The algorithm for the two methods in this program are not difficult but needs the first and second derivative, the gradient, inverted hessian matrix. To do all of that job the program contains `FUNCTION`'s to be the most general that we can.

For the two methods is obligatory has a starting point, so the program asks the coordinate starting point, also compute the value of the function in these point. The function is also in a `FUNCTION`, to change for any other function that we want only changing one line of the code.

1. Steepest Descent

```
1  iter = 1; er = 1.
2  DO WHILE ( er > e .AND. iter <= max_iter )
3      coord(iter,1) = x_0;    coord(iter,2) = y_0
4      g(:) = gradient(coord(iter,1),coord(iter,2))
5      g(:) = g(:)/NORM2(g)
6      WRITE(*,FMT='(I9,3F8.3,A4,F6.3,A1,F6.3,A1)') iter-1, x_0,&
7                               y_0, z_0,'   (',g(1),',',g(2),')'
8      x_0 = x_0 - alpha*g(1)
9      y_0 = y_0 - alpha*g(2)
10     z_0 = f(x_0,y_0)
11     er = ABS(z_0 - tmp_z_0)
12     tmp_z_0 = z_0
13     iter = iter + 1
14 ENDDO
```

For Steepest Descent we need a `DO WHILE` loop to do the algorithm until the convergence criterion or the maximum number of iterations, the thing that happens first. Saving the coordinates of every step in a matrix, we compute the gradient and normalize the vector. With that values write in the screen the coordinates and the normalized gradient. Lines 3 to 6.

After that the program compute the next coordinate values with the previous coordinate, the gradient normalized and the step size (variable `alpha`)

Since the step of size is not small and we are normalizing the gradient the program does not converge, and we get a loop between two points, one going to the other and vice versa.

The line 11 is to get the difference between the function valued in the point and the previous step, saving the value in `tmp_z_0`

2. Newton-Raphson

```
DO WHILE ( er > e .AND. iter < max_iter )
    g = gradient(coord(iter,1),coord(iter,2))
    hess = hessian(coord(iter,1),coord(iter,2))
    invhess = inverse(hess,2)
    coord(iter+1,:) = coord(iter,:) - MATMUL(invhess,g)
    x_0 = coord(iter,1);  y_0 = coord(iter,2); z_0 = f(coord(iter,1),coord(iter,2))
    WRITE(*,FMT='(I9,3F8.3,A4,F6.3,A1,F7.3,A5,4F7.3,A1)') iter-1, x_0,&
                  y_0, z_0, '  (',g(1),',',g(2),')   (', hess,')'
    er = NORM2(g)
    iter = iter + 1
ENDDO
```

For the Newton-Raphson method we start deleting the values in the matrix `coord`, just staying with the starting point. Then with a `DO WHILE` with the criterion of the threshold convergence and the maximum number of iterations the program compute the gradient without normalizing.

Also compute the hessian matrix and the inverted hessian matrix, using `FUNCTIONS`'s one to compute the hessian matrix and another one to compute the inverse of any matrix.

Then compute the next coordinates with the previous one and the multiplication between the inverse hessian matrix and the gradient. Saving the values in the variables `x_0`, `y_0` and `z_0` to writing the values in the screen.

The convergence criterion with the threshold 10^{-8} is with the norm of the gradient, knowing that in minimize the gradient most be zero.

3. Functions

a) Numerical derivative

```
FUNCTION f_prime(d,x,y) RESULT (prime)
IMPLICIT NONE
REAL*8 :: h, x_p, y_p, x, y, prime
INTEGER :: d
! d=1 :: \frac{\partial}{\partial x}
! d=2 :: \frac{\partial}{\partial y}
h = 0.00001d0
x_p = x;    y_p = y
IF (d == 1) x_p = x_p + h
IF (d == 2) y_p = y_p + h
prime = ( f(x_p,y_p) - f(x,y) ) / h
ENDFUNCTION f_prime
```

To compute numerically the derivative, the `FUNCTION` needs the coordinates, know with respect which variable is the derivative, the variable `d` is for the last thing. Using the definition of derivative as a limit, the `FUNCTION` compute that with a small `h`.

