Numerical Analysis 4670: Final Assignment

Due on Monday, December 5, 2016

Glunt 10:00am

Michael Timbes

Contents

Problem 1

In this problem I am asked to find the dominate eigenvalues using the power method. The power method works by "peeling off" coefficients of a vector that emerges from the product of the original matrix and the initial vector.

$$A * \mathbf{x}^t = \mathbf{y}^{t+1} \tag{1}$$

The original matrix is A, the initial vector is the $\mathbf{x}^t$ and the result if the $\mathbf{y}^{t+1}$ vector. The next step is then to find a new $\mathbf{x}^{t+1}$ vector by taking the infinity norm of the y vector, and then dividing the y vector by that value.

$$\mathbf{x}^{t+1} = \frac{\mathbf{y}^{t+1}}{\parallel \mathbf{y} \parallel_\infty} \tag{2}$$

Then test the next eigenvalue compared to the previous and see if there is a small difference which would mean convergence. Below are the results after running my implementation of the power method "EigPow".

```
V:
  0.462516
  1.000000
  0.458367
  0.832669
  0.716117
 -------------------------
 Largest Eigenvalue:
 23.240009
```

Listing 1: Eigen Value Approximation Routines

```
Program EigPow
      IMPLICIT NONE
      DOUBLE PRECISION, ALLOCATABLE, DIMENSION (:,:):: A
      DOUBLE PRECISION, ALLOCATABLE, DIMENSION(:)::B,B2
      DOUBLE PRECISION::large=0,eig=0
      INTEGER::I,J,N=0,IOSTATUS,filenum=11

      open(7,file='A.txt')
     do
       read(7,*,iostat=iostatus)
         if(iostatus/=0) THEN !Logic Check-EOF
           exit
         else
           n=n+1
         end if
     end do
     !print*,"Size of matrix:"
     n=n-1
     !print*,n
      allocate(A(n,n),b(n),b2(n))
      !Return to BOF
     REWIND 7

     do i=1,n
       read(7,*)(a(i,j),j=1,n)
```

```
      end do
       ! print*,A

        read(7,*)b
        !print*,B
      close(7)
!do i=1,n
!write(*,'(10f10.6)')A(i,1:n)
!enddo
!write(*,*)
!write(*,*)"------------------------------"
b2=b
call EPow(A,B,n,large)
call InvPower(A,B2,N,eig)


open( unit=filenum, file='eigvals.txt', status='replace')

write(filenum,*)"RESULT:"
write(filenum,*) "A:"
do i=1,n
write(filenum,'(10f10.6)')A(i,1:n)
enddo
write(filenum,*)"-------------------------"
write(filenum,*)
write(filenum,*) "V:"
do i=1,n
write(filenum,'(10f10.6)')B(i)
enddo
write(filenum,*)"---------------------------"
write(filenum,*)"Largest Eigenvalue: "
write(filenum,'(10f10.6)')large
write(filenum,*)
write(filenum,*)
write(filenum,*) "Smallest Eig Vector:"
do i=1,n
write(filenum,'(10f10.6)')B2(i)
enddo
write(filenum,*)"---------------------------"
write(filenum,*)"Eigenvalue: "
write(filenum,'(10f10.6)')eig









close(filenum)
deallocate(A,B)

End Program EigPow
```

```
! (TAG:A) Soubroutine to take matrix A and multiply it by any vector(default is where all compon
! (TAG:B) Find the largest value in the current iteration of the vector 'b' normally known as x
! (TAG:C) Normalize the B vector to prevent overflow. Also check for convergence.
Subroutine EPow(A,B,N,large)
IMPLICIT NONE
INTEGER:: N,I,J,it=1
DOUBLE PRECISION:: A(n,n),B(n),TEMP(N),old,large
! TAG:A
        1 do i=1,n
        temp(i)=0
                do j=1,n
                temp(i)=temp(i)+(A(i,j)*b(j))

                enddo
        enddo
        do i=1,n
                b(i)=temp(i)
        enddo

! TAG:B
        old=large
        large=abs(b(1))
        do i=2,n
        IF(ABS(b(i)) .GT. large) THEN
        large=ABS(b(i))
        ENDIF
        enddo
! TAG:C
        do i=1,n
        b(i)= b(i)/large
        enddo
        if(abs(large-old)>.000001)THEN
       it= it+1
         goto 1
        ENDIF

print*,"Iterations needed for problem 1:",it
RETURN
END

!Subroutine to solve problem two in the final HW assignment.
!(TAG:A) Create the AQ Matrix
!(TAG:B) Solve Linear System-Find largest value
