Brian KYANJO

Homework #3

2. Weighted least squares.

a) Derive the normal Equations for Computing the solution x that minimizes $\|Ax-b\|_w$

$$\|y\|_w = \sqrt{y^T w y}$$

So,

$$\|Ax-b\|_w = \sqrt{(Ax-b)^T w (Ax-b)}$$

$$\|Ax-b\|_w^2 = (Ax-b)^T w (Ax-b)$$

$$\|Ax-b\|_w^2 = (x^T A^T w A x - x^T A^T w b - b^T w A x + b^T w b)$$

at minimum $\frac{d}{dx}\left(\|Ax-b\|_w^2\right) = 0$

$$\frac{d}{dx}\left(x^T A^T w A x - x^T A^T w b - b^T w A x + b^T w b\right) = 0$$

$$\frac{d}{dx}\left((A^T w A x)^T x\right) + x^T A^T \frac{d}{dx}(wAx) - \frac{d}{dx}\left((A^T w b)^T x\right)$$

$$-b^T w A = 0$$

$$x^T A^T w A + x^T A^T w A - b^T w^T A - b^T w A = 0$$

$$2x^T A^T w A = 2b^T w^T A \quad , \text{since } w^T = w$$

Taking Transpose both sides.

$$(x^T A^T w A)^T = (b^T w A)^T$$

$$A^T W A X = A^T W b$$

3) Unstable algorithm.

a) Compute the Conditional number of this function for $x = 0$ and some values close to zero.

$$K = \frac{\| J(x) \|}{\| f \| / \| X \|}$$

$$f(x) = \log(x+1)/x \Rightarrow J(x) = \frac{df}{dx} = \frac{v\frac{du}{dx} - u\frac{dv}{dx}}{v^2}$$

$$J(x) = \frac{d}{dx}\left(\frac{\log(x+1)}{x}\right)$$

$$J(x) = \frac{x\left(\frac{1}{x+1}\right) - \log(x+1)}{x^2}$$

$$K(x) = \frac{\|x\| \cdot \left\| \frac{x - (x+1)\log(x+1)}{x^2(x+1)} \right\|}{\| \log(x+1)/x \|}$$

$$K(x) = \left\| \frac{x - (x+1)\log(x+1)}{(x+1)\log(x+1)} \right\|$$

At ~~as~~ $x=0$, the Condition number $K(x=0)$ is undefined and function $f(x=0)$ is also undefined.

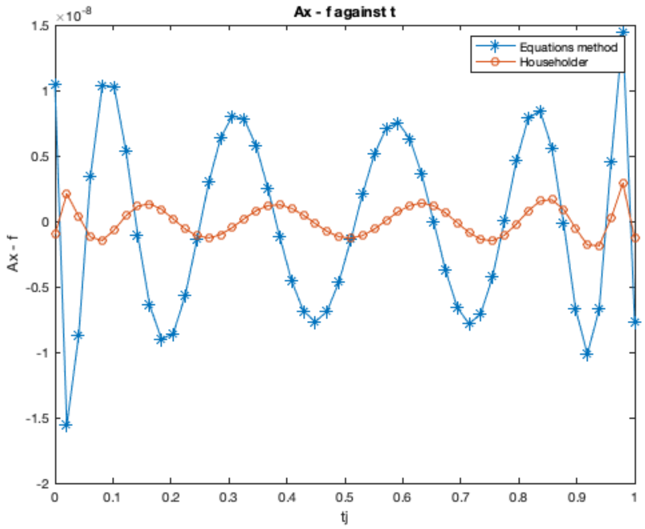
N0.1

```
clear all;
close all;
%m equally spaced points over [0,1]
m = 50; n=12;

% Vandermonde matrix t
t = zeros(m,n);
for i = 1:n
    for j = 1:m
        t(j,i) = ((j-1)/(m-1))^(n-i);
    end
end

%fliping the vandermonde matrix t to form A
A = fliplr(t);

%fuction f
tj = zeros(m,1);
for j = 1:m
    tj(j) = (j-1)/(m-1);
end

f = cos(4*tj);

format long
%(a). normal equations
x = (A'*A)\(A'*f);

%(b). QR decomposition using CGS
[q_c,r_c] = CGS(A); xc = r_c\(q_c'*f);

%(c). QR decomposition using MGS
[q_m,r_m] = MGS(A); xm = r_m\(q_m'*f);

%(d). QR decomposition using Householder
[v_h,r_h] = house(A); q_h = house2q(v_h);
x_h = r_h\(q_h'*f);

%(e). QR decomposition using inbuilt Householder
[q,r] = qr(A); xh = r\(q'*f);

%(f). QR decomposition using inbuilt svd
[u,s,v] = svd(A); xs = (u*s*v')\f;

Table = table(x,xc,xm,x_h,xh,xs, 'VariableNames',{'Normal equation','CGS','MGS','Householder','Builtin function','SVD'})

%Differences and Similarities
fprintf('The Normal equation and the MGS, slightly give the same results different from SVD, CGS, built in function and the Householder, however the S
%Plot the difference between AX - b
%a) the Equations method
plot(tj,(f - A*x),'-*')
hold on
%e) the Inbulit in Householder
plot(tj,(f - A*xh),'-o')
title('Ax - f against t')
xlabel('tj');ylabel('Ax - f')
legend('Equations method','Householder')
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate.
RCOND =  2.800825e-17.

Table =

  12×6 table
```

