MAT3110 Compulsory assignment 1

Soran Hussein Mohmmed / soranhm

September 2018

QR factorization and Cholesky factorization

In this assigment i will consider the linear system of equations A$\mathbf{x}$ = $\mathbf{b}$,

$$\begin{bmatrix} a_{1,1} & a_{1,2} & ... & a_{1,m} \\ a_{2,1} & a_{2,2} & ... & a_{2,m} \\ . & . & ... & . \\ . & . & ... & . \\ . & . & ... & . \\ a_{n,1} & a_{n,2} & ... & a_{n,m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_n \end{bmatrix}$$

Since there are more equations than unknowns, I cannot usually solve this system, but I can instead find the vector minimizes ||Ax - b||$_2$, which is known as the least squares method. I will look at 2 different data sets. I can change ϵ to make noise or set $\epsilon = 0$ to remove nois, I will later see in both cases $\epsilon = 0$ makes the approximation a bit closer to the exact solution. I will use a so-called Vandermonde matrix:

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^{m-1} \\ 1 & x_2 & x_2^2 & ... & x_2^{m-1} \\ . & . & ... & . & . \\ . & . & ... & . & . \\ . & . & ... & . & . \\ 1 & x_n & x_n^2 & ... & x_n^{m-1} \end{bmatrix}$$

with $\mathbf{x} = [c_1, c_2, ..., c_m]$ as the the vector of coefficients of p, when p(x) = $\sum_{j=1}^{m} c_j x^{j-1}$. And $\mathbf{b} = [y_1, y_2, ..., y_n]^T$. First I will find $\mathbf{x}$ with help of QR factorization and back substitution, and the try with Cholesky factorization and forward substitution.

1: QR factorization

I'm using given function in Matlab to find Q and R with respect of given A matrix given above. were Q is an nxn orthogonal matrix and R is an nxm upper triangular matrix.

$$A\mathbf{x} = \mathbf{b} \rightarrow QR\mathbf{x} = \mathbf{b} \rightarrow R\mathbf{x} = Q^T\mathbf{b}$$

Now I have that R = R(1:m,:) because the rest is zero and it will not fit into the backward substitution, and b_f in backward substitution will now be: $b_f = Q^T\mathbf{b}$, were $\mathbf{b}$ is given above as $\mathbf{y}$. Now i put this into the matlab function backward and the $\mathbf{x}$ that i get out is the coefficients infront of the polynomial, and the degree is choosen byn m. example: for m = 3 we get f(x) = $c_1 + c_2 * x + c_3 * x^2$. After calculating this with both given $\mathbf{y}$ and m, i will get the plots:

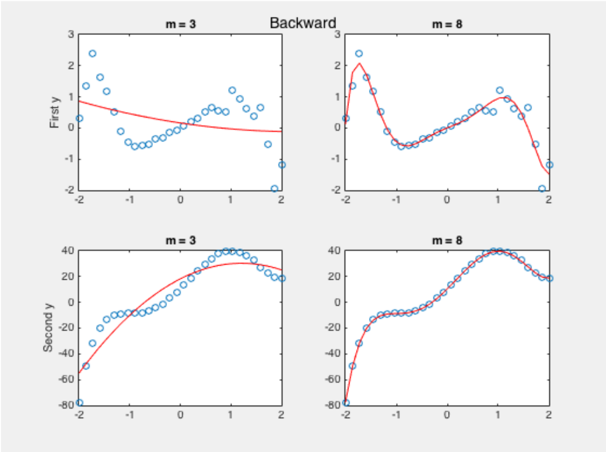

Figure 1: All the cases with both $\mathbf{y}$ and m

2: Cholesky factorization

Now I'm using the normal equation: $A^TA\mathbf{x} = A^T\mathbf{b}$, with $B = A^TA$ who is symmetric and positive definite matrix and Cholesky factorization RR^T of B, and implement forward substitution. After calculating this with both given $\mathbf{y}$ and m, i will get the plots:

