Numerical Analysis 4670: Assignment #4

Due on Monday, October 31, 2016

Glunt 10:00am

Michael Timbes

Contents

Introduction

This homework assignment's goal is to explore the various methods of solving linear systems of equations. Problem one asks to solve by the prototype method, partial pivoting with Gauss Elimination, and then with complete pivoting with Gauss Elimination. The next problem asks to create a subroutine to calculate the Choleski Factorization of a symmetric matrix to determine if it is a positive definite matrix or not. The next problem asks if the given matrix from the first problem fits the criteria of positive definite and if so what the 'G' matrix would be. Finally, the last problem asks to find a way to determine if a matrix is positive definite if it is not symmetric- thus the Choleski Factorization may not be the best method to determine the definiteness.

Problem 1

Listing 1 shows the following program that will be referenced in problem 1.

A

The prototype method is based on simple Gaussian Elimination techniques. The idea is to create an upper triangular matrix from the original matrix by eliminating the entries below the diagonal. At line 115 the subroutine TRIU solves the matrix A and its corresponding B by Gauss Elimination.
Algorithm Outline TRIU:

- Go from column 1:n.
- Go from row k+1:n.
- Store multiplier as a(i,k)/a(k,k).
- Go from j=1,n.
- Store the result of current value minus multiplier times current a(k,j) value (effectively creating a zero entry).

After the upper triangle is found from TRIU, the values are back substituted by first starting with the last coefficient in A(n,n) and solving the value of X(n) then using that value to solve the previous coefficients in A. Line 139 shows this process as discussed in class.
Algorithm Outline BackSolve:

- Initial X(n) value equal to b(n) divided by a(n,n).
- Gather next B value.
- Subtract each component's contribution to the linear equation.
- Final X(i) value is the amount left from the subtraction divided by the diagonal value in a(i,i).

B

The partial pivoting with Gauss Elimination involves finding the maximum number in the row at a diagonal matrix value, then switching the rows to have the maximum numbers in the diagonal. When the row swap is finished and a permutation matrix is made to reflect the row change, Gauss Elimination is done to create an upper triangular matrix and in the lower triangle, the various multipliers and permutation matrix are stored to later be used in back substitution by updating the B matrix to reflect the changes applied.
The reason for doing pivoting is for greater accuracy by limiting the need to have large multipliers because

Problem 1 continued on next page...

small number are being multiplied to get rid of larger numbers. Line 166 is the start of the subroutine that uses partial pivoting.
Algorithm Outline ParPLU:

- Initialize Permutation matrix.
- Assume largest number is at diagonal position a(k,k).
- Check each value in rows below current 'k' row.
- If the value at a(i,k) is larger, store value and row location. Otherwise continue.
- If the maximum value is at a future row,update permutation matrix with row information, swap rows.
- After potential row swap, continue with Gauss Elimination.

After the process of partial pivoting, a separate backwards substitution is needed with the addition of the permutation matrix. Line 204 begins the subroutine that solves the partial pivoting X matrix using B and P.
Algorithm Outline ParPLUSol:

- Set a dummy variable Y equal to the B matrix.
- Apply row swap information to the Y matrix to reflect row swaps in A matrix.
- Apply Gauss elimination to the Y matrix using the multipliers in a(i,k).
- Begin with initial X(n) value like before, then subtract the matrix coefficients and solutions from earlier X values from the Y(i) value to find the X(i) (next) value.

C

Gaussian Elimination with complete pivoting where the column and rows could switch is very similar to the partial pivoting with some minor additions for the case of a column switch as well. Although not usually used because it is $O(n^2)$ complexity, it creates an even more accurate answer to the linear system of equations. Line 239 is the beginning of the FullPLU subroutine.
Algorithm Outline FullPLU:

- Initialize Permutation matrix.
- Assume largest number is at diagonal position a(k,k).
- Check each value in rows below current 'k' row.
- If the value at a(i,k) is larger, store value and row location. Otherwise continue.
- If the maximum value is at a future row,update permutation matrix with row information, swap rows.
- Reinitialize the max value variable and check the column values for the largest value after the potential row swap, store the largest column.
- If the maximum value is at a future column, swap column.(no permutation matrix is needed unless to check that LU equals A).
- After potential row and column swap, continue with Gauss Elimination.

After the complete pivoting is used, the same function for partial pivoting solutions is used since the same basic idea applies in both pivot strategies.