!(TAG:C) Update the Xt vector-Reset AQ and note the iteration
Subroutine InvPower(A,B,N,eig)
IMPLICIT NONE
INTEGER:: N,I,J,itmax=1,P(n),ipos,it
DOUBLE PRECISION:: A(n,n),AQS(n,n),B(n),bsafe(n),xo(n),TMat(1,n),xoT(1,n),TMatA(n),AQ(n,n),E(n,n
DOUBLE PRECISION::Q,tmp,tmpb,C=0,mul=0,mulold,eig
write(*,*)"Max iterations: "
read*,itmax
!TAG:A
```

```
     do i=1,n
          xo(i)=1

     enddo
xoT(1,:)=xo(:)
bsafe=b
tmp=0
tmpb=0
     do i=1,n
          TMatA(i)=0
          do j=1,n
               TMatA(i)=TMatA(i)+(A(i,j)*xo(j))
          enddo
     tmp=tmp+(TMatA(i)*xoT(1,i))
     tmpb=tmpb+(xoT(1,i)*xo(i))
     enddo
Q=tmp/tmpb
call EYE(E,N,Q) !Creates Identity matrix and handles multiplication of I*Q
!Creates the AQ matrix as in pg.380 in the Numerical Analysis book
do i=1,n
     do j=1,n
     IF(i.eq.j) THEN
     AQ(i,j)=A(i,j)-E(i,j)
     ELSE
     AQ(i,j)=A(i,j)
     ENDIF
     enddo
enddo
AQS=AQ
do j= 0,itmax
!Call Partial Pivoting Routine
call ParPLU(n,AQ,p,xo,b)

!Find Maximum Abs Value-Store into C
call Get_Max(b,n,ipos,c)
c=b(ipos)

!TAG:C
!B is the dummy matrix for the solution of the subroutine call, xo is being updated here.
do i=1,n
xo(i) = 1/c*b(i)
enddo
mulold=mul
mul=q+(1/c)
IF(abs(mul-mulold)<.000001) THEN
goto 1
endif
it=j
AQ=AQS
enddo
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1 eig=mulold
write(*,*)"Iterations needed for problem 2: ",it
```

```
RETURN
END


! Some utility subroutines to make things easier to read:
!Subroutine that creates the 2-d identiy matrix
Subroutine EYE(E,N,Q)
implicit none
integer::N,I,J
double precision::E(N,N),Q
     do i=1,n
     do j=1,n
     IF(i.eq.j) THEN
          E(i,j)=1*Q
     ELSE
          E(i,j)=0
     END IF
     enddo
     enddo
return
end

!PARTIAL PIVOTING SUBROUTINE FROM EARLIER ASSIGNMENT
Subroutine ParPLU(n,a,p,xo,b)
implicit none
double precision :: a(n,n),mul,maxv,temp(n),xo(n),b(n)
integer :: n,i,j,k,p(n), maxr

!Initialize the permutation matrix
do i=1,n
     p(i) = i
end do
!Begin to find the max value in the matrix
do k= 1,n-1
     maxv= abs(a(k,k))
     maxr= k
          do i = k+1,n
          if (abs(a(i,k)) >maxv) THEN
               maxr=i
               maxv= abs(a(i,k))
          end if
          end do
          !Condition for row swap is when the row with the maximum
          !value is not the current row in the loop
     if(maxr.GT.k) THEN
          p(k) = maxr
          temp(k:n) = a(maxr,k:n)
          a(maxr,k:n) = a(k,k:n)
          a(k,k:n) = temp(k:n)
     end if
     !Compute the multipliers
     a(k+1:n,k) = a(k+1:n,k)/a(k,k)
          !Elimination process with multiplier from above
```

```
          do i = k+1,n
               a(i,k+1:n) = a(i,k+1:n) - a(i,k)*a(k,k+1:n)
          end do
end do

call ParPLUSol(n,a,xo,p,b) !Solves from within to maintain matrix information- also a bit easier
return
end

!Finishes the back substitution for the ParPLU
Subroutine ParPLUSol(n,a,b,p,x)
implicit none
double precision :: a(n,n),b(n),x(n),s,y(n)
integer :: n,i,j,k,p(n)

y=b !Mutator matrix

do k= 1,n-1
     if(p(k) > k) THEN
          s=y(k)
          y(k) = y(p(k))
          y(p(k)) = s
     end if
     do i = k+1,n
          y(i) = y(i) - a(i,k)*y(k) !Gauss elimination done to the B
     end do
end do
!Solve for coefficients
IF(a(n,n)/=0) THEN
x(n) = y(n) / a(n,n)
ELSE
x(n)=0
endif


     do i= n-1,1,-1
          s=y(i)
               do j = i+1,n
                    s= s-a(i,j)*x(j)
               end do
          x(i) = s/a(i,i)

     end do
!do i=1,n
!write(*,'(10f10.6)')x(i)
!enddo
!write(*,*)
return
end
Subroutine Get_Max(b,n,maxr,maxv)
implicit none
integer::k,i,n,maxr
double precision::b(n),maxv
```

```
maxv=abs(b(1))
maxr=1
do i=2,n-1
IF(maxv.lt.abs(b(i))) THEN
maxv=abs(b(i))
maxr=i
ENDIF
enddo
return
end
```

