MAT 423: Homework 8 (11/07)

Ex 1. [3pts]

a) We re-write ϕ as:

$$\phi(x,y) = x^2 \cdot \varphi(x) + y^2,$$

with $\varphi(x) = \frac{x^2}{4} - \frac{5x}{3} + 3$. The parabola φ has its minimum at $x_* = \frac{10}{3}$ where $\varphi(10/3) = 2/9 > 0$. Thus, for $c = \frac{2}{9} > 0$

$$\phi(x,y) \geq c \cdot x^2 + y^2 \geq c\|(x,y)\|_2^2 \longrightarrow +\infty,$$

as $\|(x,y)\|_2 \to +\infty$.

Let's pick a point $\mathbf{x}_0 \in \mathbb{R}^2$ (for instance $\mathbf{x}_0 = (0,0)$) and denote $m = \phi(\mathbf{x}_0)$. Since ϕ goes to infinity at infinity then there exists $R > 0$ such that:

$$\|(x,y)\| > R \quad \Rightarrow \quad \phi(x,y) \geq m.$$

Therefore,

$$\inf_{\mathbf{x}\in\mathbb{R}^2} \phi(\mathbf{x}) = \inf_{\|\mathbf{x}\|\leq R} \phi(\mathbf{x}).$$

Since ϕ is a continuous and the domain $\overline{B}(R) = \{\mathbf{x} \in \mathbb{R}^2 \,|\, \|\mathbf{x}\| \leq R\}$ is compact, we have:

$$\inf_{\|\mathbf{x}\|\leq R} \phi(\mathbf{x}) = \min_{\|\mathbf{x}\|\leq R} \phi(\mathbf{x}).$$

b) The critical points of ϕ satisfy:

$$\nabla\phi(x,y) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Rightarrow \quad \begin{cases} x^3 - 5x^2 + 6x = 0 \\ 2y = 0 \end{cases} \quad \Rightarrow \quad \begin{cases} x(x-2)(x-3) = 0 \\ y = 0 \end{cases}$$ 1pt

Therefore, there are three critical points $(0,0)$, $(2,0)$, and $(3,0)$. Since $\phi(0,0) = 0$ is the minimum of the three values, we deduce that the global minimum is necessarily at $\mathbf{x}_* = (0,0)$. Moreover, looking at the Hessian of ϕ at the two other critical points: 1pt

$$D^2\phi(x,y) = \begin{bmatrix} 3x^2 - 10x + 6 & 0 \\ 0 & 2 \end{bmatrix},$$

we find out that $(1,0)$ is a saddle point ($D^2\phi(2,0)$ has a positive and negative eigenvalue) and $(3,0)$ is a local minimum. .5+.5pt

Ex 2. [4pts]

a) Let $\mathbf{x}^{(0)} = (\frac{1}{2}, 1)$, the first iteration of the Newton's method is given by:

$$\mathbf{x}^{(1)} = \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix} - \left(\begin{bmatrix} \frac{7}{4} & 0 \\ 0 & 2 \end{bmatrix} \right)^{-1} \begin{pmatrix} \frac{15}{8} \\ 2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} - \frac{15}{14} \\ 0 \end{pmatrix}. \quad \boxed{1\text{pt}}$$

b) See below for the implementation of the BFGS method. $\boxed{1\text{pt}}$

We plot in figure 1 the evolution of $\phi(\mathbf{x}^{(k)})$ for both the Newton's method and quasi-Newton method BFGS in logarithmic scale. We observe that both methods converge faster than exponentially but that the Newton's method is converging faster. $\boxed{1\text{pt}}$

c) Starting from $\mathbf{y}_0 = (2.2, 1.1)$, we observe in figure 2 that both methods converge to the local minimum $(2, 0)$. More surprisisng, starting at $\mathbf{z}_0 = (3.2, 1.1)$, the algorithms converge to the saddle-point $(3, 0)$. Notice that gradient descent, although converging with slower rates, would never converge to a saddle point. $\boxed{1\text{pt}}$

Ex 3. [3pts]

a) See below for the implementation of the gradient descent algorithm. $\boxed{1\text{pt}}$

b) The method converges faster as λ increases until it reaches $\lambda = .5$ where the method does no longer converges (oscillations starts to occur). $\boxed{1+1\text{pt}}$

Extra) Using the adaptive scheme, the gradient descent adapts the learning rate and the method converges (see figure below and code). $\boxed{1\text{pt}}$