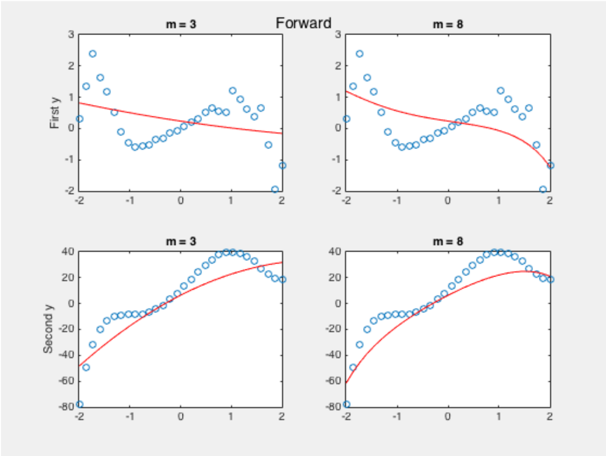

Figure 2:

3: Conclusion

I see that the plots i get is the same as the given one in the assignment, atleast for the backward substitution one, and if we look at the plots with m = 8 we will see that the one with backward substitution is more accurate then the forward substitution. When running the program with no noise **y** will be more spread and the approximations will try to follow it, but when we use a large ϵ we will get a lot more chaotic y values and the plots gives no explanation. We can look at the point of view of conditioning given below to see more accurate reason of this. For values of x close to 1 (over 1), K(x) can get large. For example if x = 1.000001, then K(1.000001) = $1.000001 * 10^6$ and thus the error will increase with a factor of about 10^6. This is what happens in our case when using forward with m = 8 (as we see in the plots).

$$K(x) = |\frac{x * f'(x)}{f(x)}|$$

```
>> main

K_b1 =

    0.8899

K_b2 =

    0.8866

K_f1 =

    0.6574

K_f2 =

    1.0132
```

