Homework 4

Sebastian Aguilera Novoa

Table of Contents

1. Unconstrained minimization

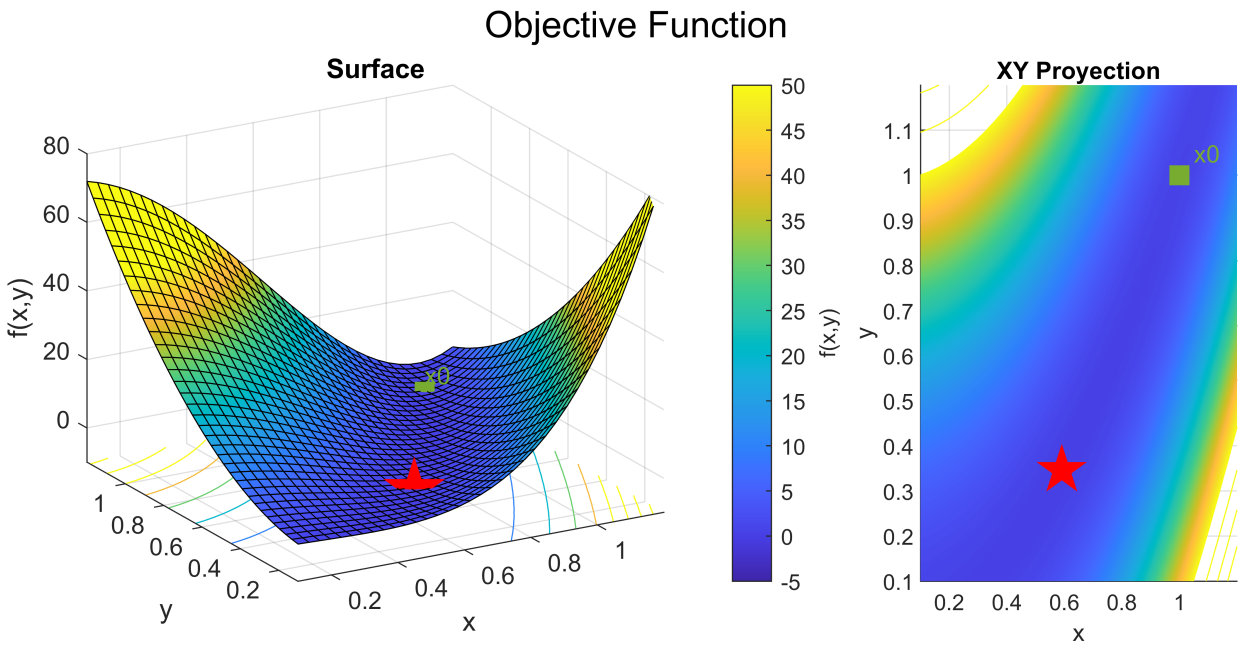
In this exercise we are going to implement different Newton methods to solve the following problem

$$\min_{x,y\in\mathbb{R}} \underbrace{\frac{1}{2}(x-1)^2+\frac{1}{2}(10(y-x^2))^2+\frac{1}{2}y^2}_{=:f(x,y)}$$

Plot the objective function $f(x,y)$ to get an idea of the shape is this.

Objective Function Plot

```
fp1 = @(x,y) 0.5*(x-1).^2 +0.5*(10*(y-x.^2)).^2 + 0.5*y.^2; x1_opt = [0.591077; 0.345913];
FinalPlot(fp1, x1_opt, [1;1], [], 1, [-30 24], "Objective Function", 1) %(f, xstar, x0, xks, fl
```



The red point is the global minimum and the pink one is the starting point.

a) Theorical Gradian and Hessian

First get in papper, the gradient and Hessian matrix of $f(x, y)$. Then, rewrite it in the form $f(x, y) = \frac{1}{2}||R(x, y)||_2^2$

where $R : \mathbb{R}^2 \to \mathbb{R}^3$ is the residual function. Determine the Hessian by the Gauss-Newton method and compare it with the exact. When do the two matrices coincide?

Jacobian

$$\nabla f(x, y) = \Big(x - 1 + 10(y - x^2)(-2x)10,\ 10(y - x^2)10 + y\Big)$$
$$= \Big(x - 1 - 200xy + 200x^3,\ -100x^2 + 101y\Big)$$

The Hessian

$$\nabla^2 f(x, y) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} = \begin{pmatrix} 600x^2 - 200y + 1 & -200x \\ -200x & 101 \end{pmatrix} \tag{1}$$

but this problem can be formulated as a least saquare problem as follows

Least Square Problem

$$\min_{x,y \in \mathbb{R}} \frac{1}{2}||R(x, y)||_2^2$$

with the residual vector is

$$R(x,y) = \Big(x-1, 10(y-x^2), y\Big)$$
$$\frac{1}{2}||R(x,y)||_2^2 = \frac{1}{2}\Big[(x-1)^2 + (10(y-x^2))^2 + y^2\Big]$$

which gives the same problem. Now let's calculate the Jacobian to the new vector field R and then approximate the Hessian by this Jacobian

$$J = \nabla R = \begin{pmatrix} \frac{\partial R_1}{\partial x} & \frac{\partial R_1}{\partial y} \\ \frac{\partial R_2}{\partial x} & \frac{\partial R_2}{\partial y} \\ \frac{\partial R_3}{\partial x} & \frac{\partial R_3}{\partial y} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -20x & 10 \\ 0 & 1 \end{pmatrix}$$

Hessian approximantion by Newton Gauss method

$$H_R = J^T J = \begin{pmatrix} 1 & 0 \\ -20x & 10 \\ 0 & 1 \end{pmatrix}^T \begin{pmatrix} 1 & 0 \\ -20x & 10 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -20x & 0 \\ 0 & 10 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -20x & 10 \\ 0 & 1 \end{pmatrix} \approx H$$
$$H_R \approx \begin{pmatrix} 1+400x^2 & -200x \\ -200x & 101 \end{pmatrix}$$

then (1) coincide with the Hessian approximation if and only if

$$1 + 400x^2 = 600x^2 - 200y + 1$$
$$200y = 200x^2$$
$$y = x^2$$

then the approximate optimization problem is

$$\min_{x\in\mathbb{R}} \ \frac{1}{2}||R(x,y)||_2^2$$
$$\text{s.t.} \qquad y = x^2$$

with

$$R(x) = R(x,x^2) = \Big(x-1, 10(x^2-x^2), x^2\Big)$$
$$\frac{1}{2}||R(x)||_2^2 = \frac{1}{2}\Big[(x-1)^2 + x^2\Big] = \frac{1}{2}(2x^2 - 2x + 1)$$

which is equivalent to solve

$$\min_{x\in\mathbb{R}} \ x^2 - x + \frac{1}{2} = g(x)$$

where the solution can be find by solving $g'(x) = 0 = 2x - 1$

$$x^* = \frac{1}{2}, \ y^* = g(x^*) = \frac{1}{4}$$

then the optimal point is $(0.5, 0.25)$, which is the approximate solution for the approximate optimization problem.

¿is this a good enough approximation?

Function, Gradients and Hessians

```
f        = @(x) 0.5*(x(1)-1)^2 + 0.5*(10*(x(2)-x(1)^2))^2 + 0.5*x(2)^2;
Gradient = @(x) [200*x(1)^3+x(1)-200*x(1)*x(2)-1; -100*x(1)^2+101*x(2)]; % G(x) = H(x,y)
Hessian  = @(x) [600*x(1)^2+1-200*x(2) -200*x(1); -200*x(1) 101];        % H(x) = H(x,y)

R         = @(x) [x(1)-1; 10*(x(2)-x(1)^2); x(2)];
JacobianR = @(x) [1 0; -20*x(1) 10; 0 1];
HessianR  = @(x) [1+400*x(1)^2 -200*x(1);-200*x(1) 101];

scale = 100; xtest = [0.5911; 0.3459]; x0 = [1*scale; 1*scale];
```

0) CasADI

One way: NLP solver

```
import casadi.*

x   = SX.sym('x'); y = SX.sym('y');
nlp = struct('x',[x;y], 'f',0.5*(x-1)^2+0.5*100*(y-x^2)^2+0.5*y^2);
S   = nlpsol('S', 'ipopt', nlp); %disp(S)
r   = S('x0',x0,'lbg',0,'ubg',0);
```

```
This is Ipopt version 3.12.3, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        3

Total number of variables............................:        2
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 4.9005099e+009 0.00e+000 1.00e+002  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 4.7596515e+007 0.00e+000 9.81e-001  -1.0 9.61e+003    -  1.00e+000 1.00e+000f  1
   2 2.6651768e+007 0.00e+000 2.80e+000  -1.7 6.45e+003    -  1.00e+000 5.00e-001f  2
   3 1.3365920e+007 0.00e+000 1.35e+000  -1.7 1.67e+003    -  1.00e+000 1.00e+000f  1
   4 6.9094484e+006 0.00e+000 1.07e+000  -1.7 1.57e+003    -  1.00e+000 1.00e+000f  1
   5 3.4369823e+006 0.00e+000 4.51e-001  -1.7 7.77e+002    -  1.00e+000 1.00e+000f  1
   6 1.8045777e+006 0.00e+000 4.22e-001  -1.7 8.60e+002    -  1.00e+000 1.00e+000f  1
   7 8.8255801e+005 0.00e+000 1.45e-001  -1.7 3.41e+002    -  1.00e+000 1.00e+000f  1
   8 4.8006908e+005 0.00e+000 1.73e-001  -2.5 4.85e+002    -  1.00e+000 1.00e+000f  1
   9 2.2516395e+005 0.00e+000 4.37e-002  -2.5 1.33e+002    -  1.00e+000 1.00e+000f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
```

```
  10 1.3317218e+005 0.00e+000 7.33e-002  -2.5 2.81e+002    -  1.00e+000 1.00e+000f  1
  11 5.6082534e+004 0.00e+000 1.23e-002  -2.5 4.09e+001    -  1.00e+000 1.00e+000f  1
  12 3.9249899e+004 0.00e+000 3.13e-002  -3.8 1.62e+002    -  1.00e+000 1.00e+000f  1
  13 1.3159944e+004 0.00e+000 3.28e-003  -3.8 5.73e+000    -  1.00e+000 1.00e+000f  1
  14 1.1519005e+004 0.00e+000 1.24e-002  -3.8 8.89e+001    -  1.00e+000 1.00e+000f  1
  15 2.8277207e+003 0.00e+000 8.81e-004  -3.8 3.28e+000    -  1.00e+000 1.00e+000f  1
  16 1.6435290e+003 0.00e+000 1.74e-003  -5.7 4.44e+001    -  1.00e+000 5.00e-001f  2
  17 8.3739044e+002 0.00e+000 1.09e-003  -5.7 1.52e+001    -  1.00e+000 1.00e+000f  1
  18 4.2497227e+002 0.00e+000 6.26e-004  -5.7 1.04e+001    -  1.00e+000 1.00e+000f  1
  19 2.1684484e+002 0.00e+000 3.98e-004  -5.7 7.81e+000    -  1.00e+000 1.00e+000f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  20 1.0986773e+002 0.00e+000 2.23e-004  -5.7 5.19e+000    -  1.00e+000 1.00e+000f  1
  21 5.6090081e+001 0.00e+000 1.47e-004  -5.7 4.03e+000    -  1.00e+000 1.00e+000f  1
  22 2.8261150e+001 0.00e+000 7.89e-005  -5.7 2.57e+000    -  1.00e+000 1.00e+000f  1
  23 1.4383309e+001 0.00e+000 5.45e-005  -5.7 2.10e+000    -  1.00e+000 1.00e+000f  1
  24 7.1433281e+000 0.00e+000 2.74e-005  -5.7 1.24e+000    -  1.00e+000 1.00e+000f  1
  25 3.6002344e+000 0.00e+000 2.04e-005  -5.7 1.10e+000    -  1.00e+000 1.00e+000f  1
  26 1.7442644e+000 0.00e+000 9.11e-006  -5.7 5.80e-001    -  1.00e+000 1.00e+000f  1
  27 8.7834222e-001 0.00e+000 7.50e-006  -8.6 5.75e-001    -  1.00e+000 1.00e+000f  1
  28 4.3502740e-001 0.00e+000 2.77e-006  -8.6 2.52e-001    -  1.00e+000 1.00e+000f  1
  29 2.5242602e-001 0.00e+000 2.49e-006  -8.6 2.88e-001    -  1.00e+000 1.00e+000f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  30 1.7200408e-001 0.00e+000 6.66e-007  -8.6 9.77e-002    -  1.00e+000 1.00e+000f  1
  31 1.4906400e-001 0.00e+000 5.06e-007  -8.6 1.12e-001    -  1.00e+000 1.00e+000f  1
  32 1.4433012e-001 0.00e+000 6.66e-008  -8.6 2.76e-002    -  1.00e+000 1.00e+000f  1
  33 1.4403722e-001 0.00e+000 1.02e-008  -8.6 1.40e-002    -  1.00e+000 1.00e+000f  1
  34 1.4403515e-001 0.00e+000 4.78e-011 -11.3 7.70e-004    -  1.00e+000 1.00e+000f  1

Number of Iterations....: 34

                                   (scaled)                 (unscaled)
Objective...............:  7.2744990300548397e-008   1.4403515281262624e-001
Dual infeasibility......:  4.7811303127615638e-011   9.4666427525869068e-005
Constraint violation....:  0.0000000000000000e+000   0.0000000000000000e+000
Complementarity.........:  0.0000000000000000e+000   0.0000000000000000e+000
Overall NLP error.......:  4.7811303127615638e-011   9.4666427525869068e-005


Number of objective function evaluations             = 45
Number of objective gradient evaluations             = 35
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 34
Total CPU secs in IPOPT (w/o function evaluations)   =      0.387
Total CPU secs in NLP function evaluations           =      0.001

EXIT: Optimal Solution Found.
           S  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |   1.00ms ( 22.22us)        0 (       0)        45
  nlp_grad_f  |        0 (       0)        0 (       0)        36
  nlp_hess_l  |        0 (       0)        0 (       0)        34
       total  | 391.00ms (391.00ms) 390.47ms (390.47ms)         1
```

```
x_opt = r.x;
t_casADI_1 = S.stats.t_proc_total;
I_casADI_1 = S.stats.iter_count; %tic(t0);
```

