Pedro Ferreira
201806093@up.pt


# 1 Exercise PP. 5

Use the Newton and Quasi-Newton (DFP and BFGS) methods to find the minimum of the unconstrained objective function $U(x,y)$ given by:

$$U(x,y) = 2x + \frac{20}{xy} + \frac{y}{3}$$

## 1.1 MATLAB Files

**PP5_main.m**

```
% Main Script
% Calls: PP5_data, armijo, strong_wolfe, Newton_method, Quasi_Newton

clear
clc
close all    % Close open figures

for k=1:3
    PP5_data % Initialize Data

    fig = figure(k);  % Open Contour Plot Figure

    if k==1
        fig.Name='Newton Method';
        fprintf(['------------------------------------- \n', ...
                 'Newton Method \n \n'])
    elseif k==2
        cc=0;
        fig.Name='Quasi-Newton Method - DFP';
        fprintf(['------------------------------------- \n', ...
                 'Quasi-Newton Method - DFP \n \n'])
    elseif k==3
        cc=1;
        fig.Name='Quasi-Newton Method - BFGS';
        fprintf(['------------------------------------- \n', ...
                 'Quasi-Newton Method - BFGS \n \n'])
    end

    % Draw Conturs
    fcontour(f.fun,[0 4.5 4 8])
```

```
    xlabel('$x$','Interpreter','latex')
    ylabel('$y$','Interpreter','latex')
    title('$U(x,y) = 2x +\frac{20}{xy}$ + $\frac{y}{3}$','Interpreter','latex')
    hold on
    grid on
    axis equal

    % Search Cycle
    t=0;
    while t<tmax && norm(f.grad(x(1),x(2))) > precision
        t=t+1;
        if k==1
            Newton_method % Computes the search direction
            alpha=armijo(f.fun,d,x,f.grad); % Step Length
        else
            Quasi_Newton % Computes the search direction
            alpha = strong_wolfe(f.fun,f.grad,x,f_obj,d); % Step Length
        end

        % New search point
        x_old=x;
        f_old=f_obj;

        x=x+alpha*d;
        f_obj=f.fun(x(1),x(2));

        % Plot the current search path
         plot([x_old(1),x(1)],[x_old(2),x(2)],'o-r')
    end
    % Plot point of minima in a different color
    plot(x(1),x(2),'ok',MarkerFaceColor='k')

    % Print the results
    fprintf([ ...
             'Number of Iterations: %d\n\n', ...
             'Point of Minima: [%d , %d]\n\n', ...
             'Objective Function Minimum Value after Optimization: %d\n\n'], ...
             t,x(1),x(2),round(f_obj,8))
end
```

### PP5_data.m

```
% Objective Function
syms x1 x2
f.fun  = matlabFunction(2*x1+20/(x1*x2)+x2/3);
f.grad = matlabFunction(gradient(f.fun, [x1 x2]));
f.hess = matlabFunction(hessian(f.fun, [x1 x2]));

% Maximum number of iterations
tmax = 2000;

% Initial Point
x = [1;5];
f_obj = f.fun(x(1),x(2));

% Precision
precision = 1E-8;
```

### armijo.m

```
% User defined parameters
delta=0.5;
gamma=0.1;
c=0.5;

% Initial guess for the step length
a=c*abs(dot(grad_f(x(1),x(2)),d))/(norm(d))^2;

% Application of the Armijo Condition
while true
    if f(x(1)+a*d(1),x(2)+a*d(2))<=f(x(1),x(2))+gamma*a*dot(
       grad_f(x(1),x(2)),d)
        break
    else
        a=delta*a;
    end
end
alpha=a;
```

