Aim : To solve the simultaneous linear equation using Gauss Elimination Method

Theory:

Gauss Elimination Method :

It is a algorithm use to solve the simultaneous linear equation using row operation on a square matrix made of coefficient of the variables of the linear equation.

Solving the simultaneous linear equations:

In this method use the square matrix of coefficients of the variables in the equations and we also add the column of the constants of the linear equations , then reduce this matrix of coefficient into a identity matrix by row operation and also apply all operation to the constants column also.
And we done this because we know we can write the linear equations like that

$$A\,X = I\,C$$

here A is square matrix of coefficients , X is column matrix of variables , C is constants column matrix and I is the identity matrix.
And we know that X is equal to

$$X = A^{-1}\,C$$

And this can be achieved by apply elementary row operation on the matrix A and also the column of the constants . And that we done in Gauss Elimination Method.

Pivot Condition :

It is happened when the diagonal element is zero or very small that can cause error when we divide other by it.
And this can be over come by flipping the rows only with those have highest diagonal element or it can be done by flipping rows also with flipping column.
This operation is called Pivoting.
This help in reduction of error due to division by small number.

Gauss Elimination Method in programming:

Actually doing this is very simple in programming. This can be done by making a 2D array which have n rows and n+1 column and then we make a loop from 1,n-1 , inside the loop we first check the pivot condition and this can be done by a subroutine and in FORTRAN we can call the matrix (a global variable) from any part that make easy to flip the rows and column of the matrix. Then inside the 1st loop we define another loop from i(index of 1st loop) to n and applying the operation over the row

$$R_j = R_j - (a_{ij}/a_{ii})*R_i$$

here *R* are the rows with index *i* and *j* and *a* is the element of the matrix

And this can be done by a another loop over the columns of the row. And then we repeat the same process for loop from n-1 to 1 and also make the divide the whole row with aii element and here we
can't apply the pivoting because it doesn't work for this.

Program in FORTRAN 95:

```
program gauss_elimination

    implicit none

    ! declaring the variables

    real , dimension(:,:),allocatable :: matrix
    real , dimension(:) , allocatable ::  solution , variable
    integer :: order , i , j , k , pivot_status
    real :: determinant , sol , mul

    ! getting the number of the variable and equation and validating it

    print *, "Enter the number of the variable :: "
    read *, order

    if(order<=0) then
       stop "Invalid number of the variable"
    endif

    ! allocate the matrix and the variable array in the heap memory

    allocate(matrix(order,order+1),variable(order))

    ! the variable array use to track the flip column operation and find which variable is where

    ! defining the variable array

    do i = 1,order
```

```
    variable(i) = i
  enddo

  ! getting the equation the form of the matrix

  print *, "Enter the equation in the form of the matrix :: "
  do i = 1, order
    do j = 1,order+1
      if (j > order) then
        print *, "Enter the constant term in ",i,"th equation :: "
        read *, matrix(i,order+1)
      else
        print *, "Enter ",i,j,"th element :: "
        read *, matrix(i,j)
      endif
    enddo
  enddo

  ! getting the pivot status from the user and validate it

  print *, "Enter the status of the pivot /n for no pivot chose 0,for half choose 1, for full choose 2"
  read *, pivot_status

  if(pivot_status < 0 .or. pivot_status > 2) then
    stop "Invalid pivot option "
  endif

  ! making the upper triangular matrix from the given matrix by row operation

  do i = 1,order-1
1    if (matrix(i,i) == 0) then
      do k = i+1,order ! check that the pivot element is zero or not
        if (matrix(k,i) .ne. 0) then
          call flip(k,i) ! changing the pivot element by flipping the row
          goto 1
        endif
      enddo
    endif
    call pivot(i) ! doing pivoting in the matrix with the status user define
    do j = i+1,order
      mul = matrix(j,i)/matrix(i,i)
      do k = i+1,order+1
        matrix(j,k) = matrix(j,k) - (mul)*matrix(i,k)
      enddo
    enddo
  enddo
```

```
! find the determinant of the upper triangular matrix and check it is not equal to zero

determinant = 1

do i = 1,order
   determinant = determinant*matrix(i,i)
enddo

if (determinant == 0) then
   stop "No Solution is exist for these equation "
endif

! allocating the solution array in which we store the solution for equation

allocate(solution(order))

! here find the solution and store it into solution array

do i = order,1,-1
   sol = matrix(i,order + 1)
   do j = i,order
      sol = sol - matrix(i,j)*solution(j)
   enddo
   solution(i) = sol/matrix(i,i)
enddo

! deallocating the matrix array to clear heap memory

deallocate(matrix)

print *, "The values of the variable are :: "

! finding the match for the variable and solution of the equations

do i = 1,order
   print *, "Value of ",i,"th element is :: "
   do j = 1,order
      if (i == variable(j)) then
         print *, solution(j)
      endif
   enddo
enddo

! clearing the heap memory by deallocating the all array stored
```

```
deallocate(variable,solution)

stop

! subroutine use for the operations

contains
    subroutine flip(row1,row2) ! this subroutine use to flip the rows

        integer ,intent(in)::row1,row2
        real :: temp_element
        integer :: i

        do i = 1,order+1
            temp_element = matrix(row1,i)
            matrix(row1,i) = matrix(row2,i)
            matrix(row2,i) = temp_element
        enddo

        return
    end subroutine flip

    subroutine flip_col(col1,col2) ! this subroutine use to flip the column

        integer , intent(in) :: col1,col2
        real :: temp_element
        integer :: i

        do i = 1,order
            temp_element = matrix(i,col1)
            matrix(i,col1) = matrix(i,col2)
            matrix(i,col2) = temp_element
        enddo
        temp_element = variable(col1) ! this is also flip the corresponding variables in the variable array
        variable(col1) = variable(col2)
        variable(col2) = temp_element

        return
    end subroutine flip_col

    subroutine pivot(row) ! this subroutine is use for to do pivoting in the matrix per user define paviting
status ( pavit_status)

        integer , intent(in) :: row
        real :: pivot_value
        integer :: row_max_pivot , i , col_max_pivot , j
```

