**Root-Finding and Function Optimization**

**“Using Steepest Descend method and Newton-Raphson method”**

**Brief description:**

This work is based on the use of two different methods for the optimization or minimization of a function defined by two variables (Fig. 1) from two established methods: the Steepest Descend method and the Newton-Raphson method.

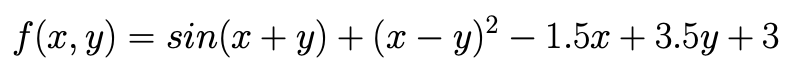
****

Fig. 1: *Function to optimize*

The created codes have been produced using the programming language called Fortran, which is a language used for mathematical calculation and, at the same time, very used in the scientific field. Throughout this work we define the two methods previously mentioned and, at the same time, the type of code used, in order to understand how it works. The whole set of variables and structures are explained together with the reason for the use of each one.

**Algorithm:**

The algorithms or methods used in this work are two: Steepest Descend method and Newton-Raphson method. These algorithms are used for the minimization of various functions. For each of the methods a different Code has been made in each case, in order to be able to verify the effectiveness and use of each one separately.

First of all, it is necessary to explain each of the methods in order to know how they work and the theory behind them:  
  
  
- Steepest Descent method:

The Steepest Descent method is used for the minimization of various functions, that means finding the minimum of that function, which can be of great help in different fields of science and mathematics. This method is based on numerical method, which is based on starting at a localized point of the function and trying to descend to the minimum point using information from the first derivative (gradient descent). Instead of finding minima by manipulating symbols, gradient descent approximates the solution with numbers.

To minimize the function, it follows the negative of the gradient (first derivative of the function), and thus allows us to go in the direction of the steepest descent. This is the general definition of gradient descent. Formally, if we start at an initial point and move a positive distance in the direction of the negative gradient, then our new point will be closer to the minimum value of the function. And finally, starting from the initial point, we keep improving little by little, from iterations, until we find a local minimum.

- Newton-Raphson method:

The Newton-Raphson method allows finding a root of a nonlinear equation as long as one starts from a good initial estimate of the equation. Newton's iterative scheme can be derived from the Taylor development of the function around the initial estimate.

This method does not need a closed interval [a,b] but it does need an evaluation of the function and at the same time its derivative. The general procedure is simple, but must be understood part by part; first, if the function is continuous and differentiable, it is approximated to a given point from its tangent using the first Taylor polynomial. At this point we evaluate the point on the abscissa axis where the tangent line crosses and makes the function equal to zero. This new point approximates in a better way the value that makes the function zero, in this way, the procedure described above is performed iteratively by changing the values of the variable or variables. It should be noted that all the changes and calculations are explained in the following section. Finally, the general algorithm is based on choosing an initial point, evaluating the function at that point and according to the result proceed to start the iterative calculation by changing the variable or variables of the function and calculating the first derivatives at those points and at the same time, evaluating those results.

**Implementation of the code:**

This final part of the paper discusses the general structure of the code used for each of the methods, their parts, subroutines and also the main variables used. It must be said that the programs have been tested for correct operation. In this case, two different codes have been made, one for each method explained. Each of the programs is defined from a set of subroutines and main variables. The structure and way of calculation are explained below:

For the first program (Steepest Descent method), the principal variables, among others, are the following:

nvar 🡪 number of variables in the function, dimension of gradient vector.

distance 🡪 distance between two successive points upon optimization.

vnorm 🡪 norm (length) of the gradient vector.

The structure of this code is based on three distinct parts: first a subroutine is placed to calculate the gradient vector applied in the minimization method, then the main program section where the code for the steepest descent method is defined and, finally, the function to be optimized is defined.

The only subroutine of the program is used to calculate the gradient vector used. First the dimensions of the vectors used are defined, then the value of h is calculated and a do loop is entered which goes from 1 to the number of variables of the function to optimize, which define the dimension of the vector. Within this loop the finite difference method is used to calculate each of the elements of the gradient and at the same time a function is used to recover the vector for the next iteration. Finally, the gradient is normalized using a do loop.

Then we enter the part corresponding to the main program, where we start defining the dimensions of the matrix and, at the same time, of the gradient vector. Some parameters are defined before entering a do loop (this set of parameters are defined within the program itself). This do loop refers to the Steepest Descent method procedure itself. First the previously defined subroutine is called and all the values of each iteration are written on the screen. Then a do loop is defined within it, to define the calculation error, to calculate this error the distances between successive points are calculated, the value of the function at the previous point and the current one, and the absolute difference between successive points of the function is calculated. Finally, the iteration number and the calculation of the next point in the sequence are calculated.

Finally, the results are written once the calculation is finished and in the last part of the program the function to be optimized is defined using implicit double precision to obtain better results.

For the second program (Newton-Raphson method), the main variables, among others, are the following:

nvar 🡪 number of variables in the function.

r\_hess 🡪 hessian matrix.

x 🡪 nvar-dimensional vector submitted to function evaluation (within the subroutines "hessian" and "gradient" it is referred as vec).

x\_new 🡪 nvar-dimensional to keep track of updates (equivalent to x, but different within the iteration).

Distance 🡪 distance between two successive points upon optimization (calculated as square root of sum over (x\_i(new) - x\_i("old"))^2 + ... + (x\_n(new) - x\_n("old"))^2

v\_norm\_grad 🡪 norm (length) of the gradient vector.

The general structure of the program is divided into five initial subroutines which are complemented throughout the calculation, then the main program is introduced, which uses the named subroutines and finally, the function for optimizer is defined.

The first subroutine refers to the calculation of the Hessian matrix. It starts with the definition of the dimensions of the matrix and the calculation of the variable h. Then a do loop is started with another loop inside it to define the values of each of the terminals that make up the matrix. These loops are performed by taking coefficients from the rows and columns. It should be noted that the calculation in this subroutine is performed from vectors.

The second subroutine is used to calculate the gradient vector. The dimension of the vector is defined and then a do loop is performed to calculate each of the vector terminus, finally the vector is recovered to calculate the next iteration.

The next subroutine refers to the exchange subroutine, this is used for exchanging rows if the matrix to be inverted has zero anywhere in the diagonal, in principle, this subroutine may never be triggered in these kinds of programs, but it should always accompany inversion subroutine (or its functionality be within the inversion subroutine, to be able to invert any invertible matrix).

The penultimate subroutine is used for matrix inversion. First the dimensions of the different matrices used during the calculation are defined. Then the partitioned nx2n matrix is created from a do loop. Next, the Gaussian elimination process is started, this process begins with the adjustment of the first row of the matrix and then starts the process of introducing zeros in the terminus located under the main diagonal of the matrix, this process is performed from two do loops to go through all the positions of the matrix. Finally, the introduction of zeros in the left pivots of the matrix is performed from two do loops, one of which aims to find a constant value to be able to subtract rows. At the end of this part of the program the right part of the calculated matrix is selected and recombined in the inverse, thus finishing the calculation.

To finish and before entering the main program, the subroutine of multiplication of the matrix by the vector is performed, in this way using two do loops, you can perform the calculation using a summation by traversing the positions of the matrix (rows and columns).

The next part of the code corresponds to the main program. In this, certain parameters specified in the code itself are defined and a do while loop is started to evaluate the results of each of the iterations against the convergence value. In this case the subroutines are called and the general calculation of each new point is performed. Then the calculation of the distance between successive points is performed, this is the value that is compared to see if the convergence has been obtained. Finally, the different values and final results are generated and the function to be minimized is described in the last section of the work.