Tutorial letter 202/2/2018

APPLIED LINEAR ALGEBRA

APM1513

Semester 2

Department of Mathematical Sciences

IMPORTANT INFORMATION:

This tutorial letter contains solutions to assignment 2

BARCODE

*Assignment 2

QUESTION 2. a) Solving the system using the A<u>c</u>onstruction we have
i)**Q2ai.m**

```
A=[2 -1 3;4 2 -5;6 3 1];
b=[8;-9;12];
x=A\b
```

which give the output

```
ans =
    0.50000
    2.00000
    3.00000
```

ii)Solving the system using Gauss_Seidel method. In this instance you needed to know how to use write codes using the functions as explained in your tutorial letter. The technique to put the Gauss_Seidel code inside the iterative_linear_solver code since it does most of the import calculations with regard to **TOL**. You could directly amend the Gauss_Seidel code to accommodate the **TOL**. Notice also that I have slightly modified the iterative_linear_solver code. Input *A*, *b* and

```
xinitial= [0;0;0], TOL=10^{-7}, k=200 (say)
```

to
iterative_linear_solve.m

```
 function
xnew=iterative_linear_solve(A,b,xinitial,TOL,max_it)
xold=xinitial; k=0;
do xnew=Gauss_Seidel(A,b,xold); err=max(abs((xnew-xold)./xnew));
xold=xnew; k=k+1;
until((err<TOL) | (k>max_it));
k if (k>max_it) disp("ERROR: METHOD DID NOT CONVERGE");
xnew=[]; endif endfunction
```

and
Gauss-Seidel.m

```
function xnew=Gauss-Seidel(A,b,xold) n=size(A)(1);
At=A; xnew=xold; for k=1:n
At(k,k)=0;
end
for k=1:n xnew(k)=(b(k)-At(k,:)*xnew)/A(k,k);
end endfunction
```

we found that the Gauss_Seidel method did not converge at all and we got the answer

```
ERROR: METHOD DID NOT CONVERGE
```

To see this much further, take for instance, max_it=10000, the method still does not converge.
b) Solving the system using the A\construction we have
i) **Q2bi.m**

```
A=[10 1 2;1 10 -1;2 1 10];
b=[3;1.5;-9];
x=A\b
```

which give the output

```
ans =
    0.50000
    0.00000
   -1.00000
```

ii) We found that the Gauss_Seidel method did in fact converged when $k = 16$(number of iterations).
QUESTION 3. octave/matlab codes
Q3.m

```
 function test=matrix(A) [n,m]=size(A) if (n==m)
   disp('matrix is a square')
else
    disp('matrix is not a square')
end for k=1:n
    for l=1:m
        if (k==l) & (abs(A(k,l))>sum(abs(A(k,1:1:m)))-abs(A(k,l)))
           disp('diagonaly dominant')
        end
        if (k==l) & (abs(A(k,l))<sum(abs(A(k,1:1:m)))-abs(A(k,l)))
            disp('not diagonaly dominant')
        end
    end
end end
```

a) We use the matrix $A = [2 - 13; 42 - 5; 631]$ and $B = [034; 133]$ to test if they are a square matrix or not using the above code
i) for the matrix A the output is

```
 n =

     3


m =

     3

matrix is a square
```

ii) and for matrix B output is

```
 n =

     2


m =

     3

matrix is not a square
```

b) To test the above code for matrix diagonality we use the matrix in question 2 (b) and we indeed see that the matrix is indeed diagonally dominant with the following output from the above code:

```
 diagonaly dominant diagonaly
dominant diagonaly dominant
```

This is the results for each row computation for testing strictly diagonality condition and we have done it for A_{11}, A_{22}, and A_{33}. The condition on which strictly diagonality for matrices is discussed in your study guide page 62. The above code also works for any other matrix that is not diagonally dominant for example take the following matrix
$A = [234; 263; 986]$ for which the code gives the output

```
 not diagonaly dominant diagonaly dominant not diagonaly dominant
```

which clearly shows that the metrix A is not diagonal as we see for the first and third rows for A_{11} and A_{33} respectively. In this case the strictly diagonality condition is not satisfied.

QUESTION 4.LEFT AS AN EXERCISE!