Problem 1 continued on next page...

Comparison

```
RESULT:
 A after TRIU call:
  7.000000  5.000000 17.000000 10.000000
  0.000000  0.428571 -0.142857 -0.142857
  0.000000  0.000000  4.666667 -0.333333
  0.000000  0.000000  0.000000  0.642857


 X Values:
  3.777778 -2.111111 -0.222222 -1.111111

 Matrix for the PLU:
 17.000000 12.000000 46.000000 24.000000
  0.294118  0.470588 -1.529412 -0.058824
  0.411765  0.125000 -3.250000  0.875000
  0.588235 -0.125000  0.538462 -0.346154


 X Values from PLU:
  3.777778 -2.111111 -0.222222 -1.111111

 Matrix for the FPLU:
 46.000000 24.000000 17.000000 12.000000
  0.260870  0.739130  0.565217  0.869565
  0.369565  1.529412 -0.764706 -2.176471
  0.521739  3.352941  0.192308 -0.346154


 X Values from FPLU:
 -0.222222 -1.111111  3.777778 -2.111111
```

The answers were the same out to six decimal places, past that the complete pivoting was seemingly more accurate but usually only four to six decimal places is sufficient for solutions. The complete pivoting is also not in the same order since there was not a permutation matrix applied to account for the column switching since the B matrix is only one column.

Listing 1: FORTRAN(F95) Script for Problem 1(a-c)

```
!Michael Timbes
!Numerical Analysis
!Assignment-4
PROGRAM A4
IMPLICIT NONE
DOUBLE PRECISION, ALLOCATABLE, DIMENSION(:) :: B,BCOP,X,XPLU,XFLU
DOUBLE PRECISION, ALLOCATABLE, DIMENSION(:,:) :: A,ACOP,ATRI,APAR,AFUL
INTEGER, ALLOCATABLE, DIMENSION (:) :: P,Q
INTEGER :: N=0,R,I,J,FILENUM,IOSTATUS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!This fortran script contains a few subroutines to find solutions of a !
!matrix through various methods.(1) Through standard Gauss elimination !
!through upper triangularization.(2) ParPLU Factorization using matrix !
!'P' that stores the row pivioting. (3) PLU Factorization that also    !
!uses a 'P' matrix for row pivioting but also includes code for column !
!pivoting."OPS' is used to measure the operations needed,result in out.!


open(7,file='mdata2.txt')

 do
  read(7,*,iostat=iostatus)
     if(iostatus/=0) THEN !Logic Check-EOF
          exit
     else
          n=n+1
     end if
 end do
!Allocate Storage Needed (based on number of lines in data file)
print*,"Size of matrix:"
!read*,n

n=n-1
print*,n

allocate(a(n,n),b(n),x(n),p(n),q(n),xflu(n))
allocate(xplu(n),acop(n,n),bcop(n),atri(n,n),apar(n,n),aful(n,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

close(7)

print*,"Enter the B:"
!read*,b !Reads the last values of each row
print*,b
ACOP=A
BCOP=B
call TRIU(n,a,b)
ATRI=A
call BackSolve(n,a,b,x)
A=ACOP
B=BCOP
call ParPLU(n,a,p)
APAR=A
call ParPLUSol(n,a,b,p,xplu)
```

```
A=ACOP
B=BCOP
call FullPLU(n,a,p,q)
AFUL=A
call ParPLUSol(n,a,b,p,xflu)
filenum=9
open( unit=filenum, file='results.txt', status='replace')

write(filenum,*)"RESULT:"
write(filenum,*) "A after TRIU call:"
do i=1,n
write(filenum,'(10f10.6)')ATRI(i,1:n)
enddo
write(filenum,*)
write(filenum,*)
write(filenum,*)"X Values:"
write(filenum,'(10f10.6)')x(1:n)
write(filenum,*)
write(filenum,*)"Matrix for the PLU:"
do i=1,n
write(filenum,'(10f10.6)')APAR(i,1:n)
enddo
write(filenum,*)
write(filenum,*)
write(filenum,*)"X Values from PLU:"
write(filenum,'(10f10.6)')xplu(1:n)
write(filenum,*)
write(filenum,*)"Matrix for the FPLU:"
do i=1,n
write(filenum,'(10f10.6)')AFUL(i,1:n)
enddo
write(filenum,*)
write(filenum,*)
write(filenum,*)"X Values from FPLU:"
write(filenum,'(10f10.6)')xflu(1:n)
write(filenum,*)




close(filenum)

!Deallocate matricies
deallocate (a,b,x,p,q,xplu,xflu,acop,bcop)

END PROGRAM A4



!Upper triangular using Gauss elimination and multipliers are stored within the
!A matrix zero entries
Subroutine TRIU(N,A,B)
implicit none
integer:: n,i,j,k
```

```
double precision:: a(n,n), b(n), mult
!A is the coefficient matrix, B are the given solutions to the equations
!mult is the multiplier variable
do k=1,n-1 !Need to have the n-1 since there are no zeros in the last column

     do i=k+1,n !Go along the i th row values

          mult = a(i,k)/a(k,k)
          do j=1,n
               a(i,j)= a(i,j)-(mult*a(k,j)) !Subtract off multiplier
          end do
     b(i) = b(i) - mult*b(k) !Also subtract off in the solutions
     end do

end do
return
end

!Using the upper triangular matrix stored in the coefficient matrix,
!solve backwards for the X variables which are stored in X in main
Subroutine BackSolve(N,A,B,X)
implicit none
integer:: n,i,j
double precision :: a(n,n), b(n), x(n),sub

!Initialize first X value-make sure divide by zero doesn't happen
if(a(n,n).NE.0) THEN
x(n) = b(n)/a(n,n)
else
print*,"A(n,n) = 0 not allowed."
stop
end if
!Going from the next to last element in matrix A
do i=n-1,1,-1
     sub= b(i) !sub is the intermediate variable to help solve X(i)
          do j = i+1,n
               sub = sub-a(i,j)*x(j)
          end do
     x(i) = sub/a(i,i)
end do
return
end




!Subroutine for partial pivoting using the matrix A, the size (n),
!and the permutation matrix P
Subroutine ParPLU(n,a,p)
implicit none
double precision :: a(n,n),mul,maxv,temp(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
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
x(n) = y(n) / a(n,n)
     do i= n-1,1,-1
```