Normal equation	CGS	MGS	Householder	Builtin function	SVD
0.999999989587329	1.00001318251953	0.999999998520677	1.0000000009966	1.00000000099661	1.0000000009966
2.84029396133336e-06	-0.00225720796774053	3.05080727773958e-07	-4.22742880142516e-07	-4.22742915903599e-07	-4.22742687968714e-07
-8.00010214560535	-7.93913160486272	-8.00000853723409	-7.99998123568728	-7.99998123568936	-7.99998123569402
0.00144399781030622	-0.651916745057654	8.30962169927592e-05	-0.000318763226465563	-0.000318763182123933	-0.000318763136786353
10.6560552366266	14.2711955211007	10.6663571718914	10.6694307959049	10.6694307955344	10.6694307952905
0.0460832179149034	-11.5678311875708	3.88638149588323e-05	-0.0138202880901764	-0.0138202863975713	-0.0138202856094021
-5.81579888120301	17.0680866167968	-5.68634045840887	-5.64707562703058	-5.64707563175731	-5.64707563334524
0.231892082482534	-27.8571950090876	-0.00345684334002009	-0.0753160248519546	-0.0753160164200601	-0.0753160144223081
1.33247545357655	22.3656313402359	1.60875341462289	1.69360696399754	1.69360695433438	1.69360695282722
0.270659249202885	-8.65775021730327	0.0684591563300498	0.00603210846945386	0.00603211536110353	0.00603211596325072
-0.484158233866101	1.28940347151745	-0.400264201000408	-0.374241703362034	-0.374241706147935	-0.374241706226963
0.107803579981325	0.0280984905476861	0.0927344155691747	0.0880405760611674	0.0880405765490434	0.0880405765380861

```
The Normal equation and the MGS, slightly give the same results different from SVD, CGS, built in function and the Householder, however the SVD,
 built in function and the Householder matrix give almost similar results different from CSG.
```



Published with MATLAB® R2020b

No.2

```
% Compute the weighted least squares solution using the diagonal Gaussian
% weight with t = 1/23
clear all
close all

tex = 1/23;
delta = 1;

%exact approximation
fex = cos(4*tex);

%Guassian function
w = @(t,tj,delta) exp(-(abs(t - tj)/delta).^2);

%m equally spaced points over [0,1]
m = 50; n=12;

% Vandermonde matrix t
t = zeros(m,n);
for i = 1:n
    for j = 1:m
        t(j,i) = ((j-1)/(m-1))^(n-i);
    end
end

%fliping the vandermonde matrix t to form A
A = fliplr(t);

%fuction f
tj = zeros(m,1);
for j = 1:m
    tj(j) = (j-1)/(m-1);

end

f = cos(4*tj);

%Compute the weighted least square Using the Diagonal Gaussian  weight, W
W = diag(w(tex,tj,delta));

format long
%Report the polynomial coefficients of the weighted least squares solution.
fprintf('Polynomial coefficients of the weighted least squares solution \n');
[qw,rw] = qr(W*A); xw = rw\(qw'*(W*f))

%Non Weighted least squares solution xh
fprintf('Polynomial coefficeients non weighted least squares solution \n');
[q,r] = qr(A); xh = r\(q'*f)

%Report the value of the polynomial with these coefficients at t =1/23
% Vandermonde matrix t
tc = zeros(m,n);
for i = 1:n
    for j = 1:m
        tc(j,i) = (1/23)^(n-i);
    end
end

%fliping the vandermonde matrix t to form A
Ac = fliplr(tc);

%value of the polynomial at t =1/23
pw = Ac*xw; pw(11);
fprintf('Polynomial value computed using weighted coeffieients:')
disp(pw(11));

%Compare these coefficients
%for non weighted coefficients
pnonw = Ac*xh; pnonw(11);
fprintf('Polynomial value computed using non-weighted coeffieients:');
disp(pnonw(11));
fprintf('Exact Polynomial value computed directly:')
disp(fex)
%Which method provides better approximation?
fprintf('Comparing the three polynomial values, its clear that the onw computed with the weighted \n coefficients best approximates the polynomial com
```

```
Polynomial coefficients of the weighted least squares solution

xw =

   1.000000000624414
  -0.000000284232230
  -7.999986974996736
  -0.000227372752382
  10.668683630048903
  -0.010250539427060
  -5.657718013285455
```

```
  -0.054936278918560
   1.668544782170136
   0.025157808795187
  -0.382483093694967
   0.089572717334469

Polynomial coefficeients non weighted least squares solution

xh =

   1.000000000996605
  -0.000000422742916
  -7.999981235689359
  -0.000318763182124
  10.669430795534385
  -0.013820286397571
  -5.647075631757315
  -0.075316016420060
   1.693606954334381
   0.006032115361104
  -0.374241706147935
   0.088040576549043

Polynomial value computed using weighted coeffieients:   0.984915205139420

Polynomial value computed using non-weighted coeffieients:   0.984915205008979

Exact Polynomial value computed directly:   0.984915205128733

Comparing the three polynomial values, its clear that the onw computed with the weighted
 coefficients best approximates the polynomial compared to the one computed with non-weighted coefficients.
```