b) Numerical second derivative

```
FUNCTION f_dprime(d,x,y) RESULT (dprime)
IMPLICIT NONE
REAL*8 :: h, x, y, dprime, fl, fr
INTEGER :: d
! d=1 :: \frac{\partial^2}{\partial x^2}
! d=2 :: \frac{\partial^2}{\partial x \partial y}
! d=3 :: \frac{\partial^2}{\partial y \partial x}
! d=4 :: \frac{\partial^2}{\partial y^2}
h = 0.00001d0
SELECTCASE(d)
    CASE(1)
        fl = f_prime(1,x-h,y)
        fr = f_prime(1,x+h,y)
    CASE(2)
        fl = f_prime(1,x,y-h)
        fr = f_prime(1,x,y+h)
    CASE(3)
        fl = f_prime(2,x-h,y)
        fr = f_prime(2,x+h,y)
    CASE(4)
        fl = f_prime(2,x,y-h)
        fr = f_prime(2,x,y+h)
ENDSELECT
dprime = (fr-fl)/(2.d0*h)
ENDFUNCTION
```

As the previous `FUNCTION` we use a definition of derivative to approximate computationally the value. Now using the definition of the second symmetric derivative:

$$\mathop{\text{lím}}_{h\to 0} \frac{f(x+h) - 2f(x) + f(x-h))}{h^2} \tag{1}$$

with a small value to the variable `h`.

And using the first argument of the `FUNCTION` to know respect which is the derivative. Lines 5 to 8, commented.

c) Gradient and Hessian Matrix

```
FUNCTION gradient(x,y) RESULT(g)
IMPLICIT NONE
REAL*8 :: x, y
REAL*8, DIMENSION(2) :: g

g(1) = f_prime(1,x,y)
g(2) = f_prime(2,x,y)
ENDFUNCTION

FUNCTION hessian(x,y) RESULT(hess)
IMPLICIT NONE
REAL*8 :: x, y
REAL*8, DIMENSION(2,2) :: hess

hess(1,1) = f_dprime(1,x,y)
hess(1,2) = f_dprime(2,x,y)
hess(2,1) = f_dprime(3,x,y)
hess(2,2) = f_dprime(4,x,y)
ENDFUNCTION
```

To compute the gradient and the hessian is just have the corrects derivatives in the correct form. These two `FUNCTION`'s does not need more comments.

d) Inverse Matrix

The last subsection we need was written as one of the programs in the intensive course homework. The extension is not too short, but is easy, is the Gaussian-Jordan algorithm to get the inverse matrix.

```
FUNCTION inverse(M,n) RESULT (inv)
IMPLICIT NONE
INTEGER :: i, j, k, n, l, aux3
REAL*8 :: aux, aux2
REAL*8, DIMENSION(n,n) :: M, inv
REAL*8, DIMENSION(n,2*n) :: S
REAL*8, DIMENSION(2*n) :: V
!---- First, add the identity
S(:,:) = 0.
DO i = 1, n
    aux3 = n+i             !"diagonal" on the right side
    S(i,aux3) = 1.
ENDDO
DO i = 1, n
    DO j = 1, n
        S(i,j) = M(i,j)    !data in the left side
    ENDDO
ENDDO
!---- 2nd up diagonal
DO i = 1, n-1
    aux = S(i,i)           !diagonal elemnt
    DO j = 1, 2*n
        V(j) = S(i,j)/aux !divided by the diagonal elemnt
    ENDDO
    DO k = i+1, n
        aux2 = S(k,i)
```

```
        DO j = 1, 2*n      !Actualize the matrix
            S(k,j) = S(k,j) - V(j)*aux2
        ENDDO
    ENDDO
ENDDO
!---- 3rd diagonal, zeros in the up part of matrix
DO i = 1, n-1
    DO l = i+1, n
        DO k = 1, n
            aux = S(k,k)
            DO j = 1, 2*n
                V(j) = S(k,j)/aux
            ENDDO
            aux2 = S(i,k)
            DO j = 1, 2*n
                S(i,j) = S(i,j) - V(j)*aux2
            ENDDO
        ENDDO
    ENDDO
ENDDO
!---- 4th diagonal equals to one
DO i = 1, n
    aux = S(i,i)
    DO j = 1, 2*n
        S(i,j) = S(i,j)/aux
    ENDDO
ENDDO
!---- only the inverse
DO i = 1, n
    DO j = 1, n
        aux3 = n+j
        inv(i,j) = S(i,aux3)
    ENDDO
ENDDO
ENDFUNCTION
```