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

!Subroutine for complete pivoting using the matrix A, the size (n),
!and the permutation matrix P.
235 !Note that this algorithm is n^2+n since two max functions are being used
!as a result, the matrix values are being compared n times for the row,
!then n times for the column, then the Gauss elimination is being done.

Subroutine FullPLU(n,a,p,q)
240 implicit none
double precision :: a(n,n),mul,maxv,temp(n)
integer :: n,i,j,k,p(n),q(n), maxr,maxc

!Initialize the permutation matrix
245 do i=1,n
     p(i) = i
end do
!Begin to find the max value in the matrix
do k= 1,n-1
250      maxv= abs(a(k,k))
     maxr= k
     maxc= k
          !Find max row index
          do i = k+1,n
255            if (abs(a(i,k)) >maxv) THEN
                    maxr=i
                    maxv= abs(a(i,k))
               end if
          end do
260


     !If the current row is not the max value
     if(maxr.GT.k) THEN
          p(k) = maxr !Store change in the P matrix
265       temp(k:n) = a(maxr,k:n)
          a(maxr,k:n) = a(k,k:n)
          a(k,k:n) = temp(k:n)

     end if
270       !Find max column index
          maxv=abs(a(1,k)) !Reset Max
          do i = k+1,n
               if (abs(a(1,i)) >maxv) THEN
                    maxc=i
275                 maxv= abs(a(1,i))
               end if
```