Second Way: Default solver

```
tstart = tic();
```

```
opti  = casadi.Opti();
x_opt = opti.variable(2);

opti.minimize(f(x_opt));

opti.solver('ipopt');
sol = opti.solve();
```

```
This is Ipopt version 3.12.3, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        3

Total number of variables............................:        2
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 5.0000000e-001 0.00e+000 1.00e+000  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 4.7656250e-001 0.00e+000 6.25e+000  -1.7 1.00e+000    -  1.00e+000 2.50e-001f  3
   2 2.4789447e-001 0.00e+000 4.79e-001  -1.7 8.78e-002    -  1.00e+000 1.00e+000f  1
   3 2.0050355e-001 0.00e+000 2.52e+000  -1.7 3.08e-001    -  1.00e+000 5.00e-001f  2
   4 1.5482889e-001 0.00e+000 2.42e-001  -1.7 6.94e-002    -  1.00e+000 1.00e+000f  1
   5 1.4606024e-001 0.00e+000 7.13e-001  -1.7 8.19e-002    -  1.00e+000 1.00e+000f  1
   6 1.4403961e-001 0.00e+000 3.13e-003  -1.7 1.11e-002    -  1.00e+000 1.00e+000f  1
   7 1.4403515e-001 0.00e+000 3.43e-004  -3.8 2.00e-003    -  1.00e+000 1.00e+000f  1
   8 1.4403515e-001 0.00e+000 9.87e-010  -5.7 1.34e-006    -  1.00e+000 1.00e+000f  1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  1.4403515269178327e-001   1.4403515269178327e-001
Dual infeasibility......:  9.8698110795325533e-010   9.8698110795325533e-010
Constraint violation....:  0.0000000000000000e+000   0.0000000000000000e+000
Complementarity.........:  0.0000000000000000e+000   0.0000000000000000e+000
Overall NLP error.......:  9.8698110795325533e-010   9.8698110795325533e-010


Number of objective function evaluations             = 20
Number of objective gradient evaluations             = 9
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 8
Total CPU secs in IPOPT (w/o function evaluations)   =      0.007
Total CPU secs in NLP function evaluations           =      0.000

EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |        0 (       0)        0 (       0)        20
  nlp_grad_f  |        0 (       0)        0 (       0)        10
  nlp_hess_l  |        0 (       0)        0 (       0)         8
       total  |   9.00ms (  9.00ms)   9.00ms (  9.00ms)         1
```

```
xopt = sol.value(x_opt);
t_casADI_2 = sol.stats.t_proc_total;
I_casADI_2 = sol.stats.iter_count;
```

00) Matlab

Using matlab solver to NLP

```
options = optimoptions('fminunc','Display','iter');%'Algorithm','quasi-newton');
t0 = tic();
[x_M, fval_M, exitflag_M, output] = fminunc(f,x0,options)
```

```
                                                First-order
 Iteration  Func-count       f(x)        Step-size       optimality
     0           3      4.90051e+09                      1.98e+08
     1           9      3.19997e+09     5.0505e-08       1.44e+08
     2          12      7.64779e+08              1       4.95e+07
     3          15      2.72601e+08              1        2.3e+07
     4          18      8.21977e+07              1       9.53e+06
     5          21      2.57264e+07              1        4.1e+06
     6          24      7.54562e+06              1       1.72e+06
     7          27      2.07635e+06              1        7.1e+05
     8          30           499877              1       2.82e+05
     9          33          97135.4              1       1.03e+05
    10          36          15743.4              1       3.13e+04
    11          39          5622.89              1       6.43e+03
    12          42          5130.99              1            600
    13          45          5126.56              1            101

Local minimum found.

Optimization completed because the size of the gradient is less than
the value of the optimality tolerance.

<stopping criteria details>
x_M = 2×1
   10.0427
  100.8531
fval_M = 5.1266e+03
exitflag_M = 1
output = struct with fields:
       iterations: 13
        funcCount: 45
         stepsize: 0.0145
     lssteplength: 1
    firstorderopt: 100.5553
        algorithm: 'quasi-newton'
          message: 'Local minimum found.↵↵Optimization completed because the size of the gradient is less than↵the
```

```
t_matlab = toc(t0);
I_matlab = output.iterations;
```

b) Newton Method

Implement your own Newton method using the Hessian and complete steps. Start from the inital value $(x_0, y_0) = (1, 1)$ and use it as a stop condition $||\nabla f(x_k, y_k)||_\infty \geq 10^{-3}$. Keep a track over the iterations (x_k, y_k) and add it to the contour plot.

In order to solve the problem by the Newton step let's remember the Taylor series around t, the descent direction

$$f(x_k + t) \approx f(x_k) + \nabla f(x_k)t + \frac{1}{2}\nabla^2 f(x_k)t^2$$

which has a minimum at

$$\frac{d}{dt} f(x_k + t) = \frac{d}{dt}\left(f(x_k) + \nabla f(x_k)t + \frac{1}{2}\nabla^2 f(x_k)t^2 \right) = 0$$
$$\nabla f(x_k) + t = 0$$
$$t = -\nabla^2 f(x_k)^{-1}\nabla f(x_k) = -\nabla^2 f_k^{-1}\nabla f_k$$

then, the approximation is

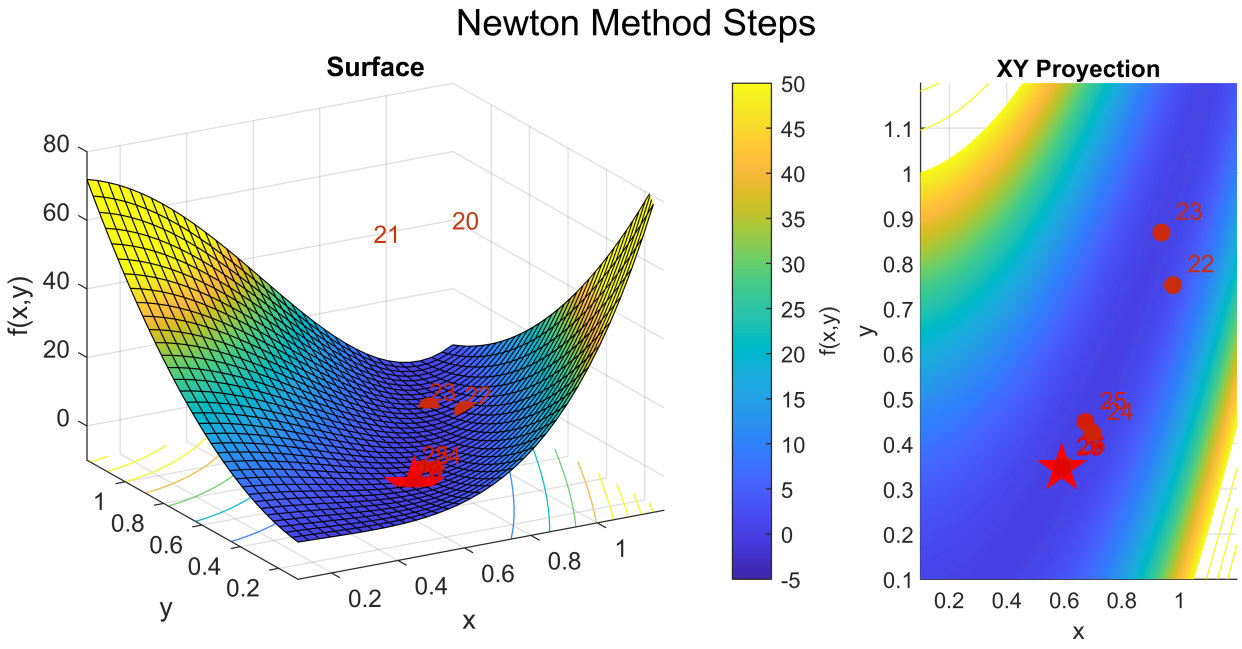
$$x_{k+1} = x_k - \nabla^2 f_k^{-1}\nabla f_k$$

which is the Newton iteration that is programming in the **`Iteration`** function.

```
Newton = @(xk) -Hessian(xk)\Gradient(xk);
[xk_Newton, c_Newton, time_Newton] = Iteration(x0, Newton, Gradient)
```

```
xk_Newton = 2×31
10^3 ×
    0.1000    0.0990    0.0661    0.0637    0.0434    0.0417    0.0286    0.0274 ···
    0.1000    9.7068    3.2568    4.0166    1.4598    1.7175    0.6369    0.7405
c_Newton = 30
time_Newton = 0.0070
```

```
FinalPlot(fp1, x1_opt, x0, xk_Newton(:,2:end-1), 1, [-30 24], "Newton Method Steps", 1)
```



c) Gauss- Newton Method

Update by adding a Newton type method that uses the Hessian of the Gauss-Newton method instead. Add the resultant iterations (x_k, y_k) to the plot.

In this approximation the updated is made by approximating the Hessian with the Jociban of the residual vector

$$H \approx J^T J = H_R$$

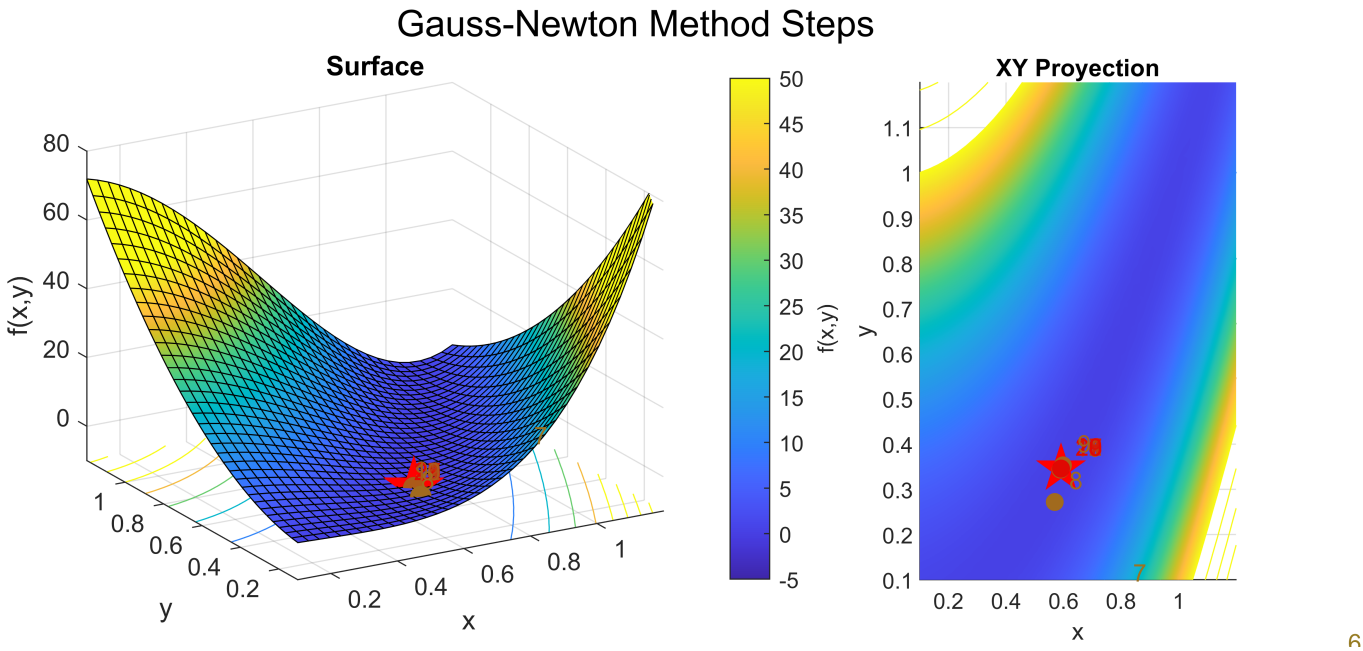
then the iterations is

$$x_{k+1} = x_k - H_R^{-1} \nabla f_k$$

```
%GaussNewtonApprox  = @(xk) -(Gradient(xk)'*Gradient(xk))\ (Gradient(xk));%'*f(xk));
%[xk_GN, c_GN] = Iteration(x0, GaussNewtonR, Gradient)
GaussNewton = @(xk) -HessianR(xk)\Gradient(xk);
[xk_GN, c_GN, time_GN] = Iteration(x0, GaussNewton, Gradient)
```

```
xk_GN = 2×22
  100.0000   49.9988   24.9970   12.4938    6.2384    3.1056    1.5387    0.7916 ···
  100.0000   -0.2450   -0.2400   -0.2299   -0.2096   -0.1688   -0.0867    0.0677
c_GN = 21
time_GN = 0.0029
```

```
FinalPlot(fp1, x1_opt, x0, xk_GN(:,2:end-1), 1, [-30 24], "Gauss-Newton Method Steps", 1);
```



d) Add Regularization

Check how the steepest descent method works in this example. Now the Hessian approximation is αI, where α is a positive scalar and I is the identity matrix of appropiate size. What values does the algorithm converge for?

