Math 4670: Assignment Linear Systems 2

Due on Wed., November 16, 2016

Glunt 9:05am

Michael Timbes

Contents

Problem 1

A

To iterate Jacobi's method it is important to understand where it comes from. A matrix can be broken down into three main components, the upper triangle, lower triangle, and diagonal matrix. If the problem is solve a linear system of equations, the form is usually

$$Ax = b. \tag{1}$$

Where A is a matrix and b are the solutions to the equations that are used to help find x values that satisfy the equations. When breaking apart the matrices the results are three separate matrices that when added together still result in the original A matrix.

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix}, U = \begin{matrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{matrix}, L = \begin{matrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{matrix}, D = \begin{matrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{matrix}$$

From these components, the following form of the linear system of equations is derived

$$(D + L + U)(x) = b. \tag{2}$$

Solving for x and considering that normally the matrices that work well with Jacobi's method are diagonally dominate, choose the stepping to be along the diagonal values.

$$x^{k+1} = D^{-1}((-L - U) * x^k + b) \tag{3}$$

(source: http://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-jacobis-method)
The idea then is to begin stepping in x along the diagonal and with every step using the previous x values (starting with an initial guess) to eventually arrive at a value that converges. The following are the values produced by the source code. I double checked the values with the past assignment's methods and they matched almost perfectly.

Results:

```
RESULT:
 A:
 71.000000  5.000000 17.000000 10.000000
  5.000000 41.000000 12.000000  7.000000
 17.000000 12.000000 46.000000 24.000000
 10.000000  7.000000 24.000000 75.000000



 B:
  1.000000
  0.000000
  2.000000
  1.000000



 X:
  0.004162
 -0.013828
  0.045865
 -0.000608
```

Problem 1 continued on next page...

B

The Gauss-Seidel method is actually very similar to Jacobi's method except that it directly uses the x values after they are found instead of iterating through the whole method then replacing the x values. It turns out that the iterations needed for this method were far fewer (two as opposed to six in Jacobi's method). The reason there weren't as many iterations needed is really in the nature the method was designed, to speed up the process of calculating the next x value. Below is the results I received from Jacobi and then also Gauss-Seidel.

Results:

```
RESULT:
 A:
 71.000000  5.000000 17.000000 10.000000
  5.000000 41.000000 12.000000  7.000000
 17.000000 12.000000 46.000000 24.000000
 10.000000  7.000000 24.000000 75.000000


 B:
  1.000000
  0.000000
  2.000000
  1.000000


 X:
  0.004162
 -0.013828
  0.045865
 -0.000608

 X2:
  0.004162
 -0.013828
  0.045865
 -0.000608
```

Problem 1 FORTRAN Source:

Listing 1: FORTRAN Script for Jacobi and Gauss Seidel

```fortran
Program A06
      IMPLICIT NONE
DOUBLE PRECISION, ALLOCATABLE, DIMENSION(:) :: B,B2,D,X,X2,X3
DOUBLE PRECISION, ALLOCATABLE, DIMENSION(:,:) :: A,Z
INTEGER :: N,n2,I,J,K,FILENUM,IOSTATUS,itmax
```

```
open(7,file='matrixd3.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),z(n,n),b(n),b2(n),x(n),x2(n),d(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)
print*,"Please Enter number of max iterations: "
read*,itmax

call Jac(n,itmax,A,b,x)

call Gseidel(n,itmax,A,b,x2)

open( unit=filenum, file='matrix3results.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,*) "B:"
do i=1,n
write(filenum,'(10f10.6)')B(i)
enddo
write(filenum,*)
write(filenum,*)
write(filenum,*) "X:"
do i=1,n
```

```fortran
write(filenum,'(10f10.6)')x(i)
enddo
write(filenum,*)
write(filenum,*)"X2:"
do i=1,n
write(filenum,'(10f10.6)')x2(i)
enddo

close(filenum)

deallocate(a,b,x,x2,d)
End Program A06



Subroutine Jac(n,itmax,A,b,x)
implicit none
integer::n,i,j,k,itmax
double precision:: a(n,n),x(n),currentx(n),b(n),summ=0,break=0,tol=0.0000001,v1,v2
print*,"Jacobi:"

!Initial Guess that all are zero
do i=1,n
x(i)=0
enddo

!DO K TIMES (or until convergence)
do k=1,itmax
!Reset Break Condition
break=0
!Copy Current X Approximations
do j=1,n
     currentx(j) = x(j)
enddo
!As follows in Jacobi
do i=1,n
summ=0
     do j=1,n
     if(j/=i) THEN
     summ = summ+(a(i,j)*x(j))
     ENDIF
     enddo

     x(i) = ((b(i)-summ)/a(i,i))
print*,x(i)

enddo

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