```
            end do

        !If the current column is not the max value
        if(maxc.GT.k) THEN
            q(k) = maxc !Store change in the Q matrix (not really needed)
            temp(1:n) = a(1:n,maxc)
            a(1:n,maxc) = a(1:n,k)
            a(1:n,k) = temp(1: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
    return
    end
```

Problem 2

Problem 2 in this report will also cover problems 3 and 4 as well for readability.

2

This problem asks to create a subroutine that computes the Choleski factorization of a given *symmetric* matrix. The problem I chose was worked out and found to have the same matrix 'G' which represents the Choleski factorization matrix.
In order to find if a matrix is positive definite it must have a certain set of characteristics a few key characteristics are outlined below(can be found chapter 6.6 pgs.415-416 of the textbook).

- $x^t Ax > 0$
- $det(A) > 0$
- For i=1,n $A(i,i) > 0$ (diagonal must be positive)

The characteristics above lead to a method of decomposition that is faster than an LU decomposition to solve a system of equations. With positive definitive matrices the identity $A = GG^t$ is used to help speed up finding solutions to a linear system.
In order for the identity to be true the sum of the components of G and its transpose must be result to the original matrix A which is slightly reminiscent of squaring a square root of the components down the diagonal and then to balance the factored matrix 'G', a variation of the Gauss elimination is performed with multipliers which also results in zeros in the upper triangle of G. Line 78 begins the CHO_F subroutine to find the factorization of A matrix.
Algorithm Outline CHOF:

- Start from $j : 1, n$.
- Test if there is a negative or a zero at A(j,j).
- If the evaluation is false, take the square root of A(j,j), store in G(j,j).
- Do from $i : j + 1, n$.
- Store the multiplier of A(i,j) divided by g(j,j).
- Do from $k : j + 1, n$
- Gauss elimination at A(i,k) = A(i,k)- (G(i,j) times G(k,j))

To check the answer simply take the transpose and then multiply the two to see if the original matrix is the result(done at subroutine 'TPOSE').

```
RESULT:
A:
      9.000000      -3.000000       3.000000       9.000000
     -3.000000      17.000000      -1.000000      -7.000000
      3.000000      -1.000000      17.000000      15.000000
      9.000000      -7.000000      15.000000      44.000000

G:
  3.0000000000   0.0000000000   0.0000000000   0.0000000000
 -1.0000000000   4.0000000000   0.0000000000   0.0000000000
```

```
   1.0000000000   0.0000000000   4.0000000000   0.0000000000
   3.0000000000  -1.0000000000   3.0000000000   5.0000000000

 G^T:
   3.0000000000  -1.0000000000   1.0000000000   3.0000000000
   0.0000000000   4.0000000000   0.0000000000  -1.0000000000
   0.0000000000   0.0000000000   4.0000000000   3.0000000000
   0.0000000000   0.0000000000   0.0000000000   5.0000000000


 G*G^T:
   9.0000000000  -3.0000000000   3.0000000000   9.0000000000
  -3.0000000000  17.0000000000  -1.0000000000  -7.0000000000
   3.0000000000  -1.0000000000  17.0000000000  15.0000000000
   9.0000000000  -7.0000000000  15.0000000000  44.0000000000

 Positive Definite?
 T
```

3

The same functions were used to find if the given matrix A is positive definite and what the resulting factorization is.

```
RESULT:
 A:
       7.000000       5.000000      17.000000      10.000000
       5.000000       4.000000      12.000000       7.000000
      17.000000      12.000000      46.000000      24.000000
      10.000000       7.000000      24.000000      15.000000

 G:
   2.6457513111   0.0000000000   0.0000000000   0.0000000000
   1.8898223650   0.6546536707   0.0000000000   0.0000000000
   6.4253960412  -0.2182178902   2.1602468995   0.0000000000
   3.7796447301  -0.2182178902  -0.1543033500   0.8017837257

 G^T:
   2.6457513111   1.8898223650   6.4253960412   3.7796447301
   0.0000000000   0.6546536707  -0.2182178902  -0.2182178902
   0.0000000000   0.0000000000   2.1602468995  -0.1543033500
   0.0000000000   0.0000000000   0.0000000000   0.8017837257


 G*G^T:
   7.0000000000   5.0000000000  17.0000000000  10.0000000000
   5.0000000000   4.0000000000  12.0000000000   7.0000000000
  17.0000000000  12.0000000000  46.0000000000  24.0000000000
  10.0000000000   7.0000000000  24.0000000000  15.0000000000
```

```
 Positive Definite?
 T
```

4

This problem asks to write a subroutine that finds if a matrix that is unsymmetrical is positive definite. Using the characteristic that the diagonal can not have negatives, the subroutine 'PDEF' at line 127 checks only the diagonal of the matrix to see if there are numbers less than zero. The following is the example from problem 3 but with a negative down the diagonal. Even though I wrote a separate function to test the condition, the earlier call to CHO_F checks the same condition and outputs an error to the terminal.

