Minimization problem without constraints

The Newton's method

Sofia Almirante, 1527718

December 2022

1 Introduction

The aim of this work is to be able to solve an optimization problem without constraints applying the Newton's method.

We are going to program a basic Newton method in MATLAB, show some examples of its functioning, and argue the advantages, drawbacks and hos is the convergence of the Newton method.

1.1 1. Univariate root finding

For the first exercise, we are going to program a basic Newton method for univariate root finding.

First, we have created a MATLAB function that determines the value of a determined **function** and its derivative and second derivative in a certain point. In the code shown below, we have chosen the function $f(x) = -\frac{1}{\exp(x^2)}$.

```

function [F, FF, FFF] = CALCfun(X)
    fun = @(x) 1/exp(x^2);
    syms x;

    f(x) = fun(x);
    F = f(X);

    ff(x) = diff(f);
    FF = ff(X);

    fff(x) = diff(ff);
    FFF = fff(X);
end
```

The function that contains the **Newton**'s method is the following:

```
function [sol, Fsol, h, NTiter, Error] = Newton(x0, MAXit, tol2)
    NTiter = 0;
    V = zeros(1,MAXit);
    x = x0;
    [F, FF, FFF] = CALCfun(x);
    Error = []
    while abs(FF) > tol2 && NTiter < MAXit
        x = double(x-FF/FFF);
        [F, FF, FFF] = CALCfun(x);
        NTiter = NTiter+1;
        V(NTiter) = x;
        if NTiter>1
            Error = [Error, abs(V(NTiter)-V(NTiter-1))];
        end
    end
    sol = x;
    Fsol = double(FF);

    h = Error(NTiter-1)

end
```

Running this code, we obtain the following outputs:

INPUTS:

- $x_0 = 0.4$
- MAXit = 50,
- tol2 = 10e-10

OUTPUTS:

- NTiter = 4
- h = 5.9207e-06
- sol = 4.1509e-16

We observe for this initial condition, the solution is very closed to the desired one (x = 0), and the error is of the order of 10^{-6}. This is found in a very small amount of iterations (only 4). For this, we can assume the Newton's method has worked when starting with an initial condition close to the solution.

In Fig. 1, we can see the plot of the absolute value of the difference between $x_{k+1} and x_k$ for every iteration with these conditions.

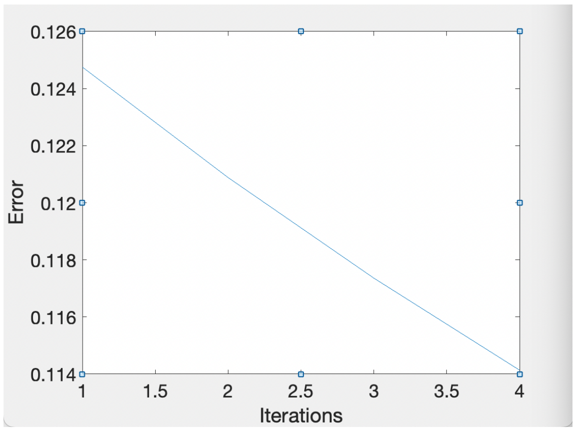

Figure 1: Absolute value of the difference between the value in consecutive iterations ('Error') in function of the iteration for $x0$=0.4.

In Fig. 2 we find the same plot starting at a value more far away from the solution. For this case, we have the following inputs and outputs:

INPUTS:

- $x_0 = 1$
- MAXit = 50,
- tol2 = 10e-10

OUTPUTS:

- NTiter = 21
- h = 0.1033
- sol = 5.0465

The root of the chosen function is at $x = 0$, but it gets very flat from $x = 4$. For this reason, the Newton's method fails finding the solution, as there are more points that have a derivative value close to zero.

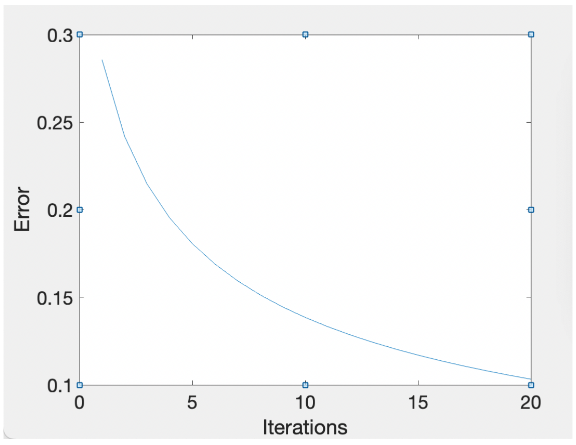

Figure 2: Absolute value of the difference between the value in consecutive iterations ('Error') in function of the iteration for $x0$=1.

As we expected, the Newton's method is very efficient when the starting condition is close to the condition, but might not converge to the solution when starting far from it. This is a very important take away point of this assignment, as it is important to realize the importance of knowing the roots of the function beforehand in order for this method to be useful and trustworthy.

1.2 2. Finding the minimum of a 3D function

For the second exercise, we are going to program a basic Newton method to find the minimum of a 3D function.

