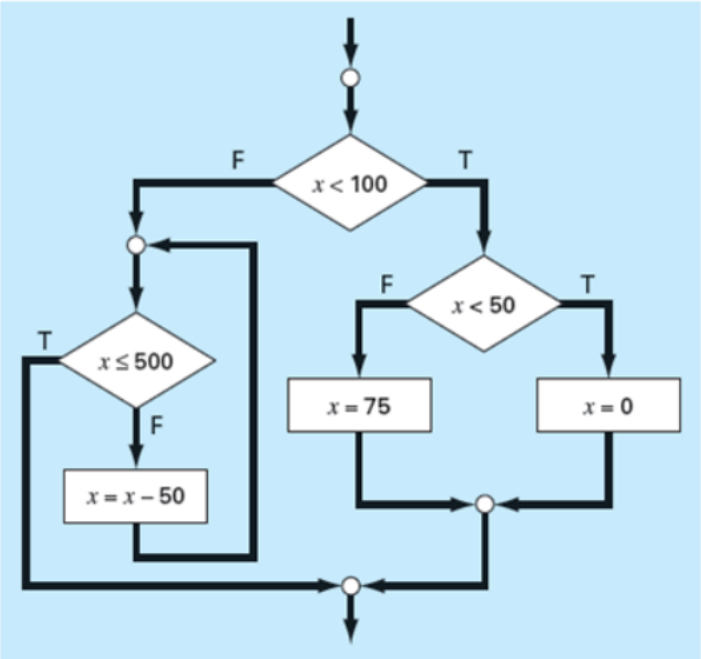
EXAMPLES OF PSEUDOCODE

1) Pseudocode for fixed-point iteration.

```
FUNCTION Fixpt(x0, es, imax, iter, ea)
  xr = x0
  iter = 0
  DO
    xrold = xr
    xr = g(xrold)
    iter = iter + 1
    IF xr ≠ 0 THEN
      ea = |(xr − xrold)/xr| · 100
    END IF
    IF ea < es OR iter ≥ imax EXIT
  END DO
  Fixpt = xr
END Fixpt
```

2) Write pseudocode to implement the flowchart in Figure.



The two if conditions and one do exit loop is inside main if statement. The pseudo code can be written as follows:

```
IF x < 100 THEN
    IF x < 50 THEN
        x = 0
    ELSEIF
        x = 75
    ENDIF
ELSEIF
    IF x <= 500
      DO
        x = x - 50
        IF x <= 500 EXIT
      ENDDO
ENDIF
```

3) Write a pseudocode to perform (a) forward elimination and (b) back substitution.

(a)

```
DOFOR k = 1, n − 1
 DOFOR i = k + 1, n
   factor = a_{i,k} / a_{k,k}
   DOFOR j = k + 1 to n
     a_{i,j} = a_{i,j} − factor · a_{k,j}
   END DO
   b_i = b_i − factor · b_k
 END DO
END DO
```

(b)

```
x_n = b_n / a_{n,n}
DOFOR i = n − 1, 1, −1
  sum = b_i
  DOFOR j = i + 1, n
    sum = sum − a_{i,j} · x_j
  END DO
  x_i = sum / a_{i,i}
END DO
```

4) Write a Pseudocode to implement partial pivoting.

```
p = k
big = |a_{k,k}|
DOFOR ii = k+1, n
  dummy = |a_{ii,k}|
  IF (dummy > big)
     big = dummy
     p = ii
  END IF
END DO
IF (p ≠ k)
  DOFOR jj = k, n
     dummy = a_{p,jj}
     a_{p,jj} = a_{k,jj}
     a_{k,jj} = dummy
  END DO
  dummy = b_p
  b_p = b_k
  b_k = dummy
END IF
```

