Tutorial Letter 202/1/2014

Applied Linear Algebra

APM1513

Semester 1

Department of Mathematical Sciences

This tutorial letter contains solutions for assignment 02.

BAR CODE

Learn without limits.

UNISA university of south africa

Assignment 2

QUESTION 2. Here are the Octave/Matlab codes
(a) Solving the systems using the $A\backslash b$ command.
Code: Q2a

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

which give

```
x = 0.5000
    2.0000
    3.0000
```

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=20 (say)
```

to
Code: iterative_linear_solver.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
```

We now use the the Gauss_Seidel code below as a sub code which is called by the main iterative_linear_solve code above.

Code: 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
```

In this case the Gauss_Seidel method did not converge at all. It gave 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 systems using the $A\backslash b$ command.

Code: Q2b

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

which give

```
x = 0.5000
    0.0000
   -1.0000
```

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=20 (say)
```

to

Code: iterative_linear_solver.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
```

We now use the the Gauss_Seidel code below as a sub code which is called by the main iterative_linear_solve code above.

Code: 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
```

From the above codes we found that the system does indeed converge. It converges to

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

after $k = 15$ 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 $A = [034; 133; 086]$ 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 (a) 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 EXERCISE!

QUESTION 5. octave/matlab code
Q5.m

```
format long
n=20
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. octave/matlab code

Q6.m

```
format long

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

xs=A\b

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

Below is the output graph is

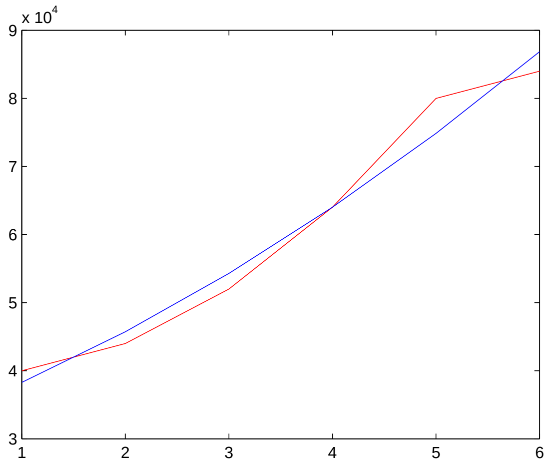

Figure 1: The sales curve.

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. octave/matlab codes

```
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')
```

Below is the output graph

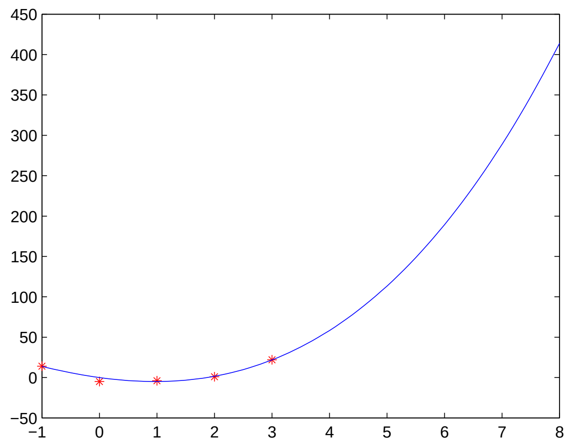

Figure 2: The sales curve.

QUESTION 8. octave/matlab codes
Let choose 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]
[P1 L1]=eig(B)
diag=inv(P1)*B*P1
```

Then we modify the Octave script file as

```
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
```

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

```
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.

QUESTION 9. octave/matlab codes
Here is the modified Octave script file

```
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.