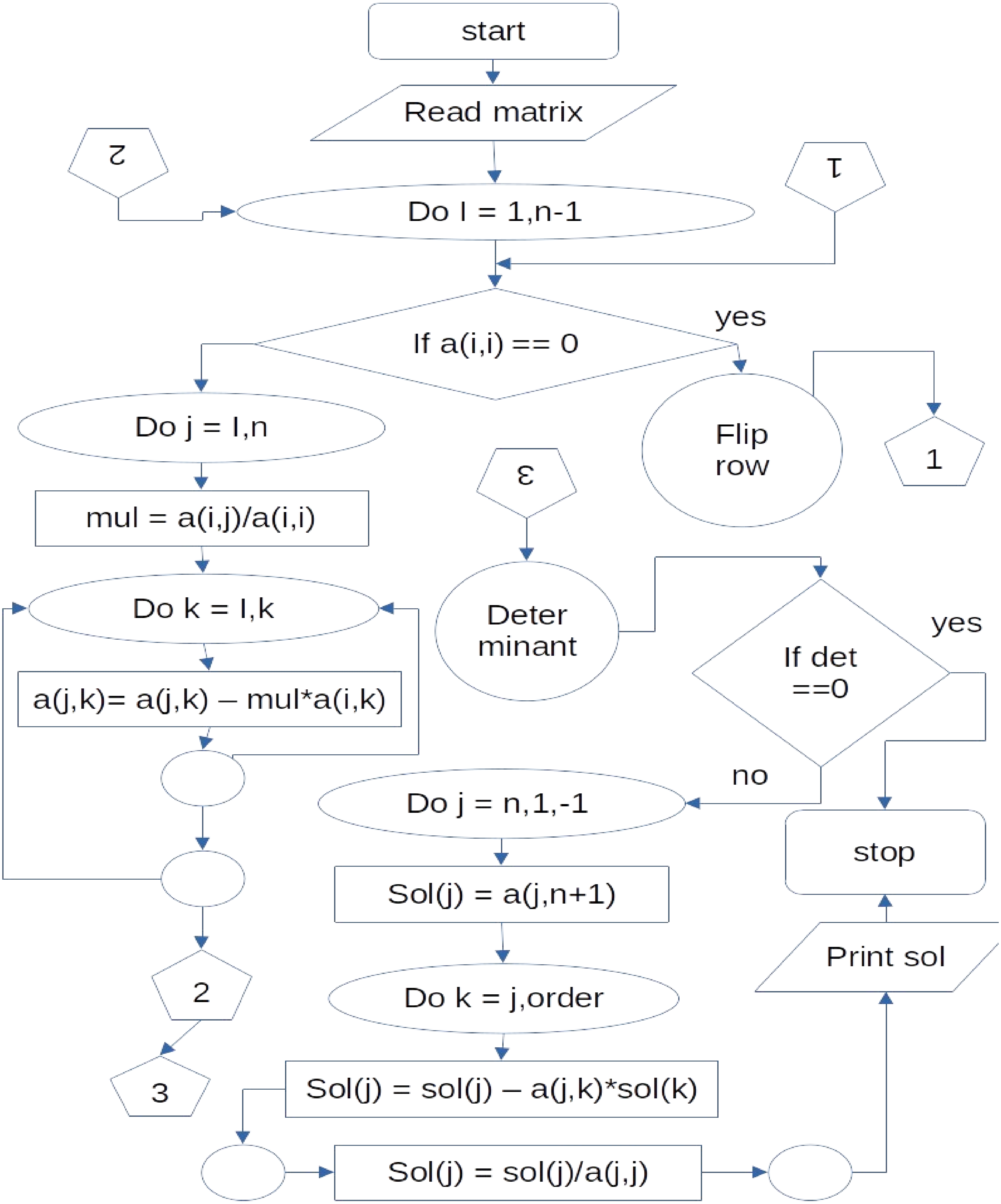
```
        if (pivot_status == 0) then ! for no pivoting
           return

        else if (pivot_status == 1) then ! for half pivoting
           pivot_value = matrix(row,row)
           do i = row,order
              if (matrix(i,row) > pivot_value) then
                 row_max_pivot = i
                 pivot_value = matrix(i,row)
              endif
           enddo
           call flip(row,row_max_pivot) ! only flip the row which have maximum pivot element
           return

        else if (pivot_status == 2) then ! for do full pivoting in the matrix
           pivot_value = matrix(row,row)
           do i = row,order
              do j = row,order
                 if (pivot_value < matrix(i,j)) then
                    pivot_value = matrix(i,j)
                    row_max_pivot = i
                    col_max_pivot = j
                 endif
              enddo
           enddo
           call flip(row,row_max_pivot) ! here we check the element in the whole remaining matrix
           call flip_col(row,col_max_pivot)
           return
        endif

        return
     end subroutine pivot

end program
```