In this case, a regularization term is added to the Hessian,

$$H_{reg} \approx H + \alpha I$$

then the iteration is

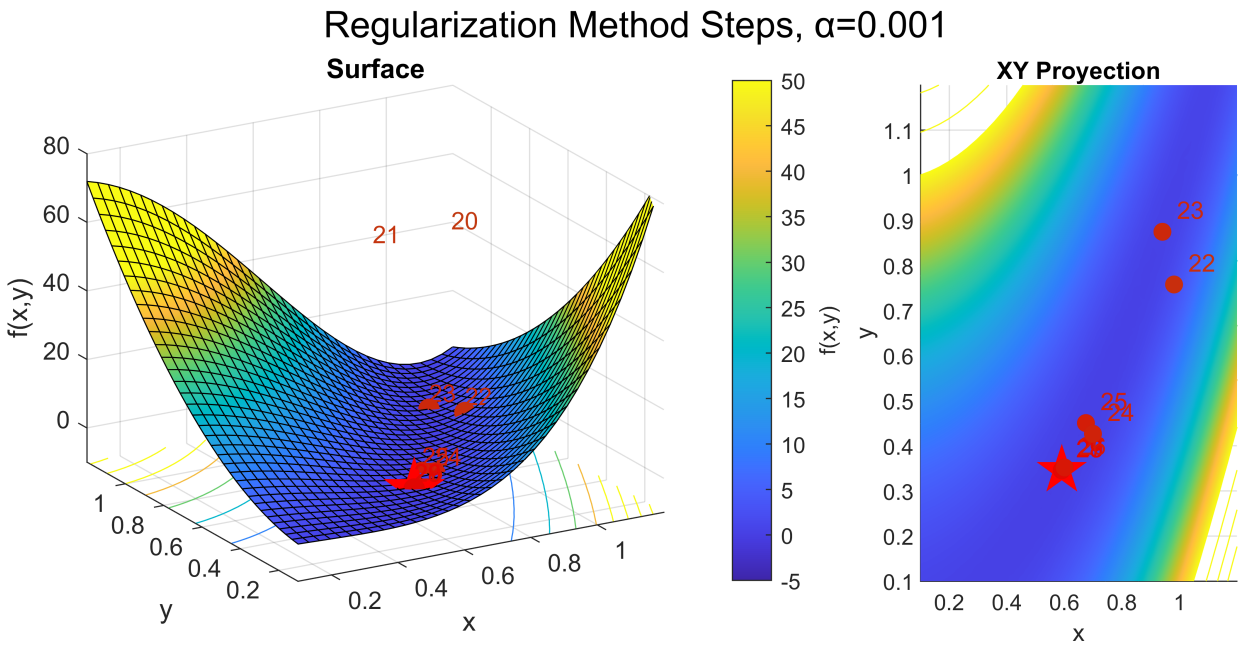
$$x_{k+1} = x_k - H_{reg}^{-1}\nabla f_k = x_k - (H + \alpha I)^{-1}\nabla f_k$$

- $\alpha = 1$

```
HessianApprox  = @(xk) -(Hessian(xk)+0.001*eye(size(xk,1)))\Gradient(xk);
%HessianApproxR = @(xk) -(HessianR(xk)+alpha*eye(size(xk,1)))\Gradient(xk);
[xk_reg0, c_reg0, time_reg0] = Iteration(x0, HessianApprox, Gradient)
```

```
xk_reg0 = 2×31
10^3 ×
    0.1000    0.0990    0.0662    0.0638    0.0435    0.0417    0.0286    0.0274 ···
    0.1000    9.7065    3.2632    4.0196    1.4634    1.7200    0.6388    0.7421
c_reg0 = 30
time_reg0 = 0.0023
```

```
FinalPlot(fp1, x1_opt, x0, xk_reg0(:,2:end-1), 1, [-30 24], "Regularization Method Steps, α=0.0
```

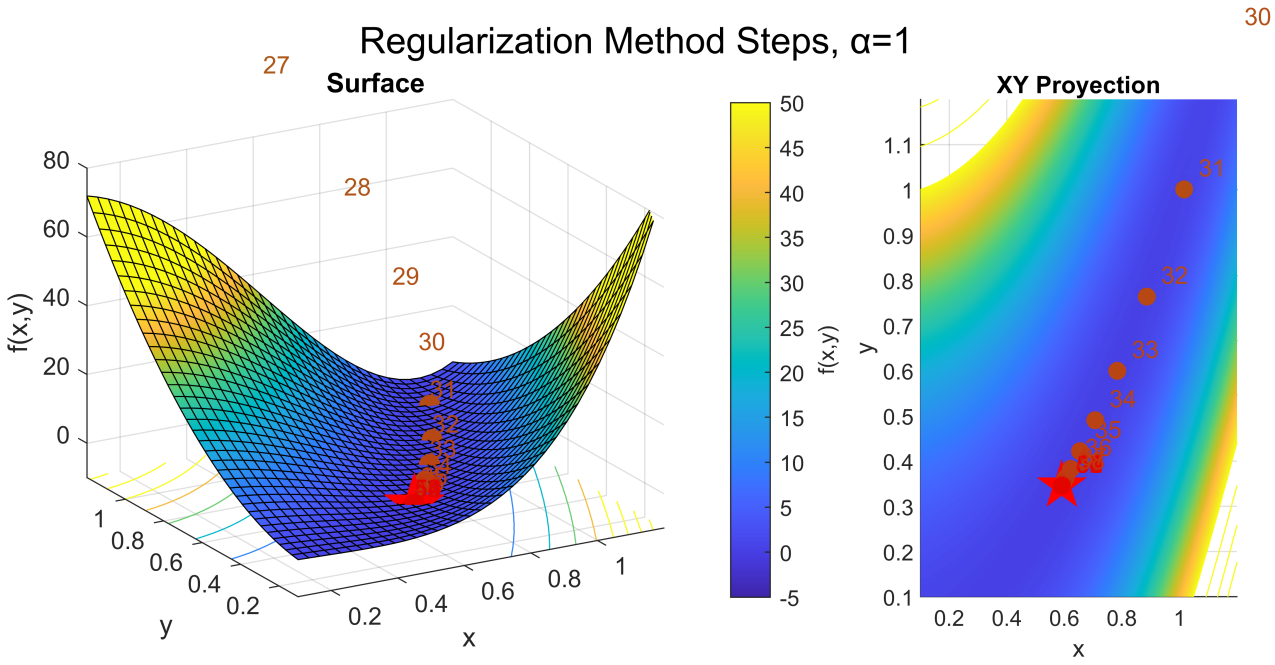


- $\alpha = 1$

```
alpha = 1; HessianApprox  = @(xk) -(Hessian(xk)+alpha*eye(size(xk,1)))\Gradient(xk);
%HessianApproxR = @(xk) -(HessianR(xk)+alpha*eye(size(xk,1)))\Gradient(xk);
[xk_reg1, c_reg1, time_reg1] = Iteration(x0, HessianApprox, Gradient)
```

```
xk_reg1 = 2×56
10^3 ×
    0.1000    0.0981    0.0815    0.0717    0.0606    0.0527    0.0449    0.0389 ···
    0.1000    9.4332    6.3347    5.0132    3.5314    2.7009    1.9476    1.4646
c_reg1 = 55
time_reg1 = 0.0024
```

```
FinalPlot(fp1, x1_opt, x0, xk_reg1(:,2:end-1), 1, [-30 24], "Regularization Method Steps, α=1",
```



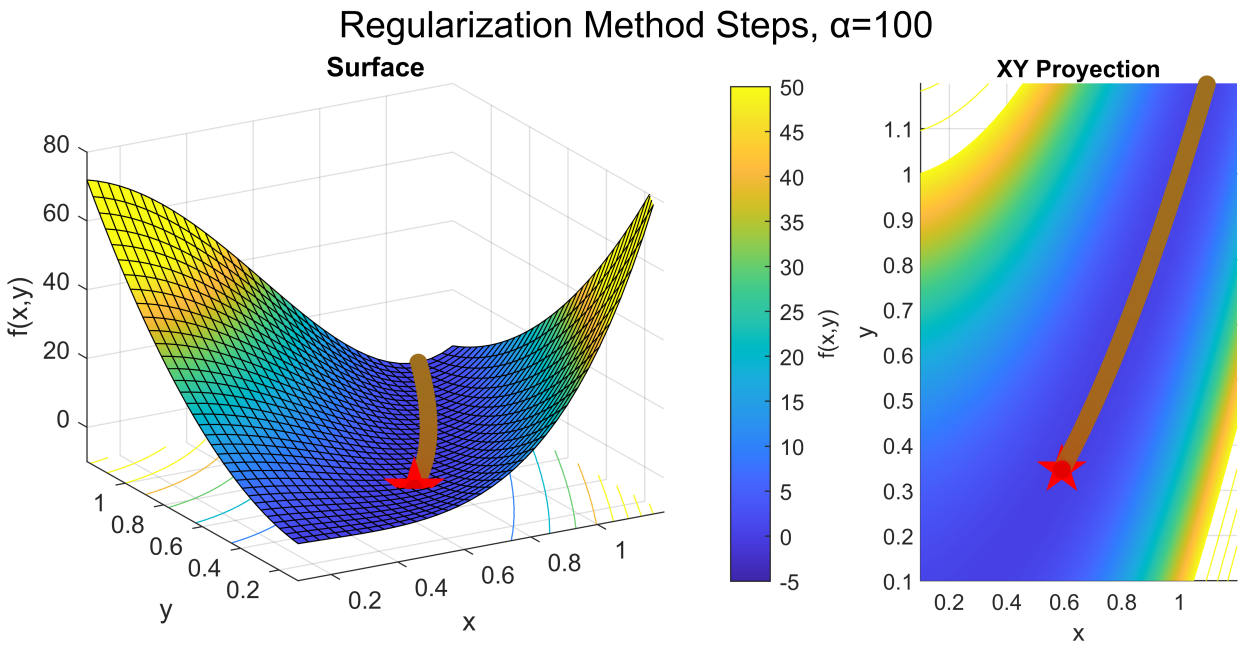

```
%[xk_reg_R, c_reg_R, time_reg_R] = Iteration(x0, HessianApproxR, Gradient)
%xk_reg_R(:,end)'
%FinalPlot(fp1, x1_opt, x0, xk_reg_R(:,2:end-1), 1, [-30 24], "Regularization Method Steps wit
```

- $\alpha = 100$

```
HessianApprox  = @(xk) -(Hessian(xk)+100*eye(size(xk,1)))\Gradient(xk);
[xk_reg2, c_reg2, time_reg2] = Iteration(x0, HessianApprox, Gradient)
```

```
xk_reg2 = 2×2249
10^3 ×
    0.1000    0.0751    0.0616    0.0564    0.0554    0.0551    0.0548    0.0546 ···
    0.1000    2.5435    3.0658    3.0967    3.0676    3.0373    3.0072    2.9774
c_reg2 = 2248
time_reg2 = 0.0274
```

```
FinalPlot(fp1, x1_opt, x0, xk_reg2(:,2:end-1), 1, [-30 24], "Regularization Method Steps, α=100
```

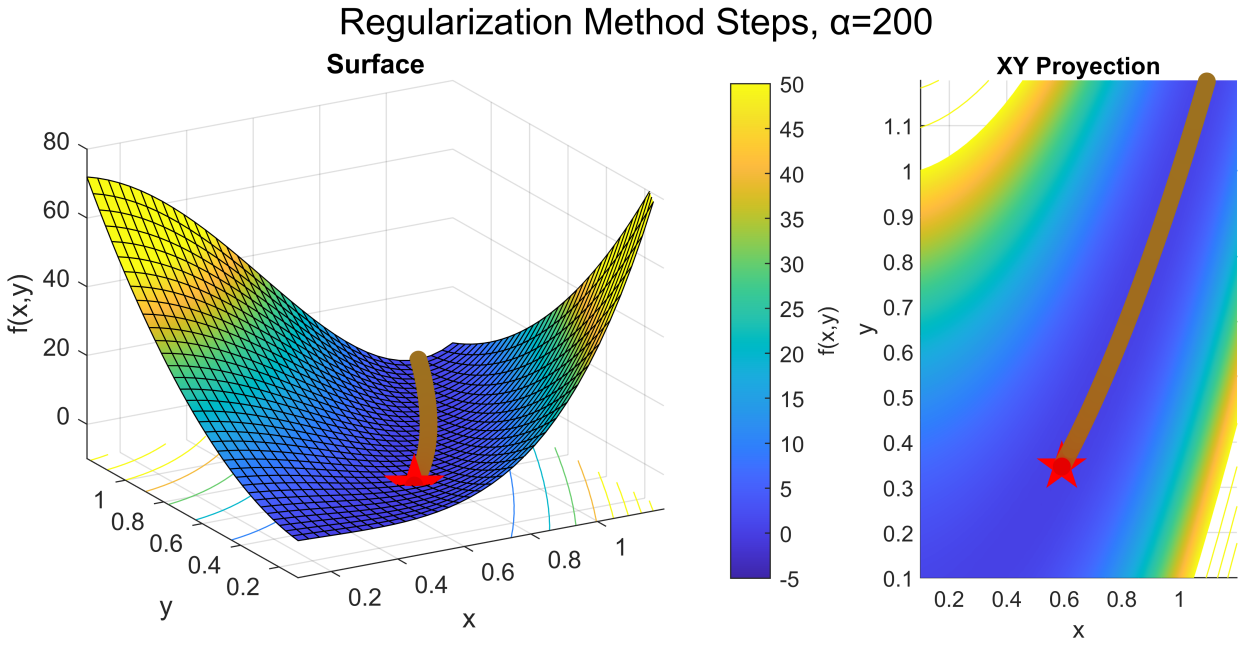


- $\alpha = 200$

```
HessianApprox  = @(xk) -(Hessian(xk)+200*eye(size(xk,1)))\Gradient(xk);
[xk_reg3, c_reg3, time_reg3] = Iteration(x0, HessianApprox, Gradient)
```

```
xk_reg3 = 2×4391
10^3 ×
    0.1000    0.0716    0.0550    0.0470    0.0447    0.0444    0.0442    0.0441 ···
    0.1000    1.4998    1.9100    1.9813    1.9770    1.9672    1.9574    1.9477
c_reg3 = 4390
time_reg3 = 0.0519
```

```
FinalPlot(fp1, x1_opt, x0, xk_reg3(:,2:end-1), 1, [-30 24], "Regularization Method Steps, α=200
```