**strong_wolfe.m**

```
function alpha = strong_wolfe(func,grad,x0,f0,d)
% Compute a line search to satisfy the strong Wolfe conditions.
% Based on algorithm 3.5. Page 60. "Numerical Optimisation".
    Nocedal & Wright.
% INPUTS:
%  func: objective function handle
%  x0: starting point
%  f0: initial function evaluation
%  d:  search direction
% OUTPUTS:
% alpha: step length

% Variables Initialisation
g0 = grad(x0(1),x0(2));
c1 = 1e-4;
c2 = 0.9;
alpha_max = 2.5;
alpha_im1 = 0;
alpha_i   = 1;
f_im1 = f0;
dphi0 = g0.'*d;
i = 0;
max_iters = 20;

% Search for alpha satisfying the Strong-Wolfe conditions
while true

  x   = x0 + alpha_i*d;
  f_i = func(x(1),x(2));
  g_i = grad(x(1),x(2));

  if (f_i > f0 + c1*dphi0) || ( (i > 1) && (f_i >= f_im1) )
    alpha = alpha_zoom(func,grad,x0,f0,g0,d,alpha_im1,alpha_i);
    break;
  end
  dphi = g_i.'*d;
  if ( abs(dphi) <= -c2*dphi0 )
    alpha = alpha_i;
    break;
  end
  if ( dphi >= 0 )
    alpha = alpha_zoom(func,grad,x0,f0,g0,d,alpha_i,alpha_im1);
```

```
    break;
  end

  % Update alpha
  alpha_im1 = alpha_i;
  f_im1 = f_i;
  alpha_i = alpha_i + 0.8*(alpha_max-alpha_i);

  if isnan(alpha_i)
      break
  end

  if (i > max_iters)
    alpha = alpha_i;
    break;
  end

  i = i+1;

end

end

function alpha = alpha_zoom(func,grad,x0,f0,g0,d,alpha_lo,
   alpha_hi)
% Based on algorithm 3.6, Page 61. "Numerical Optimisation".
   Nocedal & Wright.
% INPUTS:
%  func: objective function handle
%  x0: starting point
%  f0: initial function evaluation
%  d:  search direction
% OUTPUTS:
%  alpha: zoomed in alpha.

% Variables Initialisation
c1 = 1e-4;
c2 = 0.9;
i = 0;
max_iters = 20;
dphi0 = g0.'*d;

while true
  alpha_i = 0.5*(alpha_lo + alpha_hi);
  alpha = alpha_i;
```

```
  x = x0 + alpha_i*d;
  [f_i] = func(x(1),x(2));
  g_i   = grad(x(1),x(2));
  x_lo = x0 + alpha_lo*d;
  f_lo = func(x_lo(1),x_lo(2));
  if ( (f_i > f0 + c1*alpha_i*dphi0) || ( f_i >= f_lo) )
    alpha_hi = alpha_i;
  else
    dphi = g_i.'*d;
    if ( ( abs(dphi) <= -c2*dphi0 ) )
      alpha = alpha_i;
      break;
    end
    if ( dphi * (alpha_hi-alpha_lo) >= 0 )
      alpha_hi = alpha_lo;
    end
    alpha_lo = alpha_i;
  end
  i = i+1;
  if (i > max_iters)
    alpha = alpha_i;
    break;
  end
end

end
```

**Newton_Method.m**

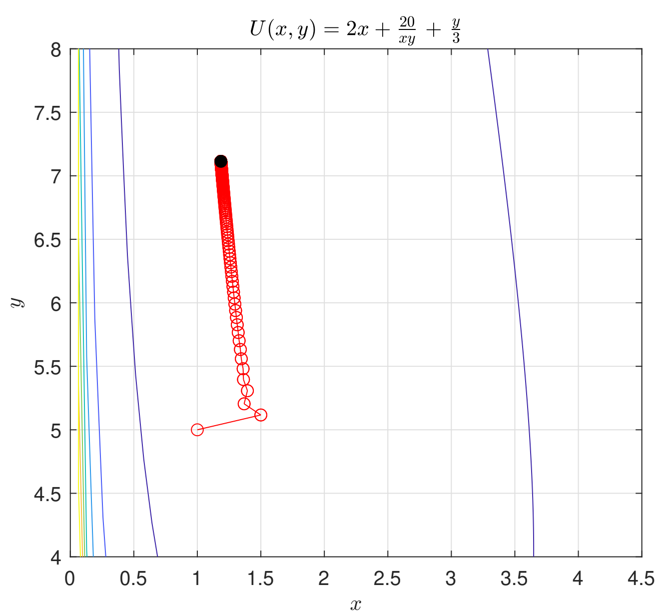
```
c1 = 1;      % c1>0
c2 = 2;      % c2>0
p  = 3;      % p>=2
q  = 4;      % q>=3

dn = -f.hess(x(1),x(2))\f.grad(x(1),x(2));

if f.grad(x(1),x(2)).'*dn<=-c1*norm(f.grad(x(1),x(2)))^q && norm(dn)^p<=c2*norm(f.grad(x(1),x(2)))
    d=dn;
else
    d=-f.grad(x(1),x(2));
end
```