5) Write a pseudocode to implement Gauss elimination with partial pivoting.

```
SUB Gauss (a, b, n, x, tol, er)
  DIMENSION s(n)
  er = 0
  DOFOR i = 1, n
    s_i = ABS(a_{i,1})
    DOFOR j = 2, n
      IF ABS(a_{i,j})>s_i THEN s_i = ABS(a_{i,j})
    END DO
  END DO
  CALL Eliminate(a, s, n, b, tol, er)
  IF er ≠ -1 THEN
     CALL Substitute(a, n, b, x)
  END IF
END Gauss

SUB Eliminate (a, s, n, b, tol, er)
  DOFOR k = 1, n - 1
    CALL Pivot (a, b, s, n, k)
    IF ABS (a_{k,k}/s_k) < tol THEN
      er = -1
      EXIT DO
    END IF
    DOFOR i = k + 1, n
      factor = a_{i,k}/a_{k,k}
      DOFOR j = k + 1, n
        a_{i,j} = a_{i,j} - factor*a_{k,j}
      END DO
      b_i = b_i - factor * b_k
    END DO
  END DO
  IF ABS(a_{n,n}/s_n) < tol THEN er = -1
END Eliminate

SUB Pivot (a, b, s, n, k)
  p = k
  big = ABS(a_{k,k}/s_k)
  DOFOR ii = k + 1, n
    dummy = ABS(a_{ii,k}/s_{ii})
    IF dummy > big THEN
       big = dummy
       p = ii
    END IF
  END DO
  IF p ≠ k THEN
     DOFOR jj = k, n
       dummy = a_{p,jj}
       a_{p,jj} = a_{k,jj}
       a_{k,jj} = dummy
     END DO
     dummy = b_p
     b_p = b_k
     b_k = dummy
     dummy = s_p
     s_p = s_k
     s_k = dummy
  END IF
END pivot

SUB Substitute (a, n, b, x)
  x_n = b_n/a_{n,n}
  DOFOR i = n - 1, 1, -1
    sum = 0
    DOFOR j = i + 1, n
      sum = sum + a_{i,j} * x_j
    END DO
    x_n = (b_n - sum) / a_{n,n}
  END DO
END Substitute
```

6)

The syntax of MATLAB resembles that of FORTRAN. To get an idea of the similarities, let us compare the codes written in the two languages for solution of simultaneous equations $\mathbf{Ax} = \mathbf{b}$ by Gauss elimination. Here is the subroutine in FORTRAN 90:

```
subroutine gauss(A,b,n)
   use prec_mod
   implicit none
   real(DP), dimension(:,:), intent(in out) :: A
   real(DP), dimension(:),   intent(in out) :: b
   integer, intent(in)                      :: n
   real(DP) :: lambda
   integer  :: i,k
 ! --------------Elimination phase--------------
   do k = 1,n-1
      do i = k+1,n
         if(A(i,k) /= 0) then
            lambda = A(i,k)/A(k,k)
            A(i,k+1:n) = A(i,k+1:n) - lambda*A(k,k+1:n)
            b(i) = b(i) - lambda*b(k)
         end if
      end do
   end do
 ! ------------Back substitution phase----------
   do k = n,1,-1
      b(k) = (b(k) - sum(A(k,k+1:n)*b(k+1:n)))/A(k,k)
   end do
   return
end subroutine gauss
```

The equivalent MATLAB function is (MATLAB does not have subroutines):

```
function b = gauss(A,b)
n = length(b);
%-----------------Elimination phase-------------
for k = 1:n-1
    for i = k+1:n
```

```
        if A(i,k) ~= 0
            lambda = A(i,k)/A(k,k);
            A(i,k+1:n) = A(i,k+1:n) - lambda*A(k,k+1:n);
            b(i)= b(i) - lambda*b(k);
         end
      end
end
%--------------Back substitution phase-----------
for k = n:-1:1
    b(k) = (b(k) - A(k,k+1:n)*b(k+1:n))/A(k,k);
end
```

7)Matrix multiplication

```
SUBROUTINE Mmult (a, b, c, m, n, l)
DOFOR i = 1, n
  DOFOR j = 1, l
    sum = 0.
    DOFOR k = 1, m
      sum = sum + a(i,k) · b(k,j)
    END DO
    c(i,j) = sum
  END DO
END DO
```