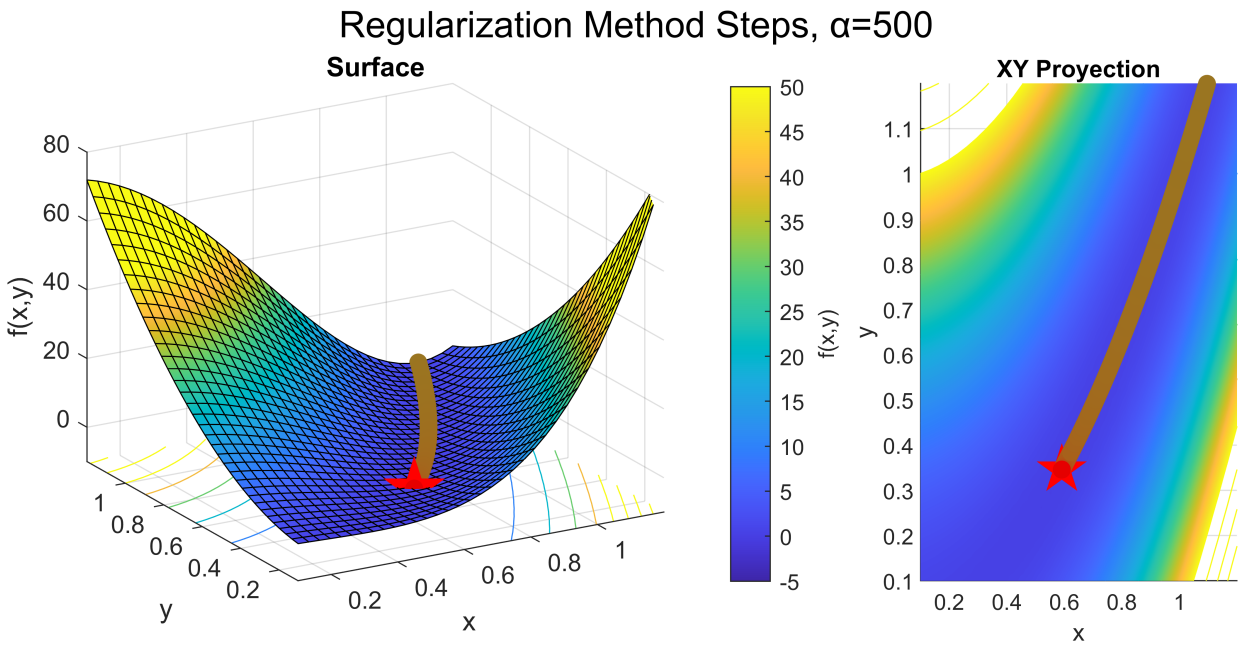


- $\alpha = 500$

```
HessianApprox  = @(xk) -(Hessian(xk)+500*eye(size(xk,1)))\Gradient(xk);
[xk_reg4, c_reg4, time_reg4] = Iteration(x0, HessianApprox, Gradient)
```

```
xk_reg4 = 2×10618
10^3 ×
    0.1000    0.0689    0.0495    0.0384    0.0332    0.0319    0.0318    0.0318 ···
    0.1000    0.7136    0.9393    1.0057    1.0162    1.0149    1.0129    1.0108
c_reg4 = 10617
time_reg4 = 0.2247
```

```
FinalPlot(fp1, x1_opt, x0, xk_reg4(:,2:end-1), 1, [-30 24], "Regularization Method Steps, α=500
```



e) Methods Performance

Compare the performance of the implemented methods. Consider the iteration path (x_k, y_k), the number of iterations and the execution time. To measure the time, you can use the command **`tic`** and **`toc`** from Matlab.

```
Method  = ["Newton"; "Gauss-Newton"; "Regularization, α=1e-3";"Regularization, α=1"; "Regulariz
          "Regularization, α=200"; "Regularization, α=500"; "CasADI 1"; "CasADI NLP"; "Matlab"
Time    = [time_Newton; time_GN; time_reg0; time_reg1; time_reg2; time_reg3; time_reg4; t_casAD
No_Iter = [c_Newton; c_GN; c_reg0; c_reg1; c_reg2; c_reg3; c_reg4; I_casADI_1; I_casADI_2; I_ma
tabla   = sortrows(table(Method, Time, No_Iter),2)
```

`tabla = 10×3 table`

	Method	Time	No_Iter
1	"Regularization, α=1e-3"	0.0023	30
2	"Regularization, α=1"	0.0024	55
3	"Gauss-Newton"	0.0029	21
4	"Newton"	0.0070	30
5	"CasADI NLP"	0.0090	8
6	"Regularization, α=100"	0.0274	2248
7	"Regularization, α=200"	0.0519	4390
8	"Matlab"	0.0706	13
9	"Regularization, α=500"	0.2247	10617
10	"CasADI 1"	0.3910	34

When the start point is $(1,1)^T$ and the tolerance is 10^{-8}, difference between solutions $|x_{k+1} - x_k| < \text{tol}$.

	Method	Time	No_Iter
1	"Regularization, α=1e-3"	0.0031	5
2	"Regularization, α=1"	0.0031	24
3	"Gauss-Newton"	0.0032	14
4	"Newton"	0.0062	5
5	"CasADI NLP"	0.0080	8
6	"Regularization, α=100"	0.0151	1429
7	"Regularization, α=200"	0.0212	2849
8	"Regularization, α=500"	0.0593	7107
9	"Matlab"	0.1634	12
10	"CasADI 1"	1.0140	8

We find that the fatest method is when the regularization term is added, but this may depend on the execution so statical brenchnmarks are suggested, although it has an execution time similar to the same mehotd but with bigger regularization term 1. In the other hand, the method with less steps is the CasADI NLP which gives 8 iterations in both cases, when the start point is $(1, 1)$ and $(100, 100)$, after this is the Matlab method and then the Gauss Newton method although when the start point is close to the solution the iterations less for the majority of methods and then the best methods are the Newton and regularization method with $\alpha = 1e - 2$.

2. Methods Implementation

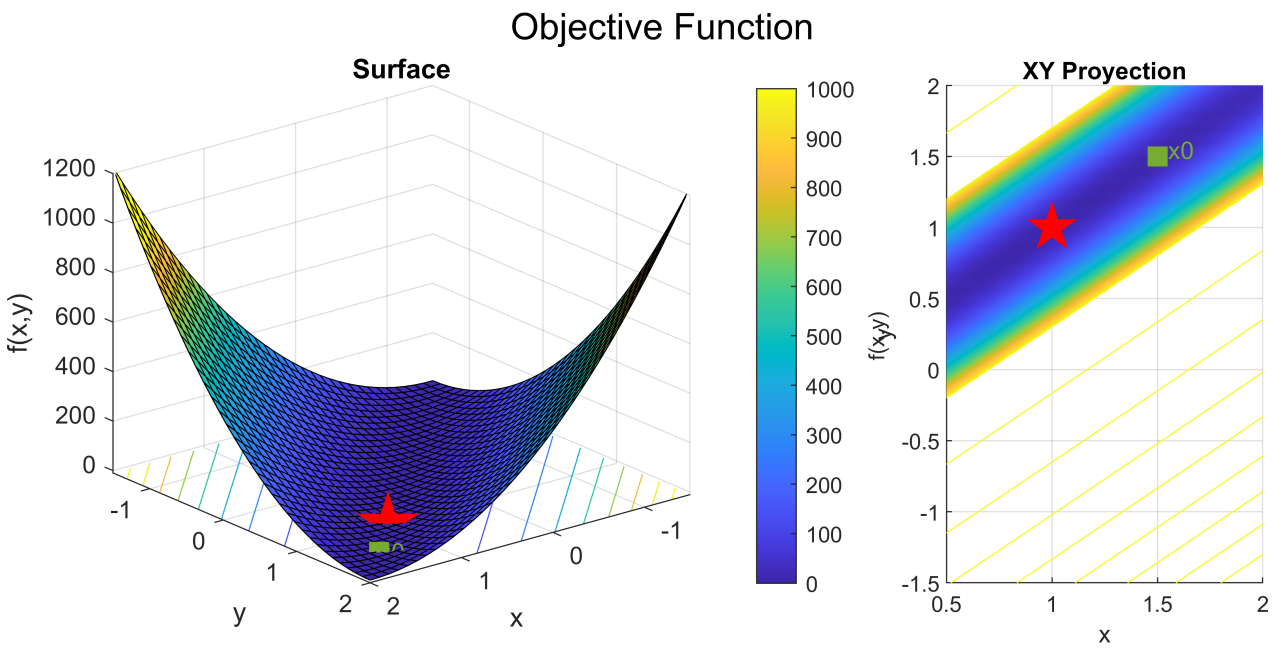
Since we want to implement some linear search algorithms, the following formulas will be usefull

Armijo Rule: $f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k,$

Iteration formula: $x_{k+1} = x_k + \alpha_k p_k$

```
fp2 = @(x,y) 100*(y-x)^2 + (1-x)^2; x2_opt = [1; 1];
FinalPlot(fp2, x2_opt, [1.5;1.5], [], 0, [-218.8 25.2], "Objective Function", 1); %(f, xstar, x
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```



0) Comparison

Using each of the previous methods solve

$$\min_{x_1,x_2\in\mathbb{R}} \underbrace{100(x_2-x_1)^2+(1-x_1)^2}_{=f(x_1,x_2)}$$

$$f(x_1,x_2) = ||R(x_1,x_2)||$$
$$R(x_1,x_2) = \big(10(x_2-x_1), 1-x_1\big)$$

Take as initial value $(1.5, 1.5)^T$, the optimum value is $(1, 1)$. The algorithm output have to show a table with the following information

$$k \qquad x_k \qquad f(x_k) \qquad \nabla f(x_k) \qquad p_k \qquad t^*$$

Compare the methods performance.

Gradient and Hessian

$$\nabla f = \big(-(2)100(x_2-x_1) - 2(1-x_1), (2)100(x_2-x_1)\big)$$
$$\nabla f = \big(200(x_1-x_2) - 2 + 2x_1,\ 200(x_2-x_1)\big)$$
$$\nabla^2 f(x,y) = \begin{pmatrix} 200+2 & -200 \\ -200 & 200 \end{pmatrix} = \begin{pmatrix} 202 & -200 \\ -200 & 200 \end{pmatrix}$$

Rules to α

```
global c1 c2 alpha0 rho
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```

```
global Armijo Wolfe StrongWolfe % Goldstein

f        = @(x) 100*(x(2)-x(1))^2 + (1-x(1))^2;
Gradient = @(x) [200*(x(1)-x(2))-2+2*x(1); 200*(x(2)-x(1))]; % G(x) = H(x,y)
Hessian  = @(x) [202 -200; -200 200];        % H(x) = H(x,y)

Armijo      = @(xk, f, Gradient, pk, alphak) f(xk+alphak*pk) <= f(xk)+c1*alphak*Gradient(xk)'*p
Wolfe       = @(xk, f, Gradient, pk, alphak) (Gradient(xk+alphak*pk)'*pk >= c2 * Gradient(xk)'*
StrongWolfe = @(xk, f, Gradient, pk, alphak) (norm(Gradient(xk+alphak*pk)'*pk,2) >= c2 * norm(G
%Goldstein   = @(xk, f, Gradient, pk, alphak) f(xk)+(1-c)*alphak*Gradient(xk)'*pk <= f(xk)+c*al

c1 = 1e-4; c2 = 0.9; alpha0 = 1; rho=1/2; %c1 = 1e-4 sa
x0 = [scale*1.5; scale*1.5];
```

00) Solution by Matlab

This problem can be write as a quadratic problem as follows

$$R = \begin{pmatrix} -10 & 10 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = Ax + b$$

$$R^T = x^T A^T + b^T$$

$$f = ||R|| = R^T R = (x^T A^T + b^T)(Ax + b) = x^T A^T Ax + 2b^T Ax + b^T b$$

$$f = \frac{1}{2} x^T Hx + g^T x$$

$$H = 2A^T A, \quad g^T = 2b^T A, \quad b^T b = 1$$

then the problem can be formulated as, omiting the constant part

$$\min_{x \in \mathbb{R}^2} \quad \frac{1}{2} x^T Hx + g^T x$$

where $H = 2A^T A$ and $g^T = 2b^T A$.

```
options = optimoptions('quadprog','Display','iter');
A = [-10 10; -1 0]; b = [0; 1];
H = 2*(A')*A;  g = 2*A'*b;
tStart = tic();
[x,fval,exitflag,output,lambda] = quadprog(H,g,[],[],[],[],[],[],[150; 30],options)
```

```
The interior-point-convex algorithm does not accept an initial point.
Ignoring X0.

 Iter           Fval  Primal Infeas    Dual Infeas  Complementarity
    0   -1.000000e+00   0.000000e+00   9.999999e-08     0.000000e+00
    1   -1.000000e+00   0.000000e+00   5.002221e-11     0.000000e+00

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
```

```
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
x = 2×1
    1.0000
    1.0000
fval = -1.0000
exitflag = 1
output = struct with fields:
            message: 'Minimum found that satisfies the constraints.↵↵Optimization completed because the objective
          algorithm: 'interior-point-convex'
      firstorderopt: 5.0022e-11
    constrviolation: 0
         iterations: 1
       linearsolver: 'dense'
       cgiterations: []
lambda = struct with fields:
    ineqlin: [0×1 double]
      eqlin: [0×1 double]
      lower: [2×1 double]
      upper: [2×1 double]
```

```
tstar_Matlab = toc(tStart);
```

000) CasADI - ipopt solver

```
tstart = tic();
opti  = casadi.Opti();
x_opt = opti.variable(2);

opti.minimize(0.5*x_opt'*H*x_opt+g'*x_opt);

opti.solver('ipopt');
sol = opti.solve();
```

```
This is Ipopt version 3.12.3, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        3

Total number of variables............................:        2
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 0.0000000e+000 0.00e+000 2.00e+000  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 -1.0000000e+000 0.00e+000 0.00e+000  -1.0 1.00e+000    -  1.00e+000 1.00e+000f  1

Number of Iterations....: 1

                                   (scaled)                 (unscaled)
Objective...............: -1.0000000000000098e+000  -1.0000000000000098e+000
```

```
Dual infeasibility......:  0.0000000000000000e+000   0.0000000000000000e+000
Constraint violation....:  0.0000000000000000e+000   0.0000000000000000e+000
Complementarity.........:  0.0000000000000000e+000   0.0000000000000000e+000
Overall NLP error.......:  0.0000000000000000e+000   0.0000000000000000e+000


Number of objective function evaluations             = 2
Number of objective gradient evaluations             = 2
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 1
Total CPU secs in IPOPT (w/o function evaluations)   =      4.870
Total CPU secs in NLP function evaluations           =      0.000

EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |        0 (       0)        0 (       0)         2
  nlp_grad_f  |        0 (       0)        0 (       0)         3
  nlp_hess_l  |        0 (       0)        0 (       0)         1
       total  |   4.88 s (  4.88 s)   4.88 s (  4.88 s)         1
```

```
xopt = sol.value(x_opt);
tstart_casADI = toc(tstart);
```

000) CasADI II - Quadratic Problem solver

```
import casadi.*
tstart = tic();

x = SX.sym('x'); y = SX.sym('y');
qp = struct('x',[x;y], 'f',100*(y-x)^2+(1-x)^2);
S = qpsol('S', 'qpoases', qp);

r = S('lbg',0);
```

```
WARNING:   Stepsize is 0.000000000000000e+000


#################   qpOASES  --  QP NO.   1   ##################

    Iter   |    StepLength    |       Info       |   nFX
 ----------+-------------------+------------------+---------
       0   |   0.000000e+000   |   REM BND    1   |     1
       1   |   1.666665e-007   |   REM BND    0   |     0
       2   |   1.000000e+000   |    QP SOLVED     |     0
```

```
x_opt = r.x
```

```
x_opt =
[1, 1]
```

```
tstart_casADI2 = toc(tstart);
```

a) Steepest Descent

Such that employs the backward globalization technique with Armijo rule.