QUESTION 5. Q5.m

```
 format long n=13; H=ones(n,n); b=ones(n); x=ones(n); for K=1:n
    for L=1:n
        H(K,L)=1/(K+L-1);

    end
end

x=(H^n\b^n)^(1/n)
```

Note: the Hilbert matrix can also be generated by the single command $H = hilb(8)$ on the Octave/Matlab command prompt. If you increase the dimension n you get the following warning: Matrix is close to singular or badly scaled. This means that the results are not that reliable for n large.

QUESTION 6. Q6.m

```
 format long

b=[40000 44000 52000 64000 80000 84000]';
 t=[1:6]';
 A=[t.^2 t ones(6,1)];
 det(A'*A)

xs=A\b

 tt=1:6;
 s=xs(1)*t.^2 + xs(2)*t + xs(3);
 plot(t,b,'r',tt,s,'b')
 err=A*xs-b
```

Below is the output graph is

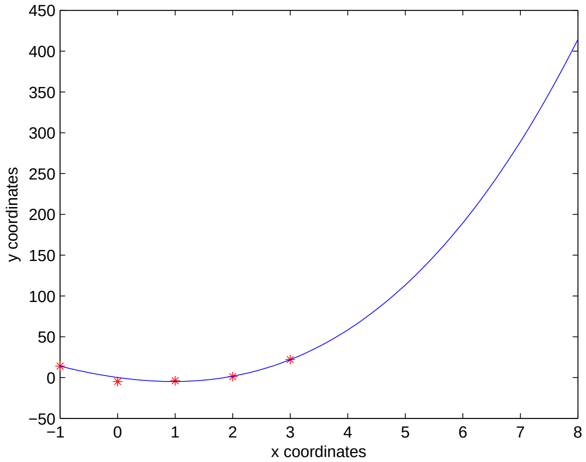


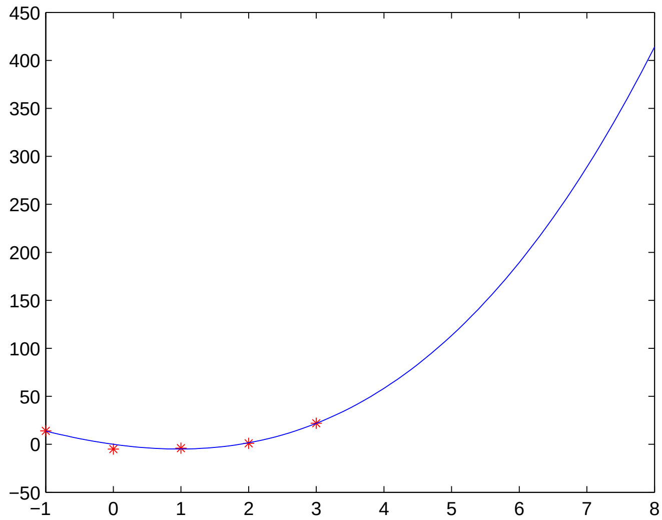
Figure 1: The sales curve.

The smooth graph is the best quadratic fit to the data while the irregular graph is the actual once. A similar problem can be found on your tutorial letter page. 74.

QUESTION 7. Q7.m

```
 format
long x=[-1 0 1 2 3]';
b=[14 -5 -4 1 22]';
A=[x x.^2 x.^3] xs=A\b;
xx=-1:0.1:8; plot(x,b,'*r',xx,xs(1)*xx+xx.^2*xs(2)+xx.^3*xs(3),'b')
```

which give the results



QUESTION 8. 7. Here is the octave/matlab script files to do the calculations for the eigenvalues and the eigenvectors
(a)**File eig1.m**

```
 A=
[2.6731385 2.6381454 3.8272855 4.7022868;
    2.7080530 -0.009461 0.1019678 2.3595453;
    5.4426950 2.4321965 4.8982919 7.1372124;
    2.7672696 3.4815125 2.3344020 8.2787928]
[P1 L1]=eig(A) diag=inv(P1)*A*P1
```

Typing the script file name $>$ *eig*1 on the command prompt we obtain

```
 A =

    2.6731385 2.6381454 3.8272855  4.7022868
    2.7080530 -0.009461 0.1019678 2.3595453;
    5.4426950 2.4321965 4.8982919 7.1372124;
    2.7672696 3.4815125 2.3344020 8.2787928



P1 =

    -0.4624204 -0.4690215 -0.4356199 -0.4616522
    -0.1748299  0.8048279 -0.7504984 -0.0093571
    -0.6844440  0.2967166  0.3110040 -0.6972133
    -0.5358529 -0.2103097  0.3876389  0.5483459
L1 =

    14.78446       0         0         0
         0    -2.16659       0         0
         0         0    0.30142        0
         0         0         0    2.92146




diag =

    4.78446       0         0         0
         0    -2.16659       0         0
         0         0    0.30142        0
         0         0         0    2.92146
```

