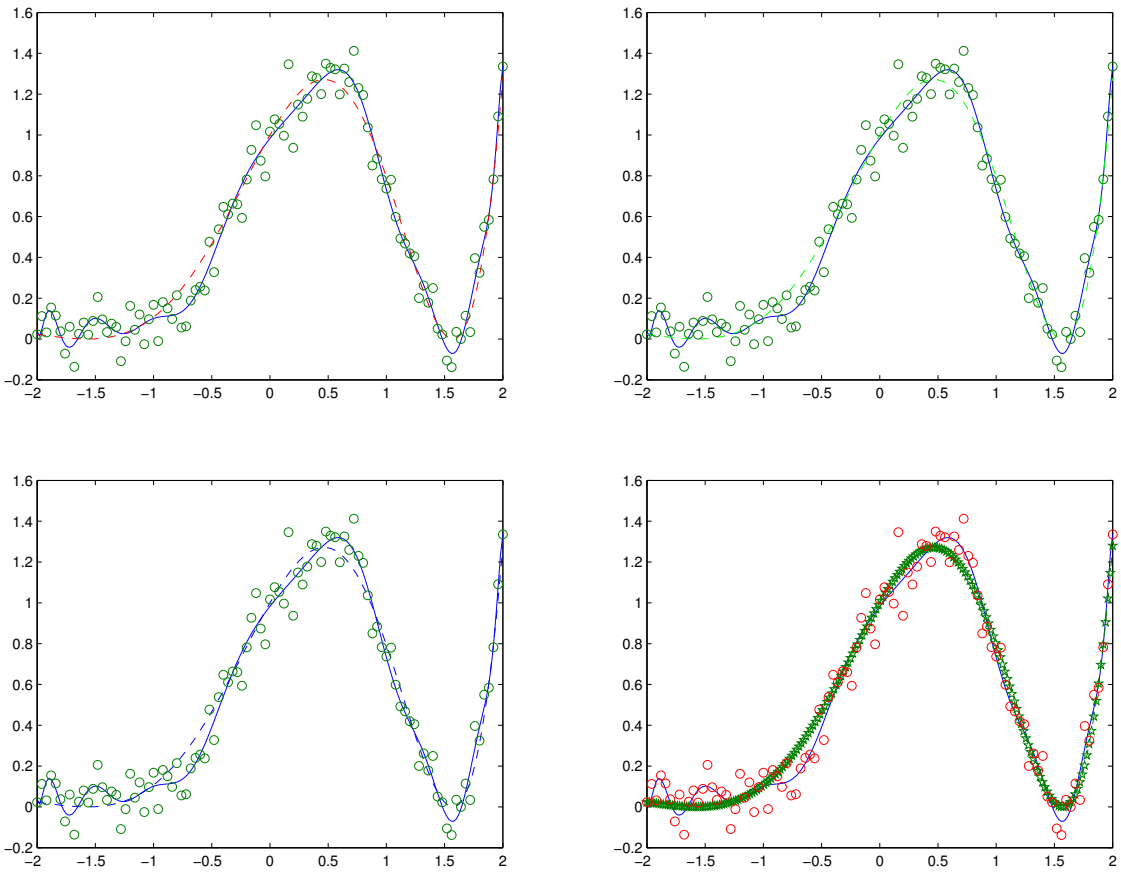

Figure 2: Approximation polynomial of f ($n = 20$, $ds = 10^{-2}$) by Cholesky and every QR factorizations

After this example, we can affirm that the QR factorization is better than Cholesky on normal equations because of the condition number although the difference between the norms in every algorithm is not big. We can also say that ***factQR3*** is faster than ***factQR2*** because of the multiplying way and when the perturbation is bigger, the solution of the problem is worse.

3. Example

In this section we will choose a random matrix A and a random vector b to solve the linear system $Ax = b$ by Cholesky and every QR factorization that we constructed before to compare the time spent in every factorization with a graphic.

The matlab code is the following:

```
vn=[50:10:1000];

for i=1:length(vn)

    n=vn(i);
    A=rand(n);
    b=rand(n,1);

    tic
    xC=factcholesky(A,b);
    time1(i)=toc;

    tic
    xQR=factQR(A,b);
    time2(i)=toc;

    tic
    xQR2=factQR2(A,b);
    time3(i)=toc;

    tic
    xQR3=factQR3(A,b);
    time4(i)=toc;
end

semilogy(vn,[time1' time2' time3' time4'])
legend('Cholesky','QR','QR2','QR3','Location','Best');
```

And this code shows us the following graph,

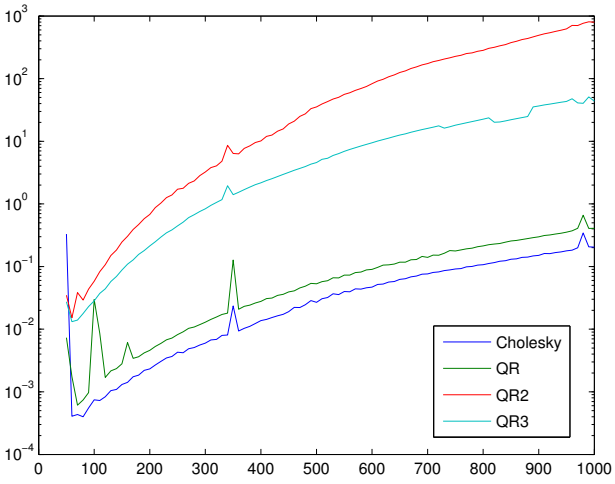

Figure 3:

At the beginning, we can see that Cholesky factorization and QR factorization have a strange behavior because n is so small, but when n is bigger, they are similar although QR factorization spends more time to solve a linear system than Cholesky.

On the other hand, QR2 and QR3 factorizations are very similar but we can see that QR3 factorization spends less time to solve it than QR2 factorization because of the multiplying way.