In this case the search direction is the gradient of the function at the corresponding point

$$p_k = \nabla f_k$$

```
pk_func_SD = @(xk) -Gradient(xk);
[ks_SD, xks_SD, fks_SD, grad_SD, pks_SD, tstar_SD, I_SD] = MethodImpl(pk_func_SD, x0, Gradient,
Tabla(xks_SD, ks_SD, grad_SD, pks_SD, fks_SD)
```

ans = 1916×8 table

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
1	1	150	150	22201	298	0	-298	0
2	2	147.6719	150	2.2055e+04	-172.2812	465.6250	172.2812	-465.6250
3	3	148.0084	149.0906	2.1729e+04	77.5739	216.4429	-77.5739	-216.4429
4	4	146.7963	145.7087	2.1375e+04	509.1154	-217.5228	-509.1154	217.5228
5	5	145.8019	146.1335	2.0979e+04	223.2836	66.3202	-223.2836	-66.3202
6	6	142.3131	145.0973	2.0745e+04	-274.2045	556.8307	274.2045	-556.8307
7	7	142.8487	144.0097	2.0256e+04	51.4897	232.2076	-51.4897	-232.2076
8	8	142.4464	142.1956	2.0013e+04	333.0569	-50.1641	-333.0569	50.1641
9	9	141.1454	142.3915	1.9796e+04	31.0635	249.2273	-31.0635	-249.2273
10	10	141.0240	141.4180	1.9622e+04	201.2613	78.7868	-201.2613	-78.7868
11	11	134.7346	138.9559	1.9667e+04	-576.7828	844.2521	576.7828	-844.2521
12	12	135.8612	137.3070	1.8397e+04	-19.4380	289.1603	19.4380	-289.1603
13	13	135.9371	136.1774	1.8214e+04	221.8063	48.0679	-221.8063	-48.0679
14	14	132.4714	135.4264	1.8158e+04	-328.0577	591.0004	328.0577	-591.0004
15	15	133.1121	134.2721	1.7588e+04	32.2309	231.9933	-32.2309	-231.9933
16	16	132.9862	133.3658	1.7435e+04	188.0435	75.9289	-188.0435	-75.9289
17	17	127.1098	130.9931	1.7412e+04	-524.4255	776.6451	524.4255	-776.6451
18	18	128.1341	129.4762	1.6343e+04	-14.1462	268.4144	14.1462	-268.4144
19	19	128.1894	128.4277	1.6183e+04	206.7148	47.6639	-206.7148	-47.6639
20	20	124.9595	127.6829	1.6108e+04	-296.7790	544.6979	296.7790	-544.6979
21	21	125.5391	126.6191	1.5627e+04	33.0822	215.9960	-33.0822	-215.9960
22	22	125.4099	125.7753	1.5491e+04	175.7251	73.0946	-175.7251	-73.0946
23	23	119.9185	123.4911	1.5418e+04	-476.6987	714.5356	476.6987	-714.5356
24	24	120.8495	122.0956	1.4519e+04	-9.5107	249.2097	9.5107	-249.2097
25	25	120.8867	121.1221	1.4378e+04	192.6889	47.0844	-192.6889	-47.0844
26	26	117.8759	120.3864	1.4290e+04	-268.3467	502.0985	268.3467	-502.0985
27	27	118.4000	119.4057	1.3884e+04	33.6567	201.1433	-33.6567	-201.1433
28	28	118.2685	118.6200	1.3764e+04	164.2427	70.2944	-164.2427	-70.2944

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
29	29	113.1360	116.4233	1.3655e+04	-433.1994	657.4713	433.1994	-657.4713
30	30	113.9820	115.1392	1.2899e+04	-5.4640	231.4281	5.4640	-231.4281
31	31	114.0034	114.2352	1.2775e+04	179.6506	46.3562	-179.6506	-46.3562
32	32	111.1964	113.5109	1.2679e+04	-242.5086	462.9013	242.5086	-462.9013
33	33	111.6700	112.6068	1.2336e+04	33.9894	187.3506	-33.9894	-187.3506
34	34	111.4045	111.1431	1.2196e+04	273.0851	-52.2762	-273.0851	52.2762
35	35	110.3377	111.3473	1.2057e+04	16.7631	201.9123	-16.7631	-201.9123
36	36	110.2722	110.5586	1.1949e+04	161.2800	57.2645	-161.2800	-57.2645
37	37	105.2322	108.7690	1.2115e+04	-498.8967	707.3612	498.8967	-707.3612
38	38	106.2066	107.3875	1.1208e+04	-25.7534	236.1667	25.7534	-236.1667
39	39	106.3072	106.4650	1.1092e+04	179.0729	31.5416	-179.0729	-31.5416
40	40	103.5092	105.9721	1.1115e+04	-287.5583	492.5768	287.5583	-492.5768
41	41	104.0709	105.0101	1.0712e+04	18.3052	187.8365	-18.3052	-187.8365
42	42	103.9994	104.2763	1.0617e+04	150.6085	55.3902	-150.6085	-55.3902
43	43	99.2928	102.5454	1.0719e+04	-453.9191	650.5048	453.9191	-650.5048
44	44	100.1794	101.2749	9.9566e+03	-20.7304	219.0892	20.7304	-219.0892
45	45	100.2604	100.4190	9.8551e+03	166.7906	31.7301	-166.7906	-31.7301
46	46	97.6543	99.9233	9.8569e+03	-260.4856	453.7942	260.4856	-453.7942
47	47	98.1630	99.0369	9.5170e+03	19.5475	174.7786	-19.5475	-174.7786
48	48	98.0867	98.3542	9.4330e+03	140.6691	53.5043	-140.6691	-53.5043
49	49	93.6908	96.6822	9.4864e+03	-412.9029	598.2844	412.9029	-598.2844
50	50	94.4972	95.5137	8.8450e+03	-16.2949	203.2894	16.2949	-203.2894
51	51	94.5609	94.7196	8.7562e+03	155.3826	31.7391	-155.3826	-31.7391
52	52	92.1330	94.2237	8.7423e+03	-235.8590	418.1250	235.8590	-418.1250
53	53	92.5937	93.4070	8.4556e+03	20.5249	162.6625	-20.5249	-162.6625
54	54	92.5135	92.7716	8.3814e+03	131.4096	51.6175	-131.4096	-51.6175
55	55	88.4070	91.1586	8.3971e+03	-375.5041	550.3180	375.5041	-550.3180
56	56	89.1404	90.0837	7.8577e+03	-12.3880	188.6688	12.3880	-188.6688
57	57	89.1888	89.3467	7.7798e+03	144.7844	31.5932	-144.7844	-31.5932
58	58	86.9265	88.8531	7.7545e+03	-213.4627	385.3157	213.4627	-385.3157
59	59	87.3434	88.1005	7.5125e+03	21.2690	151.4179	-21.2690	-151.4179
60	60	87.2603	87.5090	7.4470e+03	122.7817	49.7390	-122.7817	-49.7390
61	61	83.4234	85.9547	7.4344e+03	-341.4086	506.2554	341.4086	-506.2554

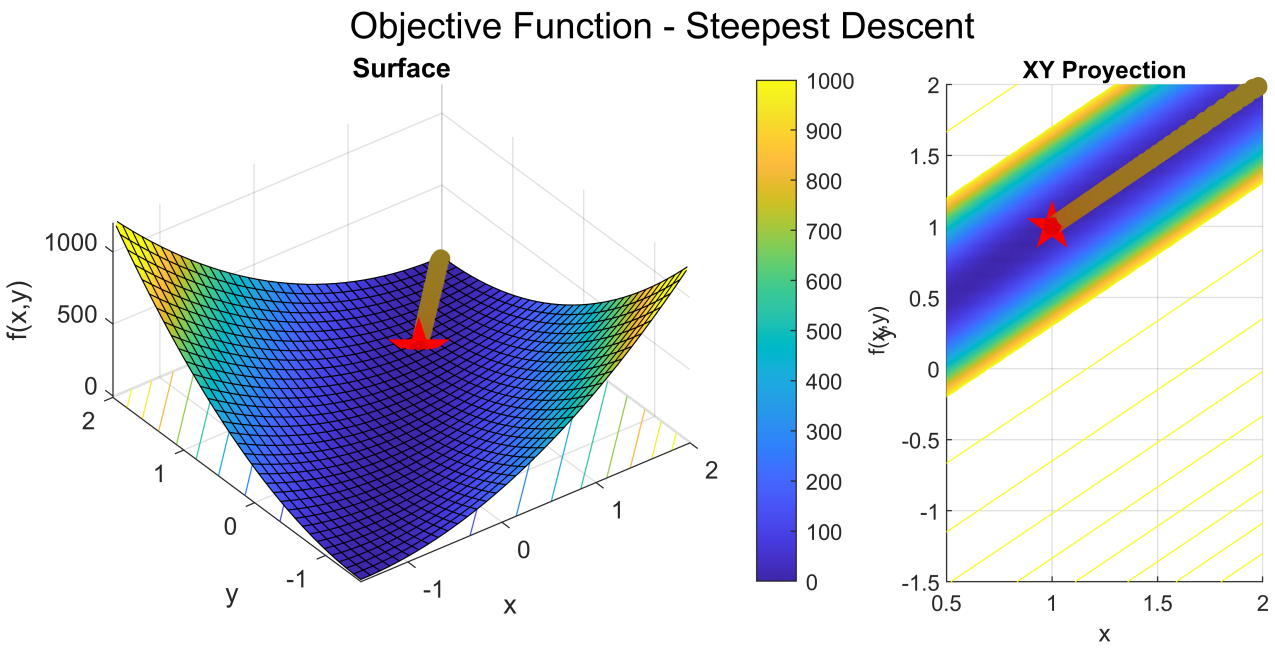
	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
62	62	84.0902	84.9659	6.9807e+03	-8.9562	175.1367	8.9562	-175.1367
63	63	84.1252	84.2818	6.9123e+03	134.9363	31.3141	-134.9363	-31.3141
64	64	82.0168	83.7925	6.8790e+03	-193.0998	355.1335	193.0998	-355.1335
65	65	82.3940	83.0989	6.6747e+03	21.8081	140.9799	-21.8081	-140.9799
66	66	82.3088	82.5482	6.6169e+03	114.7407	47.8769	-114.7407	-47.8769
67	67	78.7232	81.0520	6.5833e+03	-310.3289	465.7753	310.3289	-465.7753
68	68	79.3293	80.1423	6.2016e+03	-5.9510	162.6095	5.9510	-162.6095
69	69	79.3525	79.5071	6.1415e+03	125.7834	30.9216	-125.7834	-30.9216
70	70	77.3871	79.0240	6.1029e+03	-174.5905	327.3648	174.5905	-327.3648
71	71	77.7281	78.3846	5.9303e+03	22.1678	131.2885	-22.1678	-131.2885
72	72	77.6415	77.8717	5.8792e+03	107.2452	46.0379	-107.2452	-46.0379
73	73	74.2901	76.4331	5.8307e+03	-282.0028	428.5831	282.0028	-428.5831
74	74	74.8409	75.5960	5.5095e+03	-3.3286	151.0105	3.3286	-151.0105
75	75	74.8539	75.0061	5.4567e+03	117.2748	30.4331	-117.2748	-30.4331
76	76	73.0215	74.5306	5.4148e+03	-157.7705	301.8135	157.7705	-301.8135
77	77	73.3297	73.9411	5.2690e+03	22.3708	122.2885	-22.3708	-122.2885
78	78	73.1549	72.9857	5.2092e+03	178.1427	-33.8329	-178.1427	33.8329
79	79	72.4590	73.1179	5.1498e+03	11.1450	131.7730	-11.1450	-131.7730
80	80	72.4155	72.6031	5.1037e+03	105.2986	37.5324	-105.2986	-37.5324
81	81	69.1249	71.4303	5.1725e+03	-324.8212	461.0710	324.8212	-461.0710
82	82	69.7593	70.5297	4.7872e+03	-16.5632	154.0819	16.5632	-154.0819
83	83	69.8240	69.9278	4.7378e+03	116.8826	20.7654	-116.8826	-20.7654
84	84	67.9977	69.6034	4.7465e+03	-187.1364	321.1318	187.1364	-321.1318
85	85	68.3632	68.9762	4.5754e+03	12.1369	122.5896	-12.1369	-122.5896
86	86	68.3158	68.4973	4.5347e+03	98.3332	36.2984	-98.3332	-36.2984
87	87	65.2429	67.3630	4.5766e+03	-295.5301	424.0159	295.5301	-424.0159
88	88	65.8201	66.5348	4.2527e+03	-13.3030	142.9433	13.3030	-142.9433
89	89	65.8721	65.9765	4.2095e+03	108.8683	20.8758	-108.8683	-20.8758
90	90	64.1710	65.6503	4.2094e+03	-169.5103	295.8523	169.5103	-295.8523
91	91	64.5021	65.0724	4.0650e+03	12.9341	114.0700	-12.9341	-114.0700
92	92	64.4516	64.6268	4.0292e+03	91.8455	35.0576	-91.8455	-35.0576
93	93	61.5814	63.5313	4.0503e+03	-268.8193	389.9820	268.8193	-389.9820
94	94	62.1064	62.7696	3.7780e+03	-10.4249	132.6378	10.4249	-132.6378