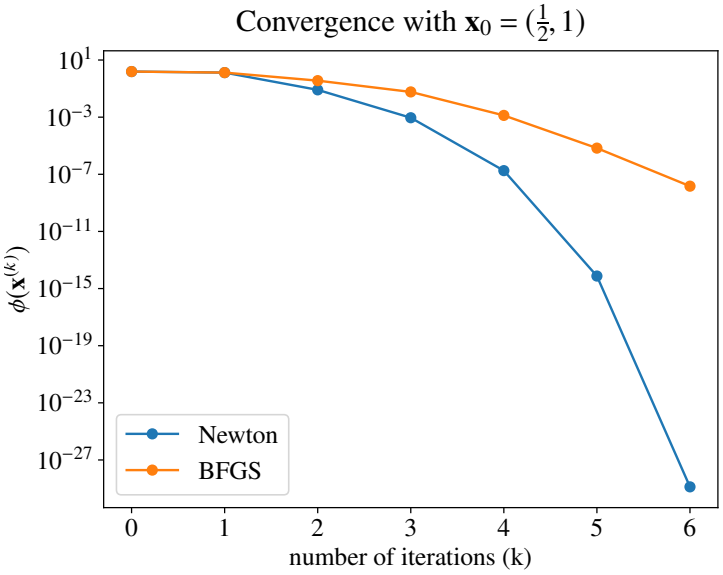

Figure 1: Comparison between the convergence of the Newton's method and the BFGS method. Both convergence faster than exponential and the Newton's method converges fastser.

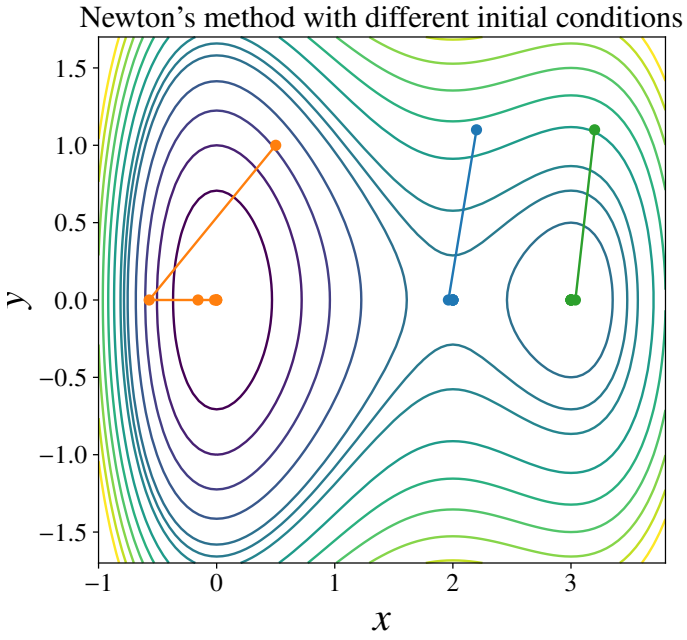

Figure 2: Newton's method using different initial conditions. The algorithm converges to three different points, they are all critical pointss.

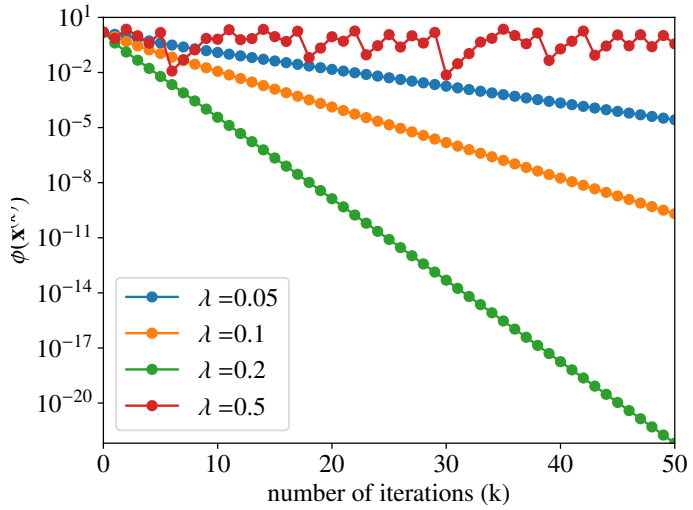

Figure 3: Convergence of the gradient descent algorithm depending on the learning rate λ. Convergence is 'linear' and not quadratic like the Newton's method and for $\lambda = .5$ the method does not converge anymore.