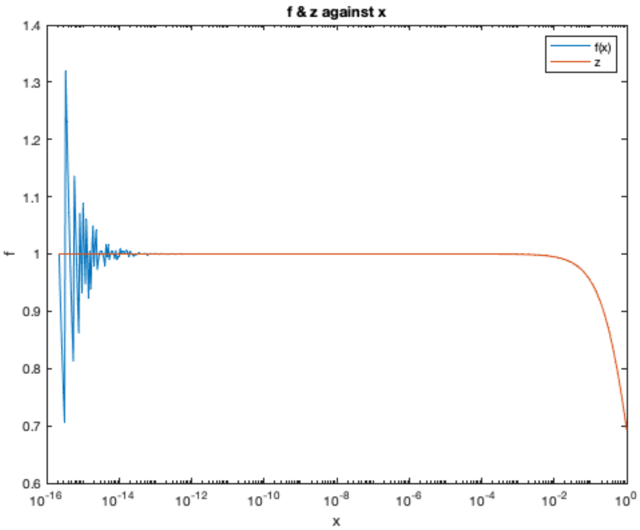
No.3

```
clear all
close all

%3a). Compute the condition number for values very close to zero.
%condition number
C = @(x) abs((x - (x+1)*log(x + 1))/((x+1)*log(x + 1)));

n = 10;

xl = linspace(-0.05,-0.0000001,n);
xr = linspace(0.0000001,0.05,n);

cl = zeros(n,1);
cr = zeros(n,1);
for i= 1:n
   cl(i) = C(xl(i));
    cr(i) = C(xr(i));
end

%What does the condition number tell you about the stability of evaluating
%f(x) near zero.
Table = table(xl',cl,xr',cr, 'VariableNames',{'x<0','C(x<0)','x>0','C(x>0)'})
fprintf('The value of x near zero for the condition number, are all small meaning the function is well condition and stable,\n since a small input to

fprintf('3b.\n');
%Evaluate th function f(x) = log(x +1)/x using the expression as given for x
f = @(x) (log(x+1))./x;

j = [0:520]';
xj = 2.^(-52 + j./10);
fj = f(xj);

%plot of f
semilogx(xj,fj);
hold on
title('f & z against x');
xlabel('x'); ylabel('f');
fprintf('The algorithm looks to be unstable near x = 0, according to the distortion of the curve observed \n near that point \n');

fprintf('3c.\n');
%Now evaluate f(x) at the same xj values as part (b)
z = 1 + xj;
y = log(z)./(z-1);
semilogx(xj,y);
legend('f(x)','z');

fprintf('Near x = 0, their is no noise the curve is stable, but in part (b), there is alot of noise in the region \n');
```

```
Table =

  10×4 table
```