Output :

```
 Enter the number of the variable ::
4
 Enter the equation in the form of the matrix ::
 Enter           1           1 th element ::
1
 Enter           1           2 th element ::
1
 Enter           1           3 th element ::
0.5
```

```
 Enter            1           4 th element ::
1
 Enter the constant term in            1 th equation ::
3.5
 Enter            2           1 th element ::
-1
 Enter            2           2 th element ::
2
 Enter            2           3 th element ::
0
 Enter            2           4 th element ::
1
 Enter the constant term in            2 th equation ::
-2
 Enter            3           1 th element ::
-3
 Enter            3           2 th element ::
1
 Enter            3           3 th element ::
2
 Enter            3           4 th element ::
1
 Enter the constant term in            3 th equation ::
-3
 Enter            4           1 th element ::
-1
 Enter            4           2 th element ::
0
 Enter            4           3 th element ::
0
 Enter            4           4 th element ::
2
 Enter the constant term in            4 th equation ::
0
 Enter the status of the pivot /n for no pivot choose 0,for half choose 1, for full choose 2
2
 The values of the variable are ::
 Value of           1 th element is ::
   2.16216207
 Value of           2 th element is ::
 -0.459459543
 Value of           3 th element is ::
   1.43243217
 Value of           4 th element is ::
   1.08108115
```

Flow Chart:



start
Read matrix
2
Do I = 1,n-1
1
If a(i,i) == 0
yes
Do j = I,n
Flip row
1
3
mul = a(i,j)/a(i,i)
Do k = I,k
Deter minant
If det ==0
yes
a(j,k)= a(j,k) – mul*a(i,k)
no
Do j = n,1,-1
stop
Sol(j) = a(j,n+1)
2
Print sol
Do k = j,order
3
Sol(j) = sol(j) – a(j,k)*sol(k)
Sol(j) = sol(j)/a(j,j)