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
95	95	62.1471	62.2515	3.7401e+03	101.4243	20.8700	-101.4243	-20.8700
96	96	60.5624	61.9254	3.7335e+03	-153.4772	272.6020	153.4772	-272.6020
97	97	60.8621	61.3930	3.6117e+03	13.5595	106.1648	-13.5595	-106.1648
98	98	60.8092	60.9783	3.5800e+03	85.8015	33.8169	-85.8015	-33.8169
99	99	58.1279	59.9215	3.5853e+03	-244.4645	358.7203	244.4645	-358.7203
100	100	58.6054	59.2209	3.3563e+03	-7.8905	123.1012	7.8905	-123.1012

⋮

```
FinalPlot(fp2, x2_opt, x0, xks_SD(:,2:end-1), 0, [-37 53], "Objective Function - Steepest Desce
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```



b) Newton

Such that employs the backward globalization technique with Armijo rule.

In this case the search direction is the product of the inverse Hessian with the gradient of the function at the corresponding point

$$p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k$$

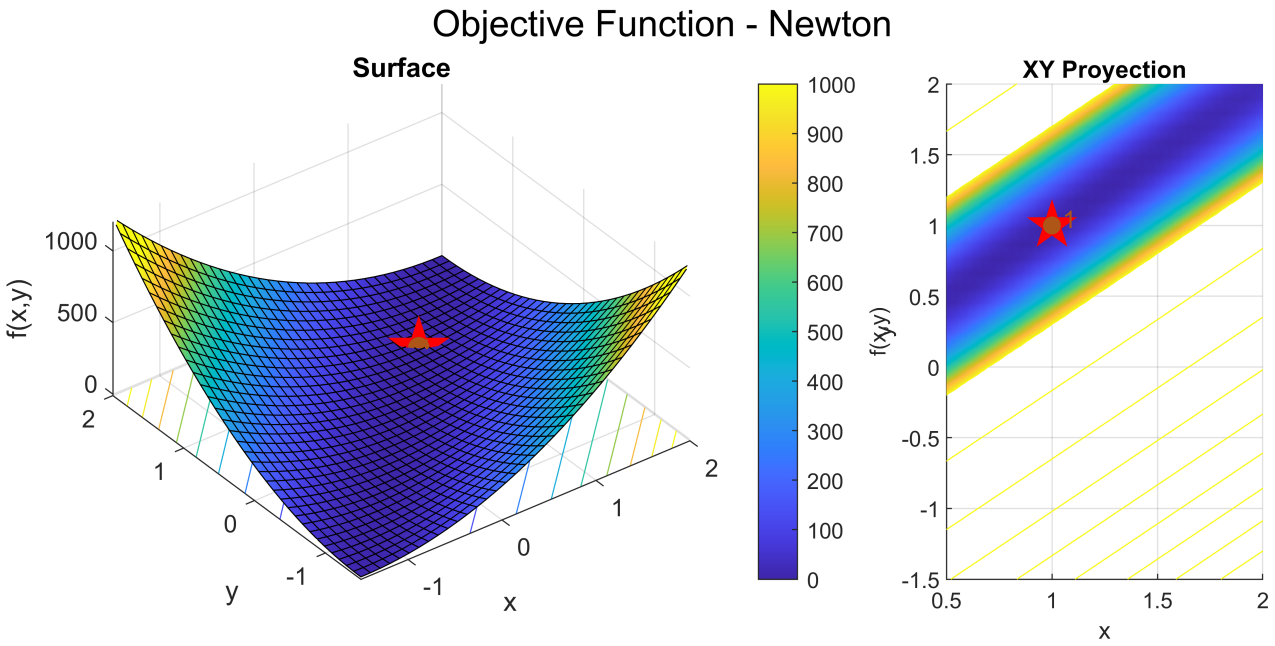
```
pk_func_Newton = @(xk) -Hessian(xk)\Gradient(xk);
[ks_N, xks_N, fks_N, grad_N, pks_N, tstar_N, I_N] = MethodImpl(pk_func_Newton, x0, Gradient, f
Tabla(xks_N, ks_N, grad_N, pks_N, fks_N)
```

```
ans = 3×8 table
```

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
1	1	150	150	22201	298	0	-149	-149
2	2	1	1	3.1052e-24	0	0	-0	-0
3	3	1	1	0	0	0	0	0

```
FinalPlot(fp2, x2_opt, x0, xks_N(:,2:end-1), 0, [-37 53], "Objective Function - Newton", 1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```



c) Quasi-Newton BFGS

Such that employs the backward globalization technique with Armijo rule

In this case the search direction is the product of the inverse approximate Hessian (B_k) with the gradient of the function at the corresponding point, in this case the Hessian approximation is calculated each iterations as follows

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla f_{k+1} - \nabla f_k, \qquad B_{k+1} s_k = y_k$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T y_k}$$

then the search direction is

$$p_k = -B_k^{-1} \nabla f_k$$

```
[ks_BFGS, xks_BFGS, fks_BFGS, grad_BFGS, pks_BFGS, tstar_BFGS, I_BFGS] = MethodBFGS(x0, Gradien
Tabla(xks_BFGS, ks_BFGS, grad_BFGS, pks_BFGS, fks_BFGS)
```

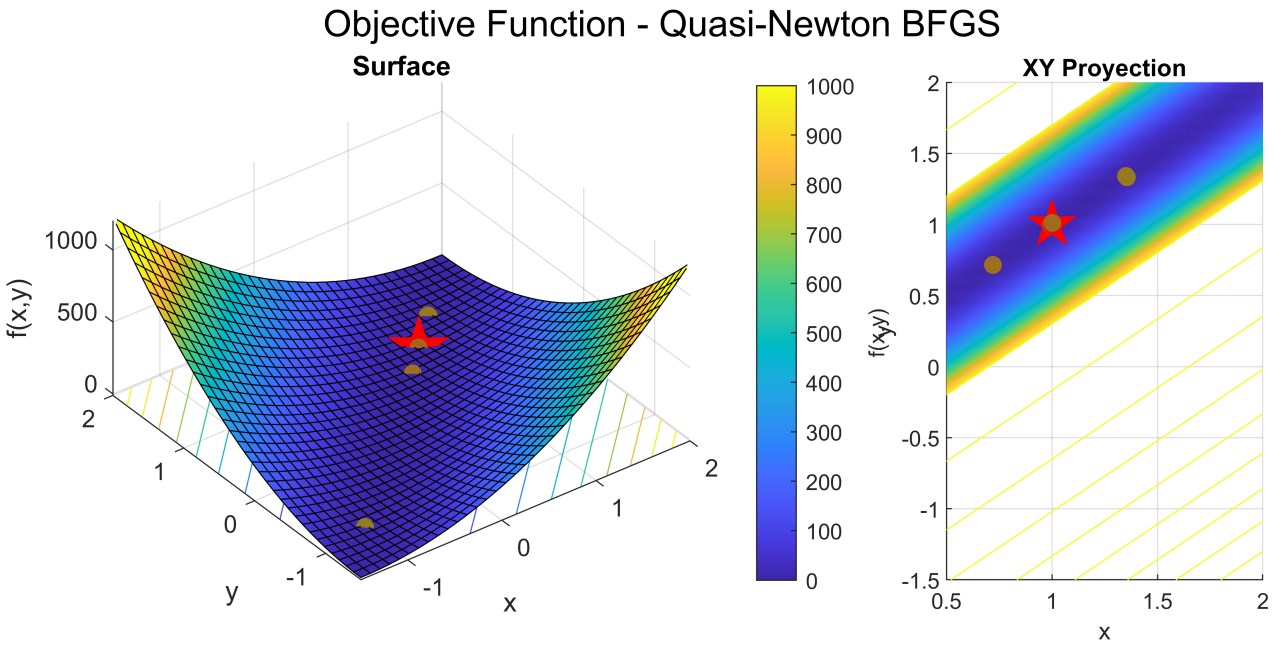
ans = 35×8 table

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
1	1	150	150	22201	298	0	-298	0
2	2	147.6719	150	2.2055e+04	-172.2812	465.6250	172.2812	-465.6250
3	3	148.0084	149.0906	2.1729e+04	77.5739	216.4429	-443.9644	-293.2487
4	4	134.1345	139.9266	2.1080e+04	-892.1473	1.1584e+03	38.6456	-2.1385e+03
5	5	134.2100	135.7497	1.7982e+04	-41.5305	307.9504	-6.2765e+03	-5.5576e+03
6	6	36.1398	48.9121	1.7548e+04	-2.4842e+03	2.5545e+03	-1.9504e+03	-7.0581e+03
7	7	32.3303	35.1267	1.7636e+03	-496.6214	559.2820	-6.4161e+03	-6.9914e+03
8	8	7.2675	7.8167	69.4420	-97.3031	109.8380	-1.4490e+03	-1.6716e+03
9	9	4.4373	4.5519	13.1284	-16.0456	22.9202	-697.2336	-698.8925
10	10	-1.0098	-0.9082	5.0725	-24.3479	20.3282	605.3664	573.6400
11	11	1.3549	1.3326	0.1756	5.1677	-4.4580	-1.9645	4.4206
12	12	1.3510	1.3412	0.1329	2.6658	-1.9638	-40.3652	-40.0641
13	13	0.7203	0.7152	0.0808	0.4635	-1.0228	36.0297	37.8457
14	14	1.0018	1.0109	0.0082	-1.8110	1.8146	-2.6465	-6.2420
15	15	0.9966	0.9987	4.3174e-04	-0.4168	0.4101	2.2536	1.7143
16	16	1.0010	1.0020	1.0055e-04	-0.1974	0.1995	-0.3301	-0.7245
17	17	1.0004	1.0006	5.3075e-06	-0.0446	0.0454	0.0564	-0.0158
18	18	1.0005	1.0006	9.9940e-07	-0.0161	0.0172	-0.1252	-0.1597
19	19	1.0003	1.0003	1.0640e-07	-0.0032	0.0037	-0.0576	-0.0646
20	20	1	1	9.8972e-09	0.0019	-0.0018	-0.0089	-0.0061
21	21	1	1	1.8369e-09	7.3401e-04	-6.8210e-04	-0.0071	-0.0062
22	22	1	1	3.1878e-12	3.2423e-06	-6.7488e-06	4.1797e-04	4.2306e-04
23	23	1	1	2.2902e-12	1.8115e-06	1.2128e-06	-3.8643e-04	-3.8752e-04
24	24	1	1	2.2713e-12	-2.5253e-06	-4.8846e-07	9.4276e-06	5.4389e-06
25	25	1	1	2.2679e-12	1.0234e-05	-1.2953e-05	-2.7445e-06	1.2813e-05
26	26	1	1	1.8793e-12	-1.9418e-06	-7.9883e-07	3.9834e-05	2.9762e-05
27	27	1	1	1.8053e-12	1.4418e-05	-1.6537e-05	2.5052e-05	5.7469e-05
28	28	1	1	1.0579e-12	1.8534e-06	-3.8737e-06	9.5401e-05	9.1768e-05
29	29	1	1	8.1063e-13	1.6189e-05	-1.5228e-05	-4.7407e-05	-1.8463e-05
30	30	1	1	1.8892e-13	4.6979e-06	-3.9221e-06	-2.0545e-05	-1.7957e-05
31	31	1	1	4.7870e-14	-4.0326e-06	4.1664e-06	-1.6323e-06	-9.4744e-06
32	32	1	1	7.0983e-15	-9.7570e-07	1.1031e-06	-1.2804e-05	-1.3943e-05
33	33	1	1	3.0095e-16	-1.8623e-07	2.1358e-07	-3.1646e-06	-3.5955e-06

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
34	34	1	1	3.9552e-17	1.2564e-07	-1.2302e-07	-7.9664e-07	-6.3320e-07
35	35	1	1	8.8152e-18	5.8684e-08	-5.9176e-08	-3.0356e-08	4.7676e-08

```
FinalPlot(fp2, x2_opt, x0, xks_BFGS(:,2:end-1), 0, [-37 53], "Objective Function - Quasi-Newtor
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```



d) Newton Dogleg Trust Region

Such that employs the backward globalization technique with the Dogleg trust region.

$$\min_{x\in\mathbb{R}^2} \quad f^T + g^T x + \frac{1}{2} x^T H x$$

First, let's verify that the matrix H is positive defined by checking its eigenvalues

```
eig(H)'
```

```
ans = 1×2
    0.9975  401.0025
```

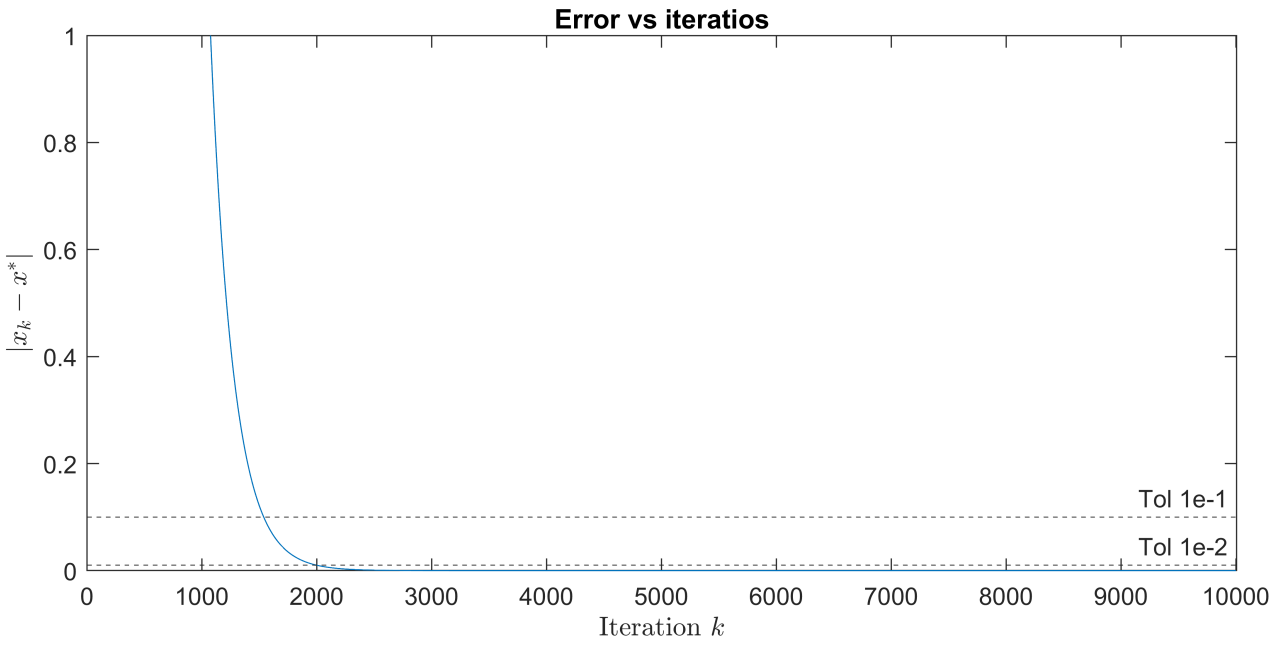
here you can see that both eigenvalues are nonzero positive numbers, then the matrix H is positive defined. First we test the `NewtonTrustRegion function` with 10000 iterations and then we solve again the problem but using the correct number of iterations for some tolerance.