You can save the above script using any name of your choice as an **m** file. Notice the difference between a scrip file and a function file.
ii) Power method code is given in the tutorial letter. Just enter A, TOL and max and continue from there.

iii) Matrix diagonalization is trivial in Octave, because the eigenvectors matrix P1 returned by Octave is also the diagonalization matrix as we can check by evaluating $P1^{-1}AP1$ in the script above,

(b)(a)**File eig2.m**

```
 B= [2.9541 3.6650 7.1445 1.9998;
    9.1406 15.3006 12.2773 10.3552;
    7.9200 9.6881 7.4619 6.7253;
   21.6960 35.6289 30.0172 20.7166]
[P1 L1]=eig(B) diag=inv(P1)*B*P1
```

Typing the script file name $> eig2$ on the command prompt we obtain

```
 B =

  2.9541  3.6650  7.1445  1.9998;
  9.1406 15.3006 12.2773 10.3552;
  7.9200  9.6881  7.4619  6.7253;
 21.6960 35.6289 30.0172 20.7166]


P1 =

  0.110699   0.730045  -0.708599  -0.353896
  0.397694  -0.508586   0.098037  -0.461470
  0.258050   0.291577   0.632058   0.230541
  0.873496  -0.351222  -0.297957   0.780163
L1 =

  48.55537  0.00000   0.00000   0.00000
   0.00000  2.29227   0.00000   0.00000
   0.00000  0.00000  -3.08485   0.00000
   0.00000  0.00000   0.00000  -1.32959
diag =

  48.55537  0.00000   0.00000   0.00000
   0.00000  2.29227   0.00000   0.00000
   0.00000  0.00000  -3.08485   0.00000
   0.00000  0.00000   0.00000  -1.32959
```

You can save the above script using any name of your choice as an **m** file. Notice the difference between a scrip file and a function file.
ii) Power method code is given in the tutorial letter. Just enter A, TOL and max and continue from there.
iii) Matrix diagonalization is trivial in Octave, because the eigenvectors matrix P1 returned by Octave is also the diagonalization matrix as we can check by evaluating $P1^{-1}AP1$ in the script above,

QUESTION 9.
Here is the modified Octave script file
power_method.m

```
 function [e_vec lam]=power_method(A,TOL,max_it) k=0; n=size(A)(1);
A_inv=inv(A);% modified here e_vec_old=rand(n,1);
do e_vec_new=A_inv*e_vec_old;
lam=(e_vec_new'*e_vec_old)/(e_vec_old'*e_vec_old);
err=norm(e_vec_new/norm(e_vec_new)-e_vec_old/norm(e_vec_old));
e_vec_old=e_vec_new;
k=k+1; until((err<TOL) | (k>max_it));
k e_vec=e_vec_new/norm(e_vec_new);
if (k>max_it) disp("ERROR: METHOD DID NOT CONVERGE");
e_vec=[]; lam=[]; endif endfunction
```

which give the following result

```
 k=36 answer = 0.33752
         0.45936
        -0.25171
        -0.78212
```

for the following matrix

```
 B=[-1.54575 -3.47002 -1.70112 -2.58917;
  -3.28104 -2.07998 -1.45597 -2.75629;
  0.55497 0.94078 2.02863 0.46100;
  8.94120 9.67047 4.47796 9.09710]
```

This is indeed the smallest eigenvector as it can be confirmed from the above results obtained by **eig2.m** code. For the following matrix

```
 B=[4.9541 6.6650 8.1445 2.9998; 10.1406 17.3006 14.2773 9.3552; 8.9200
10.6881 8.4619 7.7253; -25.6960 -40.6289 -38.0172 -21.71666],
```

we obtain

```
 k=19 answer = 0.564503
        -0.738274
         0.365124
         0.054515
```

which give a contradictory results, i.e. in question 3. we have **eig3.m** code giving complex results while here we have real results. Because of this fact, we cannot say for sure if this results are for the lowest eigenvector. This has something to do with the fact that the power_method did not work in question 3. That also means here that the power_method failed to working.