Problem 2

This problem asks to find the eigenvalue of the least magnitude and its corresponding eigenvector using the inverse power method. The idea of the inverse power method is that if there is a greatest eigenvalue for a matrix then there should be a smallest that is the maximum of the original matrix's inverse.
The steps to the inverse power method are similar to the power method. The idea is actually to continue to solve the equation below for the y vector then use the maximum value of that y vector to make a new x vector that ideally better describes the eigenvalue we are looking for.

$$\mathbf{y}^m = (A - q\mathbf{I})^{-1}\,\mathbf{x}^{m-1} \tag{3}$$

To handle the inverse simply move it to the other side so that the equation is now

$$(A - q\mathbf{I})\,\mathbf{y}^m = \mathbf{x}^{m-1}. \tag{4}$$

The only other additional step is finding the q value in the previous equation which is really the ideal number that we want to get close to. The q value is the initial approximation of the eigenvalue and can be approximated or by the following

$$q = \frac{\mathbf{x}^{0t}(A\mathbf{x}^0)}{\mathbf{x}^{0t}\mathbf{x}^0}. \tag{5}$$

Below are the results when I ran my implementation of the inverse power method.

```
Smallest Eig Vector:
  0.226723
  0.490194
  0.224689
  0.408169
  0.351036
 -------------------------
 Eigenvalue:
 23.240010
```

Problem 3

In this question I am asked to find the smallest magnitude eigenvalue from a tridiagonal matrix of size (n=1000). My approach was to use the inverse power method like the last problem but with a twist. Instead of using a routine like partial pivoting I would use Jacobi's method to solve the matrices to find the new approximations for the eigenvectors.
I had one issue though, it seemed that the approximations were larger than the answer was. I believe it might have been because of the numbers being close together the inverse power method might have not been the best option. I would definitely consider another method that approximates all eigenvalues and then taken the minimum from that set. However, below are my results from the program.

```
RESULT:
 X:
  0.000010
  0.000084
  0.000367
  0.001150
  0.002896
```