```
mtimbes@mtimbes-VirtualBox:~/CompSci$ ./a.out
 Size of matrix:
           4
  9.000000 -3.000000  3.000000  9.000000
 -3.000000-17.000000 -1.000000 -7.000000
  3.000000 -1.000000 17.000000 15.000000
  9.000000 -7.000000 15.000000 44.000000
 ERROR- SQRT OF A(J,J) NOT POSITIVE DEFINITE. END OF PROGRAM.
```

Listing 2: FORTRAN(F95) Script for Problems 2-4

```
!Michael Timbes
!A4-Prob:2-4
PROGRAM A4_CFACT
IMPLICIT NONE
DOUBLE PRECISION, ALLOCATABLE, DIMENSION(:) :: B,BCOP,X
DOUBLE PRECISION, ALLOCATABLE, DIMENSION(:,:) :: A,ACOP,L,GT,G
CHARACTER :: PD='F'
INTEGER :: N=0,I,J,FILENUM,IOSTATUS
open(7,file='mdata2.txt')

 do
  read(7,*,iostat=iostatus)
     if(iostatus/=0) THEN !Logic Check-EOF
          exit
     else
          n=n+1
     end if
 end do
!Allocate Storage Needed (based on number of lines in data file)
print*,"Size of matrix:"
!read*,n

n=n-1
print*,n
allocate(a(n,n),l(n,n),gt(n,n),b(n),acop(n,n),bcop(n),x(n),g(n,n))
!Return to BOF
REWIND 7
```

```
do i=1,n
read(7,*)(a(i,j),j=1,n)
end do
do i=1,n
print'(10f10.6)',A(i,1:n)
enddo

ACOP=A
BCOP=B
close(7)
call CHO_F(n,a,g)
call TPOSE(n,g,gt,l,1)
A=ACOP
call PDEF(n,a,PD)
!WRITE OUT FILE
filenum=9
open( unit=filenum, file='CFACTRES.txt', status='replace')

write(filenum,*)"RESULT:"
write(filenum,*) "A:"
do i=1,n
write(filenum,'(15f15.6)')ACOP(i,1:n)
enddo
write(filenum,*)
write(filenum,*) "G:"
do i=1,n
write(filenum,'(15f15.10)')G(i,1:n)
enddo
write(filenum,*)
write(filenum,*) "G^T:"
do i=1,n
write(filenum,'(15f15.10)')GT(i,1:n)
enddo
write(filenum,*)
write(filenum,*)
write(filenum,*) "G*G^T:"
do i=1,n
write(filenum,'(15f15.10)')l(i,1:n)
enddo
write(filenum,*)
write(filenum,*)"Positive Definate?"
write(filenum,*)PD


CLOSE(FILENUM)

deallocate(a,b,acop,bcop,x,g,gt,l)
end program A4_CFACT

Subroutine CHO_F(n,a,G)
IMPLICIT NONE
INTEGER :: N,I,K,J
DOUBLE PRECISION :: A(N,N),G(N,N)
```

```
!BEGIN THE FACTORIZATION

DO J=1,N
      IF(A(J,J).GT.0) THEN
      G(J,J) = SQRT(A(J,J))
      ELSE
      PRINT*,"ERROR- SQRT OF A(J,J) NOT POSITIVE DEFINITE. END OF PROGRAM."
      STOP
      ENDIF
      !
      DO I=J+1,N
            G(I,J) = (A(I,J)/G(J,J))
            DO K= J+1,I
                  A(I,K)=A(I,K) - (G(I,J)*G(K,J))
            ENDDO
      ENDDO
ENDDO


RETURN
END
Subroutine TPOSE(n,G,GT,l,comm)
IMPLICIT NONE
INTEGER :: N,I,K,J,comm
DOUBLE PRECISION :: g(n,n),gt(n,n),l(n,n),summ=0

do i=1,n
      gt(i,1:n)=g(1:n,i)
enddo
      !If the command var=1 then do
      IF(COMM.eq.1) THEN
      do k=1,n
            do i=1,n
            summ = 0
                  do j=1,n
                  summ = summ +( g(i,j)*gt(j,k))
                  enddo
            l(i,k)=summ
            enddo
      enddo

      endif
return
end
Subroutine PDEF(n,a,PD)
IMPLICIT NONE
INTEGER :: N,I
DOUBLE PRECISION :: a(n,n)
CHARACTER:: PD
      do i=1,n
      if(a(i,i).LT.0) THEN
            PD='F'
```

```
            stop
      else
      PD='T'
      endif
      enddo
return
end
```