x<0	C(x<0)	x>0	C(x>0)
-0.05	0.0260908287486144	1e-07	5.05838629918435e-08
-0.0444444555555556	0.0230796152641136	0.00555564444444444	0.00276502570151982
-0.0388889111111111	0.0200974832032458	0.0111111888888889	0.00550466255089419
-0.0333333666666667	0.0171439805604763	0.0166667333333333	0.00821933465005636
-0.0277778222222222	0.0142186649472498	0.0222222777777778	0.0109094085358283
-0.0222222777777778	0.0113211033332172	0.0277778222222222	0.0135752433734707
-0.0166667333333333	0.00845087179580214	0.0333333666666667	0.0162171911443507
-0.0111111888888889	0.00560755527792527	0.0388889111111111	0.0188355968277381
-0.00555564444444444	0.0027907473534124	0.0444444555555556	0.021430798577109
-1e-07	5.052635999267e-08	0.05	0.0240031278910547

```
The value of x near zero for the condition number, are all small meaning the function is well condition and stable,
 since a small input to the condition number yeild a small output, as observed from the table for different values of x
 near zero
3b.
The algorithm looks to be unstable near x = 0, according to the distortion of the curve observed
 near that point
3c.
Near x = 0, their is no noise the curve is stable, but in part (b), there is alot of noise in the region
```



Published with MATLAB® R2020b

No.4

```
%Stability of linear systems.

fprintf('4a).\n');
%(a).Does this imply that x^ is close to the exact solution x?
fprintf('No, since the residual values are not near zero, then xhat is not close to the exact solution. \n\n');

fprintf('4b).\n');
%Matrix A
A = 1./hankel(2:6,6:10);

%vector b
b = [0.882 0.744 0.618 0.521 0.447]';

%Accurate solution to the system
x = A\b

fprintf('\n');
fprintf('4c).\n');
%Obtain a condition number for A using this same software again
%condition number
C = cond(A)
fprintf('Since the Condition number of A is large then the system is ill-conditioned, therefore a small perturbation \n to the RHS can lead to large cl

fprintf('Consider a small perturbation on, db. \n');

db = [0.000002 0.000004 0.000008 0.00001 0.00007]'

R1=C*norm(db,2)/norm(b,2)

%ddx due to perturbation on the RHS
ddx = x + A\db

%Relative error
RE = norm((ddx-x),2)/norm(x,2)

fprintf('Since the relative Error,RE<=R1, and large then indeed this confirms that a very small residual\n after the system being perturbed on the RHS
```

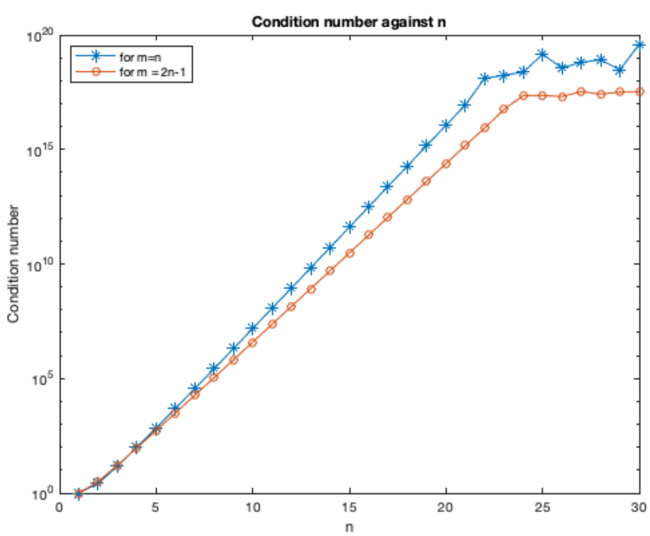
```
4a).
No, since the residual values are not near zero, then xhat is not close to the exact solution.

4b).

x =

  -2.520000000000023
   5.040000000000505
   2.519999999998072
   7.560000000002508
 -10.080000000001057


4c).

C =

     1.535043895304634e+06

Since the Condition number of A is large then the system is ill-conditioned, therefore a small perturbation
 to the RHS can lead to large change ib the system.

Consider a small perturbation on, db.

db =

   1.0e-04 *

   0.020000000000000
   0.040000000000000
   0.080000000000000
   0.100000000000000
   0.700000000000000


R1 =

  74.051562320624924


ddx =

  -2.145300000001764
   0.551040000023523
  18.217919999913303
 -13.361039999880891
  -0.667800000054639
```

```
RE =

   2.007030225261566

Since the relative Error,RE<=R1, and large then indeed this confirms that a very small residual
 after the system being perturbed on the RHS, is big enough to allow for the solution to be as far away.
```