Problem 3 continued on next page. . .

```
Eig Val:
 0.998700
```

Listing 2: Different program for the tridiagonal matrix

```
Program EigPow_2
implicit none
integer:: n,filenum=7,itmax,i
double precision, allocatable, dimension (:) :: b,x,l,u,d
double precision, dimension(:):: px(10)
double precision:: mul
!print*,"n= "
!read*,n
n=1000
allocate(l(n-1),u(n-1),d(n), b(n),x(n))



call Problem3(n,L,U,D,B)
!write(*,*) "L:"
!do i=1,n-1
!write(*,'(10f10.6)')L(i)
!enddo
!write(*,*)
!write(*,*) "U:"
!do i=1,n-1
!write(*,'(10f10.6)')U(i)
!enddo
!write(*,*)
!write(*,*) "D:"
!do i=1,n
!write(*,'(10f10.6)')D(i)
!enddo
!write(*,*)

call InvPower2(n,L,U,D,B,X,px,mul)

open( unit=filenum, file='EPow2Res.txt', status='replace')

write(filenum,*)"RESULT:"
write(filenum,*) "X:"
do i=1,5
write(filenum,'(10f10.6)')px(i)
enddo
write(filenum,*)
write(filenum,*)"Eig Val:"
write(filenum,'(10f10.6)') mul
close(filenum)

deallocate(l,u,d,b,x)
End program EigPow_2
```

```
!Broken apart LUD matricies that make up A
Subroutine Problem3(n,L,U,D,B)
IMPLICIT NONE
integer:: n,i,j,k,itmax
DOUBLE PRECISION :: l(n),u(n),d(n),b(n)
do i=1,n
d(i)=2+(dble(i**2)/dble(n**4))
b(i)=1
enddo
do i=1,n-1,1
l(i)=-1
u(i)=-1
enddo
return
end
!!!
!!!
!!!
Subroutine InvPower2(n,L,U,D,B,X,px,mul)
implicit none
integer :: n,itmax=100,i,j,k,maxr
double precision :: l(n-1),u(n-1),d(n),b(n),x(n),tmp(n),summ,tol=.0000000001,px(10)
double precision :: maxv,tmpa=0.0,tmpb=0.0,q2=0.0,c,mul,mulold
x(:)=1
b(:)=1
!Create Q
tmp(1)=d(1)+u(1)
do i=2,n
IF(i.lt.n) THEN
tmp(i)=l(i-1)+d(i)+u(i)
ELSE
tmp(i)=d(i)+l(i-1)
endif
enddo
do i=1,n
tmpa=tmpa+(tmp(i)*x(i))
tmpb=tmpb+(x(i)*x(i))
enddo
q2=tmpa/tmpb

!write(*,'(10f10.6)')q2
do i=1,n
d(i)=d(i)-q2
!write(*,'(10f10.6)')d(i)
enddo

do j=1,20
call Jacv2(n,l,u,d,b,x,px)
call Get_Max(x,n,maxr,maxv)
c=x(maxr)
!print*,c
do i=1,n
b(i) = 1/c*x(i)
```

```fortran
enddo
mulold=mul
mul=q2+(1/c)
!DIAG
!write(*,*)"RESULT:"
!write(*,*) "X:"
!do i=1,n
!write(*,'(10f10.6)')x(i)
!enddo
!write(*,*)
!write(*,*)
!write(*,*) "C:"
!write(*,'(10f10.6)')c
!write(*,*)
!write(*,*)
IF(abs(mul-mulold)<.000001) THEN
write(*,'(10f10.6)')mul
if(n.gt.10) THEN
px=x(1:10)
endif
goto 2
endif
enddo


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
2 return
end
!!!
!!!
!!!
!!!
!Modified Jacobi Method for tri diagonal matricies

Subroutine Jacv2(n,L,U,D,B,X,px)
implicit none
integer :: n,itmax=10,i,j,k
double precision :: l(n-1),u(n-1),d(n),b(n),x(n),xo(n),summ,tol=.00001,px(5)
double precision :: v1=0,v2=0,a
!Initial Guess that all are zero
do i=1,n
x(i)=0

enddo

do k=1,itmax

x(1) = (b(1) - (u(1)*x(2)))/d(1)
!print*,"First X:",x(1)
do i=2,n
summ=0

summ = (l(i-1)*x(i-1))
```

```
 if(i.lt.n) THEN
summ = summ+(u(i)*x(i+1))
endif

!print*,"sum ",summ,i
!enddo

x(i)=((b(i)-summ)/d(i))
!print*,"X:",x(i)
xo(i)=x(i)

enddo !end-i

do i=1,n
v1= v1+abs(xo(i)**2)
v2= v2+abs(x(i)**2)
enddo
v1 =sqrt(v1)
v2=sqrt(v2)

 if(abs(v2-v1) < tol) THEN
print*,abs(v2-v1)
goto 1
ELSE
v1=0
v2=0
END IF

enddo !end-k

! print*,"Convergence Found at:",k
1 px=xo(1:5)
return
end


Subroutine Get_Max(b,n,maxr,maxv)
implicit none
integer::k,i,n,maxr
double precision::b(n),maxv

maxv=abs(b(1))
maxr=1
do i=2,n-1
IF(maxv.lt.abs(b(i))) THEN
maxv=abs(b(i))
maxr=i
ENDIF
enddo
return
end
```