Figure 3: conditioning

```
 function [x]=Backward(A,b)
% Solves the upper triangular system of equations Ax = b % A input argument, square upper tria
% b input argument
% x solution
[n,m]=size(A); % finding the size of A
if n ~= m
    disp('input is not a square matrix');
    return;
end
if size(b,1)~= n
    disp('input dimensions do not match');
    return;
end
x = zeros(n,1); % initialise x to the same dimension
if abs(A(n,n)) > 1e−12 % not comparing to zero because of possible
                       % rounding errors
    x(n) = b(n)/A(n,n); % solve for the last element of x
else
    disp('input singular'); % A is singular if any of the diagonal
                            % elements are zero
    return;
end
for k=n:−1:1 % the loop considers one row after the other backwards
    if abs(A(k,k))>1e−12 % not comparing to zero because of possible
                         % rounding errors
        temp = 0;
        for j=n:−1:k+1
            temp = temp + A(k,j) * x(j); % Multiply the elements of
                                         % the k?th row of A after the
                                         % diagonal by theelements of x
                                         % already calculated
        end
        x(k) = (b(k)−temp)/A(k,k); % solve for the k?th element of x
    else
        disp('input singular'); % A is singular if any of the diagonal
                                % elements are zero
         return;
    end
end
```

```
function [x] = Forward(A,b)
% Solve the lower triangular system of equations Ax = b
% A input argument, square lower triangular matrix
% b input argument
% x solution

[n,m] = size(A); % finding the size of A
if n ~= m
    disp('input is not a square matrix');
    return;
end
if size(b,1)~= n
    disp('input dimensions do not match');
    return;
end
x = zeros(n,1); % initialise x to the same dimension
if abs(A(n,n)) > 1e-12 % not comparing to zero because of possible
                       % rounding errors
    x(1) = b(1)/A(1,1); % solve for the first element of x
else
    disp('input singular'); % A is singular if any of the diagonal
                            % elements are zero
    return;
end
for k=2:n  % the loop considers one row after the other
    if abs(A(k,k))>1e-12 % not comparing to zero because of possible
                         % rounding errors
            temp = 0;
            for j=1:k-1
                temp = temp + A(k,j) * x(j); % Multiply the elements of
                                 % the k-th row of A before the
                                 % diagonal by theelements of x
                                 % already calculated
            end
            x(k) = (b(k)-temp)/A(k,k); % solve for the k-th element of x
    else
        error('input singular'); % A is singular if any of the
                                 % diagonal elements are zero
    end
end
```

```
function [L,D] = Cholesky(A)
% doing cholesky factorization on A
% A = LDL'
[n,m] = size(A);
L = zeros(n,n);
D = zeros(n,n);
B = A;
for k=1:n
    L(:,k) = B(:,k)/B(k,k);
    D(k,k) = B(k,k);
    B = B - D(k,k)*L(:,k)*transpose(L(:,k));
end
```

```
function [y,f,f_derv] = oblig1_1(E,y_oppg,A,x,m)
if E == 1 % calculating 1) with QR
    [Q,R] = qr(A);
    R1 = R(1:m,:);
    b_1 = inv(Q)*y_oppg';
    b = Backward(R1,b_1(1:m,:)); % constants
    if m == 3
        y = b(1) + x.*b(2) + x.^2*b(3) ;
    elseif m == 8
        y = b(1) + x.*b(2) + x.^2*b(3) + x.^3*b(4) + x.^4*b(5) +  x.^5*b(6) +
x.^6*b(7) + x.^7*b(8);
    end

    f = b(1) + x.*b(2) + x.^2*b(3);
    f_derv = b(2) + 2*x.*b(3);
elseif E == 2  % calculating 1) with Cholesky
    B = transpose(A)*A;
    [L,D] = Cholesky(B);
    b_f=A'*y_oppg';
    b = Forward(B,b_f);
    if m == 3
        y = b(1) + x.*b(2) + x.^2*b(3) ;
    elseif m ==8
        y = b(1) + x.*b(2) + x.^2*b(3) + x.^3*b(4) + x.^4*b(5) +  x.^5*b(6) +
x.^6*b(7) + x.^7*b(8);
    end
    % to find K
    f = b(1) + x.*b(2) + x.^2*b(3);
    f_derv = b(2) + 2*x.*b(3);

end
end
```

```
% Given values
n = 30; m1 = 3; m2 = 8;
start = -2;
stop = 2;
eps = 1;
rng(1);
x = linspace(start,stop,n);

%calculating two A corresponding to m's
A1 = zeros(n,m1);
for i=1:n
    for j=1:m1
        A1(i,j) = x(i)^(j-1);
    end
end

A2 = zeros(n,m2);
for i=1:n
    for j=1:m2
        A2(i,j) = x(i)^(j-1);
    end
end


r = rand(1,n)*eps;
y1 = x.*(cos(r+0.5*x.^3)+sin(0.5*x.^3));
y2 = 4*x.^5 - 5*x.^4 - 20*x.^3 + 10*x.^2 + 40*x + 10 + r;

% getting out the diffrent values from the function
[y_b1,f_b1,f_b1_derv] = oblig1_1(1,y1 ,A1,x,m1); % Backward , y1, m = 3
[y_b2,f_b2,f_b2_derv] = oblig1_1(1,y1 ,A2,x,m2); % Backward , y1, m = 8
[y_b3,f_b3,f_b3_derv] = oblig1_1(1,y2 ,A1,x,m1); % Backward , y2, m = 3
[y_b4,f_b4,f_b4_derv] = oblig1_1(1,y2 ,A2,x,m2); % Backward , y2, m = 8

% Ploting
figure(1)
suptitle('Backward')
subplot(221)
plot(x,y1,'o',x,y_b1,'r');
title('m = 3')
ylabel('First y')
subplot(222)
plot(x,y1,'o',x,y_b2,'r');
title('m = 8')
subplot(223)
plot(x,y2,'o',x,y_b3,'r');
title('m = 3')
ylabel('Second y')
subplot(224)
plot(x,y2,'o',x,y_b4,'r');
title('m = 8')...
```

```
[y_b1,f_f1,f_f1_derv] = oblig1_1(2,y1 ,A1,x,m1); % Forward , y1, m = 3
[y_f2,f_f2,f_f2_derv] = oblig1_1(2,y1 ,A2,x,m2); % Forward , y1, m = 8
[y_f3,f_f3,f_f3_derv] = oblig1_1(2,y2 ,A1,x,m1); % Forward , y2, m = 3
[y_f4,f_f4,f_f4_derv] = oblig1_1(2,y2 ,A2,x,m2); % Forward , y2, m = 8

figure(2)
suptitle('Forward')
subplot(221)
plot(x,y1,'o',x,y_f1,'r');
title('m = 3')
ylabel('First y')
subplot(222)
plot(x,y1,'o',x,y_f2,'r');
title('m = 8')
subplot(223)
plot(x,y2,'o',x,y_f3,'r');
title('m = 3')
ylabel('Second y')
subplot(224)
plot(x,y2,'o',x,y_f4,'r');
title('m = 8')

% Calculating K
K_b1 = abs((x.*f_b1_derv)/f_b1)
K_b2 = abs((x.*f_b3_derv)/f_b3)
K_f1 = abs((x.*f_f1_derv)/f_f1)
K_f2 = abs((x.*f_f3_derv)/f_f3)
```