In this case, the first **function** is going to determine the value of the function, its Gradient and its Hessian in a determined point, generalizing to 3 dimensions what we did in the previous exercise. The function used as an example is $f(x) = \frac{-1}{\exp(x^2+0.2*y^2)}$. We find the code for this in the code below:

```
function [z,g,h] = CALCfun3D(x0,y0)

    fun = @(x,y) -1/exp(x^2+0.2*y^2);
    syms x y;
    f(x,y) = fun(x,y);
    %X=[x0;y0];
    z = double(f(x0,y0));

    grad(x,y) = gradient(f,[x,y]);
    g = double(grad(x0,y0));
    hess(x,y) = hessian(f,[x,y]);
    h = double(hess(x0,y0));
```

```


end
```

We also show below the code with the **Newton**'s method for 3 dimensions:

```
function [sol, Fsol, error, NTiter, V, Error] = Newton3D(x0, y0, MAXit,
    tol)
    NTiter = 1;
    V = zeros(3, MAXit);
    x = x0;
    y = y0;
    [z, g, h] = CALCfun3D(x,y);
    Error = [];
    while norm(g) > tol && NTiter < MAXit
        V(1,NTiter) = x;
        V(2,NTiter) = y;
        V(3,NTiter) = z;
        NTiter = NTiter+1;

        var = (inv(h)*g).';
        x = x-var(1);
        y = y-var(2);
        [z,g,h] = CALCfun3D(x,y);

        if NTiter > 1
            Error = [Error, norm([V(1,NTiter),V(2,NTiter)]-[V(1,NTiter
                -1),V(2,NTiter-1)])];

        end
    end
    sol = [x,y];
    Fsol = norm(g);
    error = Error(NTiter-1)


end
```

In a first place, we chose an initial condition (x_0, y_0) close to the known solution (0,0). The inputs and outputs resulting are the following ones:

INPUTS:

- $(x_0, y_0) = (0.1, 0.1)$
- MAXit = 50,
- tol2 = 10e-10

OUTPUTS:

- NTiter = 4
- h = 5.0468e-08
- sol = 1.0e-21 *(-0.1191, -0.1059)

As we observed in the last exercise, the method works in a very efficient way when starting from a

In Fig. 3 we have shown the error between two consecutive iterations.

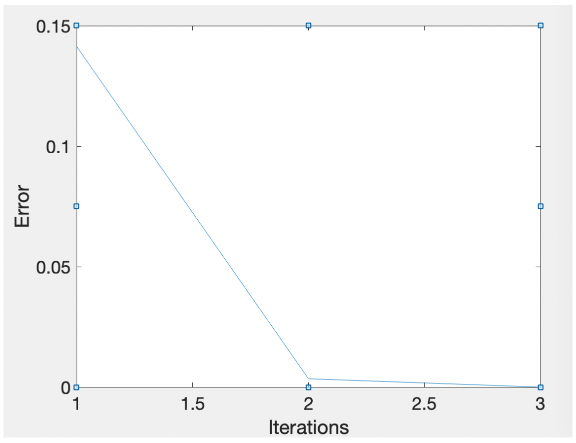

Figure 3: Absolute value of the difference between the value in consecutive iterations ('Error') in function of the iteration for $(x0, y0)$=0.1.

In a last place, we have applied the same method starting from a more far away of the solution initial point. The inputs and outputs are shown below:

INPUTS:

- $(x_0, y_0) = (1, 0.5)$
- MAXit = 50,
- tol2 = 10e-10

OUTPUTS:

- NTiter = 25
- h = 5.7100
- sol = (5.2021, 2.6011)

In Fig. 4, we have represented the error between consecutive iterations.

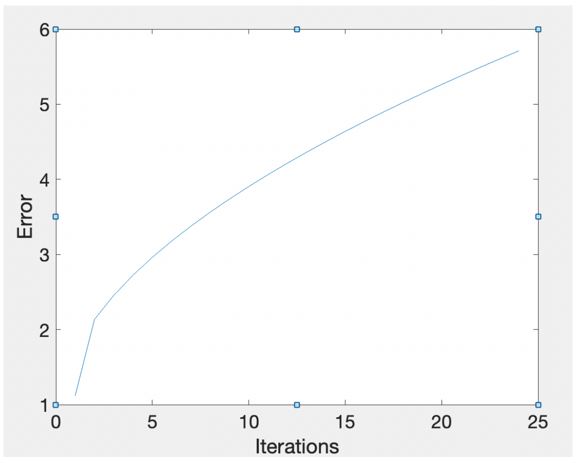

Figure 4: Absolute value of the difference between the value in consecutive iterations ('Error') in function of the iteration for $(x0, y0)$=(1,0.5).

As we did in the previous exercise, we observe that when we are not close enough to the solution in the initial condition, the method might not converge to the actual solution. In this case, as the function is also flat when far from the root, the method converges to another values with derivatives also close to 0, which leads to wrong solutions.

With all this analysis, we acquire two vital ideas regarding Newton's method. When choosing an initial condition close enough to the solution, the method will converge fast and will have a very small amount of error. This makes it a very useful and efficient tool. In the other hand, it is very important to remark that we need to start with a condition close to the solution, for which we would need already the solution. For this reason, it is not the most trustworthy and efficient method when looking for the roots of a function that we don't know the behavior of beforehand.