Problem 4

Below are the Octave scripts I wrote to help facilitate writing the Fortran programs and to check answers.
Main Program

```
%Final Assignment for Numerical Analysis
A=[1 2 3 4 5;2 6 7 8 9;3 7 0 1 2;4 8 1 10 1;5 9 2 1 5];
%A=[-4 14 0;-5 13 0;-1 0 2];
dumn=5;
for i=1:dumn
A2(i,i)=(2+(pow2(i)/power(dumn,4)));
if(i<=dumn-1)
A2(i,i+1)=-1;
A2(i+1,i)=-1;
endif
endfor
%A=A2;
N=size(A);
N=N(1,:);
x0=ones(N,1);

system("cls");
%%%%%%%%%%%%%%%%%%%%%%%%% First Problem %%%%%%%%%%%%%%%%%%%%%%%%%
%Take the original A matrix and the initial vector guess, then
%multiply the two. Then redefine x (the initial vector) as x=result/max(result)
%to normalize the result. Octave does this process in the background, and
%uses the maximum eigien approximation by default.
disp("First Probelm: ")
E=eig(A);
[E_Vector,E_Diag]=eig(A);
E_Max=max(E);
E_Max

%%%%%%%%%%%%%%%%%%%%%%%%% SECOND PROBLEM %%%%%%%%%%%%%%%%%%%%%%%%%
%Once the power method has been applied, the inverse power method is used to
%find the least eigen value. Since yˆt = Axˆt then there is a similar vector in
%terms of ’x’ that is the smallest egien value which is equal to (A-(largest egien value)*
%%From this one can solve for the x value.
%[x0,oldmul]=Einvpow(A,x0);
%iterations = input("Max Iterations: ")
iterations=15;
mul2=0;
%q value
q=(x0.’*(A*x0))/(x0.’*x0);
%Identity Matrix
I=eye(N);
%Form AQ
AQ=A-(q*I);

for i= 1:iterations
```

```
mulold2=mul2;
[x0,mul2]=Einvpow(x0,q,AQ);
%disp("Current Xt:"),disp(x0);
%disp("Current Mul:"),disp(mul)
diffy= abs(mul2-mulold2);
if(diffy<.00001)
break;
endif
endfor
disp("Second Probelm: ")
disp("Eigenvalue Approximation: "),disp(mul2);
disp("Iterations Needed: "),disp(i)


%%%%%%%%%%%%%%%%%%%%%%%%% THIRD PROBLEM %%%%%%%%%%%%%%%%%%%%%%%%%

%Here I am making the matrix as defined in the problem set.
%It is tridiagonal, if the matrix is not already tridiagonal then
%consider performing a Householder transform.
n2=5;
b2=ones(n2,1);
x2=ones(n2,1);
for i=1:n2
a2d(i)=(2+(pow2(i)/power(n2,4)));
endfor
for i=1:n2-1
a2l(i)=-1;
a2u(i)=-1;
endfor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tempv=x2;
tempv(1)=a2d(1)+a2u(1);
for i=2:n2
if(i< n2)
tempv(i)=(a2l(i-1)+a2d(i)+a2u(i));
else
tempv(i)=a2d(i)+a2l(i-1);
endif
endfor
q2=(x2.'*tempv)/(x2.'*x2);
%a2d=a2d-q2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mul=0;
iterations2=25;
for i= 1:iterations2
mulold=mul;
[x2,mul]=Einvpow2(a2l,a2u,a2d,x2,q2);
%disp("Current Xt:"),disp(x0);
%disp("Current Mul:"),disp(mul)
```

Problem 4 continued on next page. . .

```
diffy= abs(mul-mulold);
if(diffy<.00001)
break;
endif
endfor
disp("Third Probelm: ")
disp("Eigenvalue Approximation: "),disp(mul);
disp("Iterations Needed: "),disp(i)
CD= eig(A2);
%x2(1:10)
%jac2(a2l,a2u,a2d,b2)
%parpiv(A,b2)
```

Routine to Drive Inverse Power Method

```
%Author: Michael Timbes
%Function takes 4 arguments and returns two values
%AQ = (A-qI)
%q=x0.'*(A*x0)/(x0.'*x0)
%xt is the most updated x^t column matrix
%!!!!!!DEPENDENCIES: parpiv.m!!!!!!!!!!
%parpiv.m solves linear equations with Gauss elimination and
%partial (row) pivoting.
function [xt,res] = Einvpow(xt,q,AQ)
res=0;
y=parpiv(AQ,xt)
[val,ipos]=max(abs(y));
%disp("VAL"),disp(val)
c=y(ipos)
xt=1/c*y;
res=q+(1/c);
%disp("Second Probelm: "),disp(mul)
endfunction
```