```
     v2=sqrt(v2)
     if((v2-v1) < tol) THEN
          goto 1
     ELSE
          v1=0
          v2=0
     END IF
enddo

1 print*,"Convergence Found at:",k
return
end

Subroutine Gseidel(n,itmax,A,b,x2)
IMPLICIT NONE
integer::n,i,j,k,itmax
double precision:: a(n,n),x2(n),xo(n),b(n)
double precision:: summ=0,summb=0,break=0,tol=0.0000001,v1,v2
print*,"Gauss-Seidel:"

!Initial Guess
do i=1,n
x2(i)=1
enddo
do k=1,itmax

     do i=1,n
     summ=0
          do j=1,n
          if(j/=i) THEN
          summ= summ+(a(i,j)*xo(j))
          endif
          x2(i)=((b(i)-summ)/a(i,i))
          print*,x2(i)
          enddo!end-j
     xo(i)=x2(i)
     enddo !end-i

 do i=1,n
     v1= v1+abs(xo(i)**2)
     v2= v2+abs(x2(i)**2)
 enddo
     v1 =sqrt(v1)
     v2=sqrt(v2)
     if((v2-v1) < tol) THEN
          goto 1
     ELSE
          v1=0
          v2=0
     END IF

enddo !end-k
```

```
1 print*,"Convergence Found at:",k

return
end
```

Problem 2

In this problem, a tri-diagonal matrix like the matrix below is given. Using one of the previous methods a solution can be easily found.

$$A = \begin{bmatrix} 5 & 2 & \dots & 0 & 0_n \\ 2 & \ddots & \ddots & 0 & 0 \\ 0 & \ddots & 5 & 2 & 0 \\ 0 & 0 & 2 & 5 & 2_{n-1} \\ 0 & 0 & 0 & 2_{n-1} & 5_n \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1_n \end{bmatrix}$$

I decided to run the two (where n=5) dimensional matrix through the previous assignment which included partial pivoting, complete pivoting, and Gaussian elimination. The results I received are below.

Results:

```
RESULT:
 A:
  5.000000  2.000000  0.000000  0.000000  0.000000
  2.000000  5.000000  2.000000  0.000000  0.000000
  0.000000  2.000000  5.000000  2.000000  0.000000
  0.000000  0.000000  2.000000  5.000000  2.000000
  0.000000  0.000000  0.000000  2.000000  5.000000


 B:
  1.000000
  1.000000
  1.000000
  1.000000
  1.000000


 A after TRIU call:
  5.000000  2.000000  0.000000  0.000000  0.000000
  0.000000  4.200000  2.000000  0.000000  0.000000
  0.000000  0.000000  4.047619  2.000000  0.000000
  0.000000  0.000000  0.000000  4.011765  2.000000
  0.000000  0.000000  0.000000  0.000000  4.002933


 X Values:
  0.169231  0.076923  0.138462  0.076923  0.169231
```

```
 Matrix for the PLU:
  5.000000  2.000000  0.000000  0.000000  0.000000
  0.400000  4.200000  2.000000  0.000000  0.000000
  0.000000  0.476190  4.047619  2.000000  0.000000
  0.000000  0.000000  0.494118  4.011765  2.000000
  0.000000  0.000000  0.000000  0.498534  4.002933