Published with MATLAB® R2020b

```
clear all;
close all;

%Condition of the Vandermonde system.
%Experiment1
n1 = [1:30]';
C = zeros(30,1);
C(1) = 1; %Condition number of A is 1 when n=1
for n = 2:30
    m = n;
    A = vandermonde(m,n);
    C(n) = cond(A);
end

%Experiment2
C2 = zeros(30,1);
C2(1) = 1; %Condition number of A is 1 when n=1
for n = 2:30
    m = 2*n - 1;
    A = vandermonde(m,n);
    C2(n) = cond(A);
end

%plot of the two-norm condition number of A
semilogy(n1,C,'-*')
hold on

semilogy(n1,C2,'-o')
title('Condition number against n');
xlabel('n');ylabel('Condition number');
legend('for m=n', 'for m = 2n-1','Location','northwest');

fprintf('For m = n, A is a square matrix, and gives large condtion numbers as n increases, compared to when \n m=2n-1. Hence the dimension of the matr

function A = vandermonde(m,n)

    t = zeros(m,n);
    for i = 1:n
        for j = 1:m
            t(j,i) = ((j-1)/(m-1))^(n-i);
        end
    end

    %fliping the vandermonde matrix t to form A
    A = fliplr(t);

end
```

```
For m = n, A is a square matrix, and gives large condtion numbers as n increases, compared to when
 m=2n-1. Hence the dimension of the matrix affects the condition number.
```

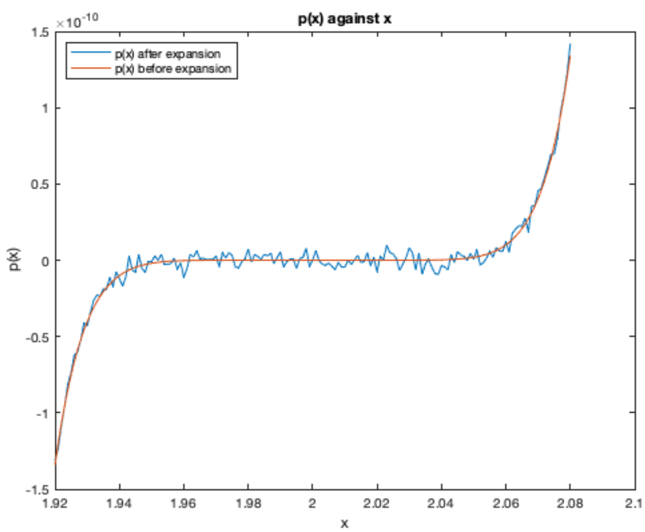


Published with MATLAB® R2020b

N0.6

```
clear all;
close all;

p = @(x) (x - 2).^9;

x = [1.920:0.001:2.080]';

%coefficeints of p
coef = [1 -2]; p1 = coef;

%expanding p to obtain the coefficiets, p1.
for i =1:8
   p1 = conv(p1,coef);
end
p1;

%Evaluating P via coefficients.
P = polyval(p1,x);

%a).Plot p(x), evaluating p via its coeffients 1, -18, 144,...
plot(x,P);
hold on
xlabel('x'); ylabel('p(x)');

%b). Produce the same plot again, now evaluating p via the expression (x-2)^9
plot(x,p(x));
xlabel('x'); ylabel('p(x)');
legend('p(x) after expansion','p(x) before expansion','Location','northwest');
title('p(x) against x');

fprintf('According to the graph, its very bad to expand a polynomial, and evaluate it at different values of x, than evaluating it before expansion\n a
```

```
According to the graph, its very bad to expand a polynomial, and evaluate it at different values of x, than evaluating it before expansion
 according to the noise displayed in the plot below for p(x) after expansion.
```



Published with MATLAB® R2020b

N0.7

```
clear all;
close all;

%Skeel condition number (CN).

% identity matrix
I = eye(10);
%Permutation Matrix P
P = [I(:,4) I(:,7) I(:,8) I(:,5) I(:,2) I(:,9) I(:,10)...
    I(:,3) I(:,6) I(:,1)];

%Verify that this is true for both the standard & Skeel
%standard CN
Cs = cond(P)
fprintf('Hence the standard condition number for P is 1\n');

%Skeel CN
Sc = norm((abs(inv(P))*abs(P)),2)
fprintf('Hence the skeel condition number for P is 1\n');

%Scaling the third column of P
P(:,3) = (10^(-10))*P(:,3);
%standard CN
Cs = cond(P)
fprintf('Hence the standard condition number for P after scaling 1s: %e \n',Cs);

%Skeel CN
Sc = norm((abs(inv(P))*abs(P)),2)
fprintf('Hence the skeel condition number for P after scaling is: 1\n');
```

```
Cs =

     1

Hence the standard condition number for P is 1

Sc =

     1

Hence the skeel condition number for P is 1

Cs =

     1.000000000000000e+10

Hence the standard condition number for P after scaling 1s: 1.000000e+10

Sc =

     1

Hence the skeel condition number for P after scaling is: 1
```