UNIVERSIDAD SIMÓN BOLÍVAR
CO5412: Optimización No Lineal I
Enero-Marzo 2011
Enrique Areyán

Tarea 3
Sartenejas, 23 de Febrero de 2011

3 El código que fue empleado en esta tarea es el siguiente:

```
classdef Base
    properties
        eta = 10^-4;
        ro  = 1/2;
        statfilename = '';
        xmin;
    end
    methods(Static)
        function [ret] = norma(x)
            ret = sqrt(x(1)*x(1)+x(2)*x(2));
        end
    end
    methods(Abstract)
        [ret] = f(obj,arg)
        [fd_x,fd_y] = grad_f(obj,arg)
        [d_x,d_y] = d(obj,arg)
    end
    methods
        function obj=Base(args)
        end
        function [lambda,k] = backtracking(obj,xk)
            eta_local = obj.eta;
            ro_local = obj.ro;
            lambda = 1;
            parar = false;
            k=0;
            while ~parar
                [d1,d2] = obj.d(xk);
                d = [d1;d2];
                [g1,g2] = obj.grad_f(xk);
                g = [g1;g2];
                %condicion de armijo
                if(obj.f(xk + lambda * d) > obj.f(xk)+eta_local*lambda*g'*d)
                    lambda = lambda*ro_local;
                    xk = xk+lambda*d;
                    k=k+1;
```

```
                else
                    parar = true;
                end
            end
        end
    end
end

classdef Newton < Base
    properties
    end
    methods(Abstract)
        [ret] = hessiano(x)
    end
    methods
        function obj = Newton(arg)
            obj = obj@Base(arg);
        end
        function [d_x,d_y] = d(obj,arg)
            [fd_x,fd_y] = obj.grad_f(arg);
            gradf = [fd_x;fd_y];
            H = obj.hessiano(arg);
            %La direccion de Newton es el hessiando inverso por el
            %gradiente
            d = H \ gradf;
            %Esto es equivalente a multiplicar por -1 a la direccion
            d_x= -1*d(1);
            d_y= -1*d(2);

        end
        function [ret]=metodonewton(obj,xk)
            parar = false;
            k=0;
            fid = fopen(strcat('stats',obj.statfilename,'.txt'),'w');
            fprintf(fid,'k\tkx\t||xk-xmin||\t||grad_f(xk)||\n');

            fid_tex = fopen(strcat('stats',obj.statfilename,'.tex'),'w');
            fprintf(fid_tex,'\\underline{Funcion:} $%s$ \\\\ \nParametros: $\\eta = %f; \\rho = %
            fprintf(fid_tex,'\\begin{tabular}{|c|c|c|c|}\\hline\n\t$k$&$x_k$&$||x_k-x_*||$&$||gra

            while ~parar
                [g1,g2] = obj.grad_f(xk);
                normagrad = obj.norma([g1;g2]);
                fprintf(fid,'%d\t(%f,%f)\t%f\t%f\n',k,xk(1),xk(2),obj.norma(xk-obj.xmin),normagra
                fprintf(fid_tex,'\n\t%d&(%f,%f)&%f&%f\\\\ \\hline',k,xk(1),xk(2),obj.norma(xk-obj
                if normagrad < 10^-5 %condicion de terminacion del algoritmos
                    parar = true;
                else
                    [d1,d2] = obj.d(xk);
                    d = [d1;d2];
```

```
                    xk = xk + d;
                end
                k = k+1;
                if(k>1000)
                    parar = true;
                end
            end
            ret = xk;
            fclose(fid);

            fprintf(fid_tex,'\n\\end{tabular}\\\\\\\\\');
            fclose(fid_tex);

        end
    end
end
classdef NewtonModificado < Newton
    properties
    end
    methods
        function obj = NewtonModificado(arg)
            obj = obj@Newton(arg);
        end
        function [d_x,d_y] = d(obj,arg)
            [fd_x,fd_y] = obj.grad_f(arg);
            gradf = [fd_x;fd_y];
            %Obtener el hessiano de la funcion
            H = obj.hessiano(arg);
            autoval = eig(H);
            cantautoval = length(autoval);
            minautoval = bitmax;
            %Determinar el minimo autovalor del hessiano en el punto
            for i=1:cantautoval
                if minautoval > autoval(i)
                    minautoval = autoval(i);
                end
            end
            %Si el minimo autovalor es mayor que cero, la matriz es P.D.
            if minautoval>0
                h = H;
            else
            %Si hay algun autovalor igual o menor que cero, la matriz no es
            %P.D. y se debe ajustar la matriz para convertirla en P.D.
                miu = abs(minautoval)+10^-2;
                h = H + miu*eye(cantautoval);
            end
            d = h \ gradf;
            d_x= -1*d(1);
            d_y= -1*d(2);
        end
```

```
        function [ret]=metodonewtonmodificado(obj,xk)
            parar = false;
            k=0;
            fid = fopen(strcat('stats',obj.statfilename,'.txt'),'w');
            fprintf(fid,'k\tkx\t||xk-xmin||\t||grad_f(xk)||\n');

            fid_tex = fopen(strcat('stats',obj.statfilename,'.tex'),'w');
            fprintf(fid_tex,'\\underline{Funcion:} $%s$ \\\\ \nParametros: $\\eta = %f; \\rho = %
            fprintf(fid_tex,'\\begin{tabular}{|c|c|c|c|c|}\\hline\n\t$k$&$x_k$&$||x_k-x_*||$&$||g

            recortes = 0;
            while ~parar
                [g1,g2] = obj.grad_f(xk);
                normagrad = obj.norma([g1;g2]);
                fprintf(fid,'%d\t(%f,%f)\t%f\t%f\t%f\n',k,xk(1),xk(2),obj.norma(xk-obj.xmin),norm
                fprintf(fid_tex,'\n\t%d&(%f,%f)&%f&%f&%d\\\\ \\hline',k,xk(1),xk(2),obj.norma(xk-

                if normagrad < 10^-5 %condicion de terminacion del algoritmos
                    parar = true;
                else
                    [d1,d2] = obj.d(xk);
                    d = [d1;d2];
                    %Usando BLI
                    [lambda,recortes] = obj.backtracking(xk);
                    xk = xk + lambda*d;
                end
                k = k+1;
                if(k>1000)
                    parar = true;
                end
            end
            ret = xk;
            fclose(fid);

            fprintf(fid_tex,'\n\\end{tabular}\\\\\\\\');
            fclose(fid_tex);
        end
    end
end
classdef BFGS < Newton
    properties
        H = [1,0;0,1];
    end
    methods
        function dummy=hessiano(arg) end

        function obj = BFGS(arg)
            obj = obj@Newton(arg);
        end
        function [d_x,d_y] = d(obj,arg)
```

```
            [g1,g2] = obj.grad_f(arg);
            g = [g1;g2];
            d = -1* (obj.H \ g);
            d_x = d(1);
            d_y = d(2);
        end
        function [ret] = metodobfgs(obj,xk)
            parar = false;
            k=0;
            fid = fopen(strcat('stats',obj.statfilename,'.txt'),'w');
            fprintf(fid,'k\tkx\t||xk-xmin||\t||grad_f(xk)||\tRecortes\n');

            fid_tex = fopen(strcat('stats',obj.statfilename,'.tex'),'w');
            fprintf(fid_tex,'\\underline{Funcion:} $%s$ \\\\ \nParametros: $\\eta = %f; \\rho = %
            fprintf(fid_tex,'\\begin{tabular}{|c|c|c|c|c|}\\hline\n\t$k$&$x_k$&$||x_k-x_*||$&$||g

            recortes = 0;
            while ~parar
                [g1,g2] = obj.grad_f(xk);
                g = [g1;g2];
                normagrad = obj.norma([g1;g2]);
                fprintf(fid,'%d\t(%f,%f)\t%f\t%f\t%f\n',k,xk(1),xk(2),obj.norma(xk-obj.xmin),norm
                fprintf(fid_tex,'\n\t%d&(%f,%f)&%f&%f&%d\\\\ \\hline',k,xk(1),xk(2),obj.norma(xk-
                if normagrad < 10^-5 %condicion de terminacion del algoritmos
                    parar = true;
                else
                    [dx,dy] = obj.d(xk);
                    dk = [dx;dy];
                    [lambda,recortes]  = obj.backtracking(xk); % = 1; %
                    xk = xk + lambda * dk;
                    [gx,gy] = obj.grad_f(xk);
                    g_1 = [gx;gy];
                    y = g_1 - g;
                    obj.H = obj.H + ((y*y')/(y'*y)) - ((obj.H*dk)*(obj.H*dk)'/(dk'*obj.H*dk));
                end
                k = k+1;
                if(k>1000)
                    parar = true;
                end
            end
            ret = xk;
            fclose(fid);
            fprintf(fid_tex,'\n\\end{tabular}\\\\\\\\\');
            fclose(fid_tex);
        end
    end
end
```