```
[xks_NDTR,time_NDTR, Ies, fks_NDTR, grad_NDTR, pks_NDTR] = NewtonTrustRegion(10000, x0, f, Grad
```

```
clf;
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```

```
figure('Position', [10 10 900 400])
plot(1:10000+1, Ies)
yline(1e-1, '--', Label='Tol 1e-1'); yline(1e-2, '--', Label='Tol 1e-2');
ylabel("$|x_{k}-x^*|$",Interpreter="latex")
xlabel("Iteration $k$",Interpreter="latex")
xlim([0 10010]); ylim([0 1])
title("Error vs iteratios")
```




```
tol1_ind = find(Ies < 1e-1); tol1_ind = tol1_ind(1);
tol2_ind = find(Ies < 1e-3); tol2_ind = tol2_ind(1);
tol3_ind = find(Ies < 1e-8); tol3_ind = tol3_ind(1);
disp(['Required iteratios to get a tolerance of 1e-1 ' num2str(tol1_ind) newline...
      'Required iteratios to get a tolerance of 1e-3 ' num2str(tol2_ind) newline ...
      'Required iteratios to get a tolerance of 1e-8 ' num2str(tol3_ind)])
```

```
Required iteratios to get a tolerance of 1e-1 1540
Required iteratios to get a tolerance of 1e-3 2465
Required iteratios to get a tolerance of 1e-8 4780
```

Implementing the method to get a tolerance of $1e-8$.

```
[xks_NDTR,time_NDTR, Ies, ks_NDTR, fks_NDTR, grad_NDTR, pks_NDTR] = NewtonTrustRegion(tol3_ind,
T_NDTR = Tabla(xks_NDTR, ks_NDTR, grad_NDTR, pks_NDTR, fks_NDTR)
```

```
T_NDTR = 4781×8 table
```

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
1	1	150	150	22201	298	0	-0.5000	0
2	2	149.5000	150	2.2077e+04	197	100	-0.5000	0
3	3	148.6083	149.5474	2.1876e+04	107.4050	187.8116	-0.8917	-0.4526
4	4	148.1119	148.6793	2.1674e+04	180.7409	113.4829	-0.4964	-0.8681
5	5	147.2650	148.1475	2.1471e+04	116.0166	176.5134	-0.8469	-0.5317
6	6	146.7157	147.3119	2.1269e+04	172.1994	119.2321	-0.5493	-0.8357
7	7	145.8936	146.7426	2.1066e+04	119.9773	169.8098	-0.8222	-0.5693
8	8	145.3165	145.9259	2.0864e+04	166.7580	121.8751	-0.5770	-0.8167
9	9	144.5092	145.3358	2.0663e+04	121.6831	165.3353	-0.8074	-0.5901
10	10	143.9164	144.5305	2.0463e+04	163.0257	122.8072	-0.5927	-0.8054
11	11	143.1177	143.9288	2.0263e+04	122.0186	162.2167	-0.7987	-0.6017
12	12	142.5166	143.1296	2.0065e+04	160.4234	122.6097	-0.6011	-0.7992
13	13	141.7220	142.5224	1.9867e+04	121.3789	160.0652	-0.7945	-0.6072
14	14	141.1178	141.7256	1.9670e+04	158.6868	121.5489	-0.6042	-0.7968
15	15	140.3239	141.1175	1.9474e+04	119.9405	158.7074	-0.7939	-0.6081
16	16	139.7210	140.3197	1.9279e+04	157.7098	119.7323	-0.6029	-0.7978
17	17	138.9245	139.7150	1.9086e+04	117.7578	158.0913	-0.7965	-0.6047
18	18	138.3272	138.9130	1.8893e+04	157.4841	117.1703	-0.5974	-0.8020
19	19	137.5249	138.3161	1.8702e+04	114.8039	158.2458	-0.8023	-0.5969
20	20	136.9377	137.5067	1.8511e+04	158.0705	113.8048	-0.5872	-0.8094
21	21	136.1261	136.9224	1.8322e+04	110.9946	159.2577	-0.8115	-0.5843
22	22	135.5543	136.1020	1.8135e+04	159.5759	109.5328	-0.5718	-0.8204
23	23	134.7299	135.5361	1.7949e+04	106.2164	161.2433	-0.8245	-0.5659
24	24	134.1798	134.7010	1.7764e+04	162.1142	104.2453	-0.5501	-0.8351
25	25	133.3386	134.1601	1.7581e+04	100.3829	164.2944	-0.8411	-0.5409
26	26	132.8173	133.3068	1.7400e+04	165.7300	97.9046	-0.5214	-0.8533
27	27	131.9563	132.7982	1.7220e+04	93.5357	168.3768	-0.8610	-0.5086
28	28	131.4707	131.9240	1.7043e+04	170.2760	90.6654	-0.4856	-0.8742
29	29	130.5880	131.4540	1.6868e+04	85.9740	173.2020	-0.8827	-0.4700
30	30	130.1434	130.5583	1.6695e+04	175.3054	82.9814	-0.4446	-0.8957
31	31	129.2395	130.1304	1.6525e+04	78.2954	178.1837	-0.9039	-0.4278
32	32	128.8372	129.2149	1.6357e+04	180.1368	75.5377	-0.4023	-0.9155
33	33	127.9150	128.8282	1.6191e+04	71.1945	182.6356	-0.9222	-0.3867

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
34	34	127.5518	127.8965	1.6027e+04	184.1711	68.9326	-0.3632	-0.9317
35	35	126.6153	127.5460	1.5866e+04	65.0956	186.1350	-0.9365	-0.3505
36	36	126.2852	126.6020	1.5706e+04	187.2000	63.3704	-0.3301	-0.9439
37	37	125.3380	126.2814	1.5549e+04	59.9943	188.6817	-0.9472	-0.3206
38	38	125.0350	125.3284	1.5393e+04	189.3820	58.6879	-0.3030	-0.9530
39	39	124.0798	125.0324	1.5239e+04	55.6352	190.5244	-0.9552	-0.2960
40	40	123.7995	124.0725	1.5087e+04	190.9959	54.6030	-0.2803	-0.9599
41	41	122.8380	123.7976	1.4937e+04	51.7516	191.9244	-0.9615	-0.2749
42	42	122.5776	122.8321	1.4788e+04	192.2645	50.8908	-0.2603	-0.9655
43	43	121.6109	122.5762	1.4640e+04	48.1653	193.0566	-0.9667	-0.2559
44	44	121.3689	121.6060	1.4494e+04	193.3194	47.4183	-0.2421	-0.9703
45	45	120.3977	121.3677	1.4350e+04	44.7795	194.0158	-0.9712	-0.2382
46	46	120.1728	120.3934	1.4207e+04	194.2283	44.1173	-0.2249	-0.9744
47	47	119.1976	120.1719	1.4066e+04	41.5457	194.8495	-0.9752	-0.2215
48	48	118.9891	119.1938	1.3926e+04	195.0254	40.9528	-0.2085	-0.9780
49	49	118.0104	118.9883	1.3787e+04	38.4378	195.5830	-0.9787	-0.2055
50	50	117.8176	118.0071	1.3650e+04	195.7300	37.9051	-0.1928	-0.9812
51	51	116.8358	117.8170	1.3514e+04	35.4402	196.2314	-0.9818	-0.1901
52	52	116.6581	116.8329	1.3380e+04	196.3549	34.9613	-0.1777	-0.9841
53	53	115.6736	116.6576	1.3247e+04	32.5416	196.8056	-0.9845	-0.1753
54	54	115.5104	115.6710	1.3115e+04	196.9093	32.1115	-0.1631	-0.9866
55	55	114.5235	115.5100	1.2985e+04	29.7333	197.3137	-0.9870	-0.1610
56	56	114.3745	114.5212	1.2856e+04	197.4008	29.3481	-0.1490	-0.9888
57	57	113.3853	114.3742	1.2728e+04	27.0082	197.7625	-0.9891	-0.1471
58	58	113.2500	113.3833	1.2602e+04	197.8356	26.6645	-0.1353	-0.9908
59	59	112.2590	113.2498	1.2477e+04	24.3603	198.1576	-0.9910	-0.1336
60	60	112.1370	112.2572	1.2353e+04	198.2188	24.0551	-0.1220	-0.9925
61	61	111.1443	112.1368	1.2230e+04	21.7846	198.5040	-0.9927	-0.1205
62	62	111.0352	111.1427	1.2109e+04	198.5550	21.5153	-0.1091	-0.9940
63	63	110.0410	111.0350	1.1989e+04	19.2764	198.8056	-0.9942	-0.1077
64	64	109.9445	110.0397	1.1870e+04	198.8481	19.0408	-0.0965	-0.9953
65	65	108.9490	109.9444	1.1752e+04	16.8318	199.0662	-0.9954	-0.0953
66	66	108.8648	108.9479	1.1636e+04	199.1015	16.6280	-0.0843	-0.9964

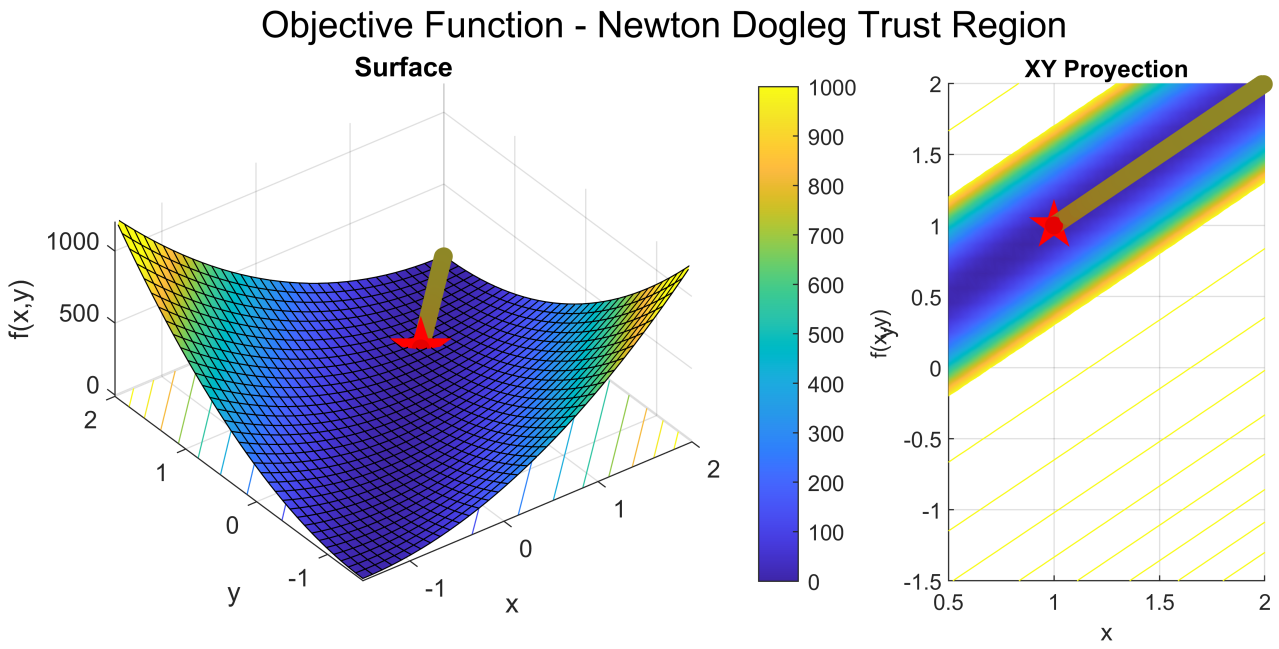
	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
67	67	107.8682	108.8647	1.1520e+04	14.4474	199.2891	-0.9965	-0.0832
68	68	107.7959	107.8673	1.1406e+04	199.3183	14.2736	-0.0723	-0.9974
69	69	106.7985	107.7959	1.1293e+04	12.1201	199.4769	-0.9974	-0.0714
70	70	106.7378	106.7977	1.1181e+04	199.5011	11.9746	-0.0606	-0.9982
71	71	105.7396	106.7378	1.1070e+04	9.8470	199.6323	-0.9982	-0.0599
72	72	105.6904	105.7390	1.0960e+04	199.6525	9.7283	-0.0493	-0.9988
73	73	104.6916	105.6904	1.0852e+04	7.6255	199.7576	-0.9988	-0.0487
74	74	104.6534	104.6911	1.0744e+04	199.7744	7.5324	-0.0381	-0.9993
75	75	103.6541	104.6534	1.0638e+04	5.4534	199.8548	-0.9993	-0.0377
76	76	103.6269	103.6538	1.0532e+04	199.8691	5.3846	-0.0273	-0.9996
77	77	102.6272	103.6268	1.0428e+04	3.3285	199.9259	-0.9996	-0.0269
78	78	102.6106	102.6270	1.0325e+04	199.9382	3.2829	-0.0166	-0.9999
79	79	101.6107	102.6106	1.0222e+04	1.2489	199.9725	-0.9999	-0.0164
80	80	101.6045	101.6106	1.0121e+04	199.9835	1.2255	-0.0062	-1
81	81	100.6045	101.6045	1.0021e+04	-0.7872	199.9962	-1	-0.0061
82	82	100.6084	100.6123	9.9218e+03	198.4357	0.7811	0.0039	-0.9922
83	83	99.6183	100.6084	9.8236e+03	-0.7794	198.0160	-0.9901	-0.0039
84	84	99.6222	99.6260	9.7263e+03	196.4710	0.7733	0.0039	-0.9823
85	85	98.6419	99.6222	9.6300e+03	-0.7717	196.0555	-0.9803	-0.0039
86	86	98.6457	98.6496	9.5347e+03	194.5258	0.7657	0.0038	-0.9726
87	87	97.6752	98.6457	9.4403e+03	-0.7641	194.1144	-0.9706	-0.0038
88	88	97.6790	97.6828	9.3468e+03	192.5998	0.7581	0.0038	-0.9630
89	89	96.7180	97.6790	9.2543e+03	-0.7565	192.1925	-0.9610	-0.0038
90	90	96.7218	96.7255	9.1627e+03	190.6929	0.7506	0.0038	-0.9535
91	91	95.7703	96.7218	9.0719e+03	-0.7490	190.2896	-0.9514	-0.0037
92	92	95.7740	95.7778	8.9821e+03	188.8049	0.7432	0.0037	-0.9440
93	93	94.8320	95.7740	8.8932e+03	-0.7416	188.4056	-0.9420	-0.0037
94	94	94.8357	94.8394	8.8051e+03	186.9356	0.7358	0.0037	-0.9347
95	95	93.9030	94.8357	8.7180e+03	-0.7343	186.5403	-0.9327	-0.0037
96	96	93.9066	93.9103	8.6316e+03	185.0848	0.7285	0.0036	-0.9254
97	97	92.9832	93.9067	8.5462e+03	-0.7270	184.6933	-0.9235	-0.0036
98	98	92.9868	92.9904	8.4616e+03	183.2523	0.7213	0.0036	-0.9163
99	99	92.0725	92.9868	8.3778e+03	-0.7198	182.8647	-0.9143	-0.0036

	k	x	y	f	Grad_x	Grad_y	pk_x	pk_y
100	100	92.0760	92.0796	8.2948e+03	181.4379	0.7142	0.0036	-0.9072