**Quasi_Newton.m**

```
% cc = 0   - DFP
% cc = 1   - BFGS

if t==1
    H=eye(2);
else
    deltak = x-x_old;
    yk = f.grad(x(1),x(2))-f.grad(x_old(1),x_old(2));
    vk = deltak/(deltak.'*yk) - (H*yk)/(yk.'*H*yk);
    deltaH = (deltak*deltak.')/(deltak.'*yk) - (H*yk*(H*yk).')
       /(yk.'*H*yk) + cc*yk.'*H*yk*vk*(vk.');
    H = H+deltaH;
end

d=-H*f.grad(x(1),x(2));
```

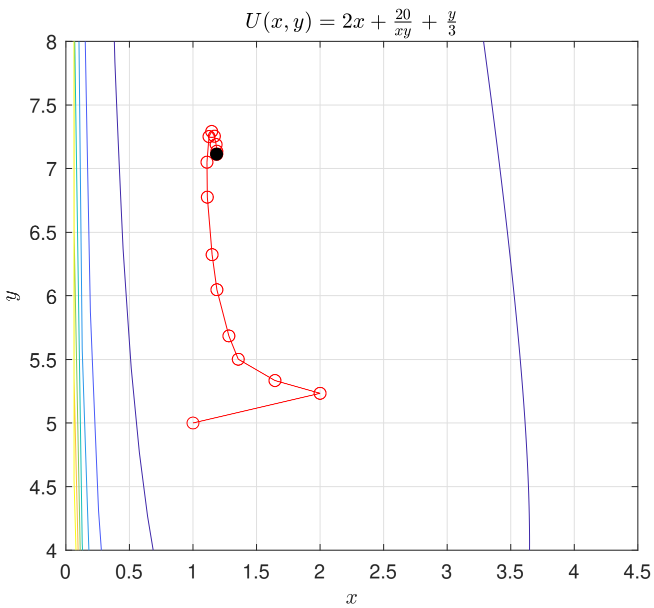
## 1.2 Results

```
--------------------------------------
Newton Method
Number of Iterations: 464
Point of Minima: [1.185631e+00 , 7.113786e+00]
Objective Function Minimum Value after Optimization: 7.113787e+00
```

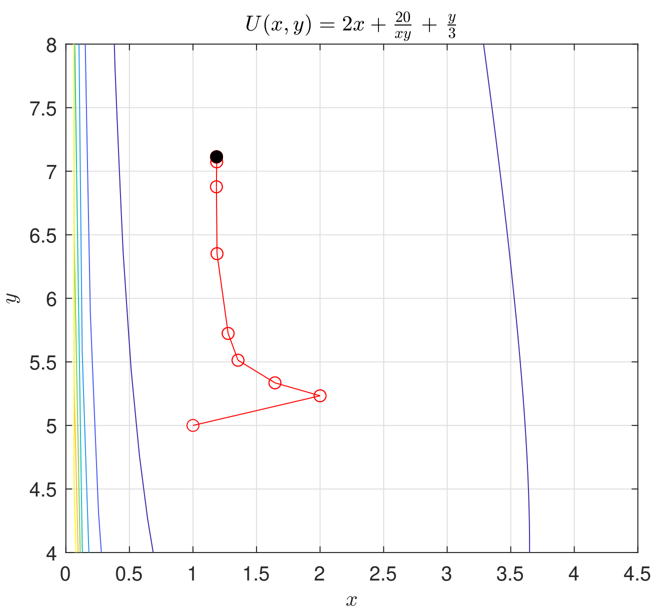



```
--------------------------------------
```

```
Quasi-Newton Method - DFP
Number of Iterations: 20
Point of Minima: [1.185631e+00 , 7.113787e+00]
Objective Function Minimum Value after Optimization: 7.113787e+00
```




```
-------------------------------------
Quasi-Newton Method - BFGS
Number of Iterations: 12
Point of Minima: [1.185631e+00 , 7.113787e+00]
Objective Function Minimum Value after Optimization: 7.113787e+00
```



# 2 Exercise PP. 7 - 1

Compare the Newton and Quasi-Newton (DFP and BFGS) methods using the Rosenbrock's parabolic valley test function.