8) LU decomposition pseudocode for a subroutine to implement the decomposition phase in LU

```
SUB Decompose (a, n)
  DOFOR k = 1, n − 1
    DOFOR i = k +1, n
      factor = a_i,k/a_k,k
      a_i,k = factor
      DOFOR j = k + 1, n
        a_i,j = a_i,j − factor * a_k,j
      END DO
    END DO
  END DO
END Decompose
```

The following is pseudocode for a subroutine to implement both substitution phases:

```
SUB Substitute (a, n, b, x)
  'forward substitution
  DOFOR i = 2, n
    sum = b_i
    DOFOR j = 1, i − 1
      sum = sum − a_i,j * b_j
    END DO
    b_i = sum
  END DO
  'back substitution
  xn = bn/an,n
  DOFOR i = n − 1, 1, −1
    sum = 0
    DOFOR j = i + 1, n
      sum = sum + a_i,j * x_j
    END DO
    x_i = (b_i − sum)/a_i,i
  END DO
END Substitute
```

9) An algorithm to implement an *LU* decomposition expression of Gauss elimination is

```
SUB Ludecomp (a, b, n, tol, x, er)
  DIM o_n, s_n
  er = 0
  CALL Decompose(a, n, tol, o, s, er)
  IF er <> -1 THEN
     CALL Substitute(a, o, n, b, x)
  END IF
END Ludecomp

SUB Decompose (a, n, tol, o, s, er)
  DOFOR i = 1, n
    o_i = i
    s_i = ABS(a_i,1)
    DOFOR j = 2, n
      IF ABS(a_i,j)>s_i THEN s_i = ABS(a_i,j)
    END DO
  END DO
  DOFOR k = 1, n - 1
    CALL Pivot(a, o, s, n, k)
    IF ABS(a_o(k),k/s_o(k)) < tol THEN
       er = -1
       PRINT a_o(k),k/s_o(k)
       EXIT DO
    END IF
    DOFOR i = k + 1, n
       factor = a_o(i),k/a_o(k),k
       a_o(i),k = factor
       DOFOR j = k + 1, n
        a_o(i),j = a_o(i),j - factor * a_o(k),j
       END DO
    END DO
  END DO
  IF ABS(a_o(k),k/s_o(k)) < tol THEN
     er = -1
     PRINT a_o(k),k/s_o(k)
  END IF
END Decompose

SUB Pivot (a, o, s, n, k)
  p = k
  big = ABS(a_o(k),k/s_o(k))
  DOFOR ii = k + 1, n
    dummy = ABS(a_o(ii),k/s_o(ii))
    IF dummy > big THEN
       big = dummy
       p = ii
    END IF
  END DO
  dummy = o_p
  o_p = o_k
  o_k = dummy
END Pivot

SUB Substitute (a, o, n, b, x)
  DOFOR i = 2, n
    sum = b_o(i)
    DOFOR j = 1, i - 1
      sum = sum - a_o(i),j * b_o(j)
    END DO
    b_o(i) = sum
  END DO
  x_n = b_o(n)/a_o(n),n
  DOFOR i = n - 1, 1, -1
    sum = 0
    DOFOR j = i + 1, n
      sum = sum + a_o(i),j * x_j
    END DO
    x_i = (b_o(i) - sum)/a_o(i),i
  END DO
END Substitute
```

10) LU decomposition in matrix inverse using subroutines in example 9.

```
CALL Decompose (a, n, tol, o, s, er)
IF er = 0 THEN
   DOFOR i = 1, n
     DOFOR j = 1, n
       IF i = j THEN
          b(j) = 1
       ELSE
          b(j) = 0
       END IF
     END DO
     CALL Substitute (a, o, n, b, x)
     DOFOR j = 1, n
       ai(j, i) = x(j)
     END DO
   END DO
   Output ai, if desired
ELSE
   PRINT "ill-conditioned system"
END IF
```