⋮

```
FinalPlot(fp2, x2_opt, x0, xks_NDTR(:,2:end-1), 0, [-37 53], "Objective Function - Newton Dogle
```

Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your function to return an output with the same size and shape as the input arguments.




```
Method  = ["Steepest Descent"; "Newton"; "BFGS"; "Newton Trust Region"; "Matlab-quadprog"; "Cas
```

Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your function to return an output with the same size and shape as the input arguments.

```
Time    = [tstar_SD; tstar_N; tstar_BFGS; time_NDTR; tstar_Matlab; tstart_casADI];
No_Tter = [I_SD; I_N; I_BFGS; tol3_ind; output.iterations; sol.stats.iter_count];
sortrows(table(Method, Time, No_Tter),2)
```

ans = 6×3 table

	Method	Time	No_Tter
1	"Newton"	0.0032	2
2	"BFGS"	0.0061	34
3	"Matlab-quadprog"	0.0503	1
4	"Steepest Descent"	0.0641	1915
5	"Newton Trust Region"	0.1016	4780

	Method	Time	No_Tter
6	"CasADI"	6.0842	1

When the start point is $(1.5, 1.5)$ and the tolerance is 1e-8, difference between solutions $|x_{k+1} - x_k| < \text{tol}$.

	Method	Time	No_Tter
1	"Newton"	0.0036	2
2	"BFGS"	0.0100	24
3	"Newton Trust Region"	0.0510	3634
4	"Matlab-quadprog"	0.0556	1
5	"Steepest Descent"	0.0631	1323
6	"CasADI"	4.2649	1

Here we find that the best methods, for both initial points, are the matlab and CasADI methods since the arrive to the solution in one step but the execution time is not te best. The fatest method is the Newton method and then the quasi-Newton BFGS method but grater number of iterations.

Comparing the results, the most efficient mehotd is the Newton method that is the fastest and arrive to the solution in two steps no matter the starting point.

Auxiliar Functions

Methods Implementation

```
function alphak = Backtracking(xk, f, Gradient, pk, Rule)
    global alpha0 rho
    alpha = alpha0;  %c = 0;
    while not(Rule(xk, f, Gradient, pk, alpha)) %&& c<100
        alpha = rho*alpha; %c = c+1;
    end
    alphak = alpha; %display(c)
end

function [Ks, Xks, Fks, Grad, Pks, tstar, I] = MethodBFGS(x0, Gradient, f, Armijo)
    tstart = tic;
    IterMax   = 2000; tol = 1e-8;
    Bk        = eye(2);
    ks        = [1];
    Gradients = [Gradient(x0)];
    pks       = [-Bk\Gradient(x0)];
    xks       = [x0];
    fks       = [f(x0)];


    for i=2:IterMax
        alphak = Backtracking(xks(:,end)', f, Gradient, pks(:,end), Armijo);
```

```
        xks       = [xks xks(:,end)+alphak*pks(:,end)];
        Gradients = [Gradients Gradient(xks(:,end))];
        pks       = [pks -Bk\Gradient(xks(:,end))];
        fks       = [fks f(xks(:,end))];
        ks        = [ks i];

        sk = xks(:,end) - xks(:,end-1);
        yk = Gradient(xks(:,end)) - Gradient(xks(:,end-1));
        Bk = Bk - (Bk*(sk*sk')*Bk')/(sk'*Bk*sk)+(yk*yk')/(yk'*yk);

        if norm(xks(:,end)-xks(:,end-1),2) < tol
            break
        end
    end
    Ks = ks;   Xks = xks; Pks = pks;
    Fks = fks; Grad = Gradients;
    tstar = toc(tstart); I = i-1;
    %display(tstar);
end


function [Ks, Xks, Fks, Grad, Pks, tstar, I] = MethodImpl(pk_func, x0, Gradient, f, Armijo)
    tstart = tic;
    IterMax   = 2000; tol = 1e-8;
    ks        = [1];
    Gradients = [Gradient(x0)];
    pks       = [pk_func(x0)];
    xks       = [x0];
    fks       = [f(x0)];

    for i=2:IterMax
        alphak = Backtracking(xks(:,end)', f, Gradient, pks(:,end), Armijo);
        xks    = [xks xks(:,end)+alphak*pks(:,end)];

        Gradients = [Gradients Gradient(xks(:,end))];
        pks       = [pks pk_func(xks(:,end))];
        fks       = [fks f(xks(:,end))];
        ks        = [ks i];

        if norm(xks(:,end)-xks(:,end-1),2) < tol
            break
        end
    end
    Ks = ks;   Xks = xks; Pks = pks;
    Fks = fks; Grad = Gradients;
    tstar = toc(tstart); I = i-1;
    %display(tstar);
end
```

```
function [xk,c,time] = Iteration(x0, func, Gradient)
    tStart = tic;
    xk = [x0]; tol = 1e-8; %MaxIter = 500; c = 0;
    while norm(Gradient(xk(:,end)), "inf")  >= tol % && (c < MaxIter)
        xk = [xk xk(:,end)+func(xk(:,end))];
    end
    c = size(xk,2)-1;
    time = toc(tStart);
end

function [xks, time, Ies, Ks, Fks, Grad, Pks] = NewtonTrustRegion(N, x0, f, Gradient, Hessian)
    tstart  = tic;
    DeltaUp = 1; eta = 1/6; % eta [0, 1/4), DeltaUp = DeltaMoño > 0
    Delta0  = DeltaUp/2;    % Deltak (0, DeltaUp)
    Deltak  = Delta0;
    xks     = [x0];

    Grad  = [Gradient(x0)];
    Pks   = [-Deltak*Gradient(xks(:,end))/norm(Gradient(xks(:,end)),2)];
    Fks   = [f(xks(:,end))];
    Ks    = [1];

    m    = @(p, xk) f(xk) + Gradient(xk)'*p + 0.5*p'*Hessian(xk)*p;

    for k=1:N
    gk = Gradient(xks(:,end));  Bk = Hessian(xks(:,end));
    if gk'*Bk*gk <=0  tauk = 1;
    else              tauk = min(norm(gk,2)^3/(Deltak*gk'*Bk*gk),1);
    end
    pk   = -tauk*Deltak*gk/norm(gk,2); %pkc
    rhok = (f(xks(:,end))-f(xks(:,end)+pk)) / (m(zeros(2,1),xks(:,end))-m(pk,xks(:,end)));
    if rhok < 0.25        Deltak = 0.25*Deltak;
    else
        if (rhok > 0.75) && (norm(pk,2)==Deltak)    Deltak = min(2*Deltak, DeltaUp);
        else                                        Deltak = Deltak;
        end
    end
    if rhok > eta   xks = [xks xks(:,end)+pk];
    else            xks = [xks xks(:,end)];
    end
    Grad = [Grad Gradient(xks(:,end))];
    Pks  = [Pks pk];
    Fks  = [Fks f(xks(:,end))];
    Ks   = [Ks k+1];
    end
    time = toc(tstart);
    Diff = xks-ones(2,N+1);
    Ies  = zeros(1,N);
    for i=1:N+1  Ies(i) = norm(Diff(:,i),2); end
end
```

```
function T=Tabla(xks, ks, grad, pks, fks)
    k  = ks'; f = fks';
    x  = xks(1,:)'; y = xks(2,:)';
    Grad_x = grad(1,:)'; Grad_y = grad(2,:)';
    pk_x = pks(1,:)'; pk_y = pks(2,:)';
    T = table(k,x,y,f,Grad_x,Grad_y,pk_x,pk_y);
end
```

Plot Functions

```
function PlotF3d(f, xstar, x0, xks, flag, viewed, titulo, ax, flagtext)
    %figure();%'Position', [10 10 900 400]); %clf();
    if flag==1
        xlim([0.1 1.2]); ylim([0.1 1.2]); zlim([-10 80]); clim([-5 50]);
        interval = [-0.5 1.2 -0.5 1.2];
    else
        xlim([-1.5 2]); ylim([-1.5 2]); zlim([-10 1200]); clim([0 1000]);
        interval = [-1.5 2 -1.5 2];
    end

    hold on
    fsurf(ax, f, interval,'ShowContours','on');
    plot3(ax, xstar(1),xstar(2),f(xstar(1), xstar(2)), 'p', MarkerSize=25, MarkerFaceColor="red
    plot3(ax, x0(1), x0(2), f(x0(1), x0(2)), 'square', MarkerSize=10, MarkerFaceColor=[0.4660 0
    text(ax, x0(1), x0(2), f(x0(1), x0(2))+5, 'x0', Color=[0.4660 0.6740 0.1880])
    c = colorbar;  c.Label.String='f(x,y)';
    left_color = [0.4660 0.6740 0.1880];  right_color = [0.9 0 0]; % color according to the att
    cmap = interp1([0, 1], [left_color; right_color], linspace(0, 1, size(xks,2)+2));
    cmap = cmap(2:end-1,:);
    %text(xstar(1),xstar(2),f(xstar(1), xstar(2))+10, "goal", Color="red")
    if flagtext==1
    for i=1:size(xks,2)
        plot3(ax, xks(1,i),xks(2,i),f(xks(1,i), xks(2,i)), '.', MarkerSize=30, Color=cmap(i,:))
        text(ax, xks(1,i),xks(2,i),f(xks(1,i), xks(2,i))+5,num2str(i), Color=cmap(i,:))
    end
    else
       for i=1:size(xks,2)
        plot3(ax, xks(1,i),xks(2,i),f(xks(1,i), xks(2,i)), '.', MarkerSize=25, Color=cmap(i,:))
       end
    end
    hold off

    grid on
    xlabel('x');   ylabel('y');    zlabel('f(x,y)');
    view(viewed); title(titulo);
end

function PlotF2d(f, xstar, x0, xks, flag, viewed, titulo, ax, flagtext)
    %figure();%'Position', [10 10 900 400]); %clf();
```

```
    if flag==1
        xlim([0.1 1.2]); ylim([0.1 1.2]); zlim([-10 50]); clim([-5 50]);
        interval = [-0.5 1.2 -0.5 1.2];
    else
        xlim([0.5 2]); ylim([-1.5 2]); zlim([-10 50]); clim([0 50]);
        interval = [-1.5 2 -1.5 2];
    end

    hold on
    fsurf(ax, f, interval,'ShowContours','on', "EdgeColor",'none','AmbientStrength', 0.2);
    plot3(ax, xstar(1),xstar(2),f(xstar(1), xstar(2))+10, 'p', MarkerSize=20, MarkerFaceColor='
    plot3(ax, x0(1), x0(2), f(x0(1), x0(2))+10, 'square', MarkerSize=10, MarkerFaceColor=[0.466
    text(ax, x0(1)+0.05, x0(2)+0.05, f(x0(1), x0(2))+10, 'x0', Color=[0.4660 0.6740 0.1880])
    %text(xstar(1),xstar(2),f(xstar(1), xstar(2))+10, "goal", Color="red")
    left_color = [0.4660 0.6740 0.1880];  right_color = [0.9 0 0]; % color according to the att
    cmap = interp1([0, 1], [left_color; right_color], linspace(0, 1, size(xks,2)+2));
    cmap = cmap(2:end-1,:);

    if flagtext==1
    for i=1:size(xks,2)
        plot3(ax, xks(1,i),xks(2,i),f(xks(1,i), xks(2,i))+10, '.', MarkerSize=25, Color=cmap(i,
        text(ax, xks(1,i)+0.05,xks(2,i)+0.05,f(xks(1,i), xks(2,i))+10,num2str(i), Color=cmap(i,
    end
    else
    for i=1:size(xks,2)
        plot3(ax, xks(1,i),xks(2,i),f(xks(1,i), xks(2,i))+10, '.', MarkerSize=25, Color=cmap(i,
    end
    end
    hold off

    grid on
    xlabel('x');   ylabel('y');    zlabel('f(x,y)');
    view(viewed); title(titulo);
end




function FinalPlot(f, xstar, x0, xks, flag, viewed, titulo, flagtext)
    clf();
    h = figure('Position', [10 10 900 400]);

    subplot(1,3,[1:2]);
    ax1 = gca;

    PlotF3d(f, xstar, x0, xks, flag, viewed, "Surface", ax1, flagtext)

    subplot(1,3,3);
    ax2 = gca;
    PlotF2d(f, xstar, x0, xks, flag, [0 90], "XY Proyection", ax2, flagtext)
```

```
    sgtitle(titulo)
end
```