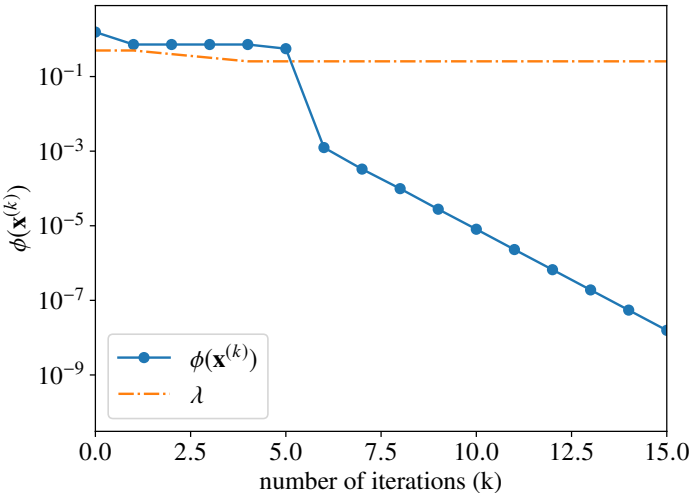

Figure 4: In the adaptive gradient descent method, the learning rate λ is updated when the function $\phi(\mathbf{x}^{(k)})$ stops decreasing. As a result, the method converges if when the initial learning rate λ_0 is equal .5.

```
function Newton_2D(nbIter,x0,y0,phi,dphi,D2_phi)
    """
       Newton's method in dimension 2 to find critical point of a function phi:R^2->R
        X_k+1 = X_k - [D^2 phi(X_k)]^{-1} dphi(X_k)
    """
    # init
    x,y = x0,y0
    value_xy = zeros(2,nbIter+1)
    value_xy[:,1] = [x0;y0]
    value_phi_Newton = zeros(nbIter+1)
    value_phi_Newton[1] = phi(x,y)
    # loop
    for k=1:nbIter
        # update
        x,y = [x;y] - inv(D2_phi(x,y))*dphi(x,y)
        # save
        value_xy[:,k+1] = [x;y]
        value_phi_Newton[k+1] = phi(x,y)
    end
    # output
    return value_xy,value_phi_Newton
end

function BFGS_2D(nbIter,x0,y0,phi,dphi,D2_phi)
    """
       BFGS method in dimension 2 to find critical point of a function phi:R^2->R
        . Step 1: A^0 = D^2 phi(X_k)
         X_1 = X_0 - [A^0]^{-1} dphi(X_0)
        . Step k+1: A^k+1 = A^k + ... (satisfy secant eq.)
         X_k+1 = X_k - [A^k]^{-1} dphi(X_k)
    """
    # init
    Xprev = [x0;y0]
    A = D2_phi(Xprev[1],Xprev[2])
    invA = inv(A)
    X = Xprev - invA*dphi(Xprev[1],Xprev[2])
    # init
    value_xy = zeros(2,nbIter+1)
    value_xy[:,1] = [x0;y0]
    value_xy[:,2] = [X[1];X[2]]
    value_phi_BFGS = zeros(nbIter+1)
    value_phi_BFGS[1] = phi(Xprev[1],Xprev[2])
    value_phi_BFGS[2] = phi(X[1],X[2])
    for k=2:nbIter
```

```
        # update A
        dx, dy = X-Xprev, dphi(X[1],X[2])-dphi(Xprev[1],Xprev[2])
        u, v = dx, invA*dy
        a, b = dot(dx,dy), dot((invA*dy),dy)
        invA .+= (a+b)/a^2*u*u' - 1/a*(u*v' + v*u')
        # testing
        #   A .+= dy*dy'/a - (A*dx)*(A*dx)'/(A*dot(dx,dx))
        #   println(A*invA)
        # update X
        Xnew = X - invA*dphis(X[1],X[2])
        Xprev .= X
        X .= Xnew
        value_xy[:,k+1] = [X[1];X[2]]
        value_phi_BFGS[k+1] = phi(X[1],X[2])
    end
    # output
    return value_xy,value_phi_BFGS
end

function Gradient_descent(nbIter,x0,y0,phi,dphi,lambda)
    """
       Apply the gradient descent method to find minimum of phi:R^2->R
        X_k+1 = X_k - lambda dphi(X_k)
    """
    # init
    x,y = x0,y0
    value_xy = zeros(2,nbIter+1)
    value_xy[:,1] = [x0;y0]
    value_xy[:,2] = [x;y]
    value_phi_GD = zeros(nbIter+1)
    value_phi_GD[1] = phi(x,y)
    # loop
    for k=1:nbIter
        x,y = [x;y] - lambda*dphi(x,y)
        value_xy[:,k+1] = [x;y]
        value_phi_GD[k+1] = phi(x,y)
    end
    # output
    return value_xy,value_phi_GD
end

function Gradient_descent_adapt(nbIter,x0,y0,phi,dphi,lambda0)
    """
       Apply the gradient descent method to find minimum of phi:R^2->R
```

```
89          X_k+1 = X_k - lambda dphi(X_k)
90         with lambda that 'adapts' if too large
91      """
92      # init
93      x,y,lambda = x0,y0,lambda0
94      value_xy = zeros(2,nbIter+1)
95      value_xy[:,1] = [x0;y0]
96      value_xy[:,2] = [x;y]
97      value_phi_GD_adapt = zeros(nbIter+1)
98      value_phi_GD_adapt[1] = phi(x,y)
99      value_lambda_adapt = zeros(nbIter+1)
100     value_lambda_adapt[1] = lambda
101     # loop
102     for k=1:nbIter
103         # update
104         x_tp,y_tp = [x;y] - lambda*dphi(x,y)
105         phi_tp = phi(x_tp,y_tp)
106         if (phi_tp<value_phi_GD_adapt[k])
107             # OK! we go down
108             x,y = x_tp,y_tp
109             value_phi_GD_adapt[k+1] = phi_tp
110         else
111             # bad! change lambda
112             lambda = .8*lambda
113             value_phi_GD_adapt[k+1] = value_phi_GD_adapt[k]
114         end
115         # save
116         value_xy[:,k+1] = [x;y]
117         value_lambda_adapt[k+1] = lambda
118     end
119     # output
120     return value_xy,value_phi_GD_adapt,value_lambda_adapt
121 end
```