 X Values from PLU:
  0.169231  0.076923  0.138462  0.076923  0.169231

 Matrix for the FPLU:
  5.000000  2.000000  0.000000  0.000000  0.000000
  0.400000  4.200000  2.000000  0.000000  0.000000
  0.000000  0.476190  4.047619  2.000000  0.000000
  0.000000  0.000000  0.494118  4.011765  2.000000
  0.000000  0.000000  0.000000  0.498534  4.002933


 X Values from FPLU:
  0.169231  0.076923  0.138462  0.076923  0.169231
```

Included in the results is what the A matrix looks like in two dimensions. The next problem instructs not to use two dimensional matrices but to solve the same system of equations.

Problem 3

This problem asks to solve the matrix from earlier using one of the iterative methods Jacobi or Gauss-Siedel but the problem further constrains the method to only using one dimensional vectors as opposed to a two dimensional matrix.
I felt that the most simple way to approach this problem was by Jacobi's method. Considering the matrix is diagonally dominate it seemed that stability would be no issue in this case.

Results:

```
RESULT:
 X:
  0.169231
 AT:           1
  0.076923
 AT:           2
  0.138462
 AT:           3
  0.076923
 AT:           4
  0.169231
 AT:           5
```

Problem 4

In the last problem, its asks for a solution to a similar looking equation that is 1,000,000 by 1,000,000 matrix which is actually too large to solve using a two dimensional matrix on any standard computer since double precision is needed that is 8 bytes per element that has to be allocated which would translate to roughly 8 TB of storage needed on RAM for just the matrix, not including the B and X values. However, many of those locations in a two dimensional matrix would be zero so by using only one dimensional matrices we can significantly reduce the amount of space needed.
The level of accuracy that the problem asked for was $1.0 * 10^{-9}$ which proved to be somewhat difficult to get, the highest precision I could get was $1.0 * 10^{-2}$ it is possible that the number of terms made it difficult to achieve the desired accuracy in a reasonable time. The results produced were interesting and not what I expected. The method didn't seem to converge to anything meaningful. The following values in the result are the specific component values at the requested positions.

Results:

```
RESULT:
 X:
  0.000000
  0.111111
  0.111111
  0.111111
  0.111111
```

Problems 3-4 FORTRAN Source:

Listing 2: FORTRAN Script to produce the matrices needed and attempt to solve

```fortran
Program P3_4
implicit none
integer:: n,filenum=7,itmax,i
double precision, allocatable, dimension (:) :: b,x,l,u,d
double precision, dimension(:):: px(5)
print*,"n= "
read*,n
allocate(l(n-1),u(n-1),d(n), b(n),x(n))



 call Problem3_4v2(n,L,U,D,B)
 call Jacv2(n,L,U,D,B,X,px)

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

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

```
deallocate(l,u,d,b,x)
End program P3_4

!Broken apart LUD matricies that make up A
Subroutine Problem3_4v2(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)=5
b(i)=1
enddo
do i=1,n-1,1
l(i)=2
u(i)=2
enddo
return
end

!Modified Jacobi Method for tri diagonal matricies

Subroutine Jacv2(n,L,U,D,B,X,px)
implicit none
integer :: n,itmax=100,i,j,k
double precision :: l(n-1),u(n-1),d(n),b(n),x(n),xo(n),summ,tol=.0000000001,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)
!Grab Certain Values
if(i.eq.1) THEN
```

```
px(1) = x(i)
endif
if(i.eq.250000) THEN
px(2) = x(i)
endif
if(i.eq.500000) THEN
px(3) = x(i)
endif
if(i.eq.750000) THEN
px(4) = x(i)
endif
if(i.eq.100000) THEN
px(5) = x(i)
endif


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

1 print*,"Convergence Found at:",k

return
end
```