Routine to Solve Linear Systems Using Partial Pivoting

```
%General Script for gaussian elimination with partial pivoting(row pivoting)
%a=[71 5 17 10;5 41 12 7;17 12 46 24;10 7 24 75 ]; %Test matrix
%b=[1; 0; 2; 1]; %Test matrix
%input("Matrix A: ",a)
%input("Matrix B: ",b)
function [x] = parpiv (a,b)
n=size(a);
n=n(1);
x=zeros(n,1);
system("cls");
%Implement Gauss Elimination- Upper Triangularization
p=ones(n);
for k=1:n-1
maxv= abs(a(k,k));
maxr= k;
```

```
for i = k+1:n
if (abs(a(i,k)) >maxv)
maxr=i;
maxv= abs(a(i,k));
endif
endfor
if(maxv==0)
disp("Error, maxvalue is: "),disp(maxv)
endif
if(maxr > k)
p(k) = maxr;
temp = a(maxr,(k:n));
a(maxr,(k:n)) = a(k,(k:n));
a(k,(k:n)) = temp;
endif
%Multipliers Being stored in the array
a(k+1:n,k) = a(k+1:n,k)/a(k,k);
%Subtract off multipliers
for i = k+1:n
a(i,k+1:n) = a(i,k+1:n) - a(i,k)*a(k,k+1:n);
endfor
endfor
%disp("A matrix POST"),disp(a)
%Solving Ax=b for x. Back sub.
y=b; %Set y to the function values
for k= 1:n-1
if(p(k) > k)
s=y(k);
y(k) = y(p(k));
y(p(k)) = s;
endif
for i = k+1:n
y(i) = y(i) - a(i,k)*y(k); %Back subsitution preparation, matching the row ops
endfor
endfor
%Solve for coefficients
if(a(n,n)!=0)
x(n)=y(n)/(a(n,n));
%disp("Initial X"),disp(x(n))
else
x(n)=0;
endif
i=n-1;
do
s=y(i);
j=i;
do
j++;
s=s-a(i,j)*x(j);
```

Problem 4 continued on next page. . .

```
until(j==n)
x(i)=s/(a(i,i));
i--;
until(i==0)
%disp("Resulting Matrix: "),disp(a)
%disp("Solution Matrix: "),disp(x)
endfunction
```

Routine to Drive Inverse Power Method for Tridiagonal Matrices

```
%Author: Michael Timbes
%Function takes 4 arguments and returns two values
%AQ = (A-qI)
%q=x0.'*(A*x0)/(x0.'*x0)
%xt is the most updated x^t column matrix
%!!!!!!DEPENDENCIES: jac2.m!!!!!!!!!!!
%jac2.m solves systems of tridiagonal symmetric matrices
function [xt,res] = Einvpow2(a2l,a2u,a2d,xt,q)
res=0;
[y]=jac2(a2l,a2u,a2d,xt);
[val,ipos]=max(abs(y));
%disp("VAL"),disp(val)
c=y(ipos);
xt=1/c*y;
res=q+(1/c);
%disp("Second Probelm: "),disp(mul)
endfunction
```

Routine to Solve Tridiagonal Systems Using Jacobi's Method

```
function[x]=jac2(l,u,d,b)
%d=[  2.0400,  2.1600,  2.3600,  2.6400,  3.0000 ];
%l=[ -1, -1, -1, -1 ];
%u=[ -1, -1, -1, -1 ];
%b=[ 1; 1; 1; 1; 1];
v1=0;
v2=0;
tol=.000001;
%SOLUTION:[  1.47395;2.00685;1.86085;1.38476;0.79492 ]
n=size(d);
n=n(2);
itmax=50;

x=zeros(1,n);
xo=zeros(1,n);
for k=1:itmax
x(1) = (b(1)-(u(1)*x(2)))/d(1);
%disp("First X:"),disp(x(1))
for i= 2:n
summ=0;
summ = (l(i-1)*x(i-1));
```

```
if(i < n)
summ = summ+(u(i)*x(i+1));
endif
%disp("sum "),disp(summ),disp(i)
x(i)=((b(i)-summ)/d(i));
%disp("X:"),disp(x(i))
xo(i)=x(i);
endfor %end-i

for i=1:n
v1= v1+abs(pow2(xo(i)));
v2= v2+abs(pow2(x(i)));
endfor
v1 =sqrt(v1);
v2=sqrt(v2);
difff=(abs(v2-v1));
if( difff< tol)
%disp(abs(v2-v1));
else
v1=0;
v2=0;
endif
endfor %end-k
%disp("Convergence Found at:"),disp(k)
%disp("Solution: "),disp(x)



endfunction
```