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

$$\mathbf{X}_1(x_1, x_2) = (-1.2, 1.0), \quad f(\mathbf{X}_1) = 24.0$$
$$\mathbf{X}^* = (1.0, 1.0), \quad f(\mathbf{X}^*) = 0.0$$

## 2.1 MATLAB Files

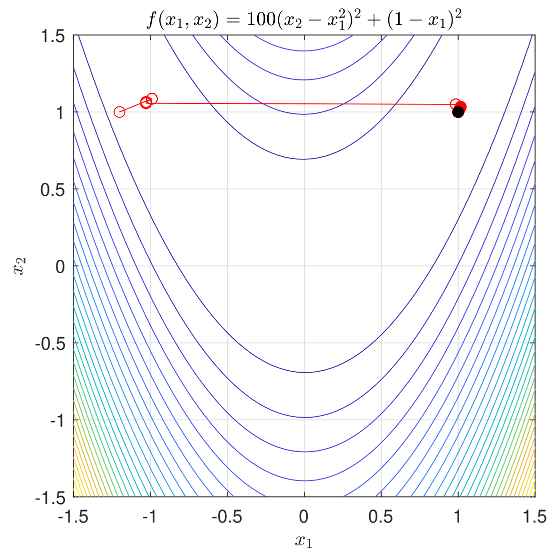
The MATLAB scripts used to perform the calculations were the same as the ones for exercise PP5, but in *PP5_main.n* line 9 was changed to *PP7 _data.m* in order to initialise the data for the new objective function and the title of the plot, in line 33, was changed as well to match the same function.

**PP5_data.m**

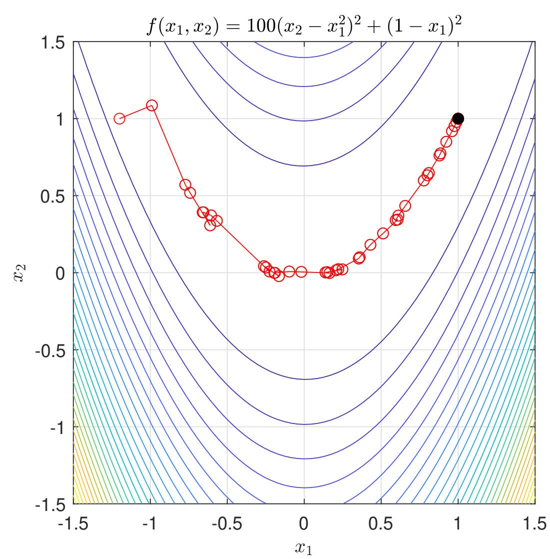
```
% Objective Function
syms x1 x2
f.fun  = matlabFunction(100*(x2-x1^2)^2+(1-x1)^2);
f.grad = matlabFunction(gradient(f.fun, [x1 x2]));
f.hess = matlabFunction(hessian(f.fun, [x1 x2]));

% Maximum number of iterations
tmax = 2000;

% Initial Point
x = [-1.2; 1];
f_obj = f.fun(x(1),x(2));

% Precision
precision = 1E-8;
```

## 2.2 Results

```
--------------------------------------
Newton Method
Number of Iterations: 74
Point of Minima: [1.000000e+00 , 1.000000e+00]
Objective Function Minimum Value after Optimization: 0
```




```
--------------------------------------
Quasi-Newton Method - DFP
Number of Iterations: 45
Point of Minima: [1.000000e+00 , 1.000000e+00]
Objective Function Minimum Value after Optimization: 0
```



```
-------------------------------------
Quasi-Newton Method - BFGS
Number of Iterations: 35
Point of Minima: [1.000000e+00 , 1.000000e+00]
Objective Function Minimum Value after Optimization: 0
```