Respecto a las funciones:

1.) Respecto a la función 1 existe un mínimo global debido a que f_1 es estrictamente convexa ya que su hessiano es positivo definido y se cumple que $\nabla f(0,0) = \vec{0}$. El mínimo global es $(0,0)$.

2.) Respecto a la función 2, podemos decir que por la proposición 4.8 se concluye que la función es convexa. (El determinante del hessiano queda así: $(12 * (x_1 - 2)^2 + 2) * 8 - 16$ y la única forma que esta sea cero es cuando $x_1 = 2$. De cualquier otra forma todos los subdeterminantes son no negativos. Por lo tanto el hessiano es S.P.D). Luego, por la proposición 4.9, todo punto estacionario es un mínimo global. El mínimo global es $(2,1)$.

3.) Respecto a la función 3, se puede decir que esta no es convexa ya que el hessiano no es S.D.P $(det(\nabla^2 f(x,y)) = 24x^2 - 1; det(\nabla^2 f(0,0)) = -1)$.

La siguiente tabla indica, por función y por método, si el mismo convergió o no. Si el método convergió indica el número de iteraciones totales, el número promedio de recortes y el error.

Algo.	**Convergió?**	**# Iter.**	**# Recort. Prom.**	$\|\|\nabla f(x_k)\|\|$	x_k	Min?
f_1, punto inicial $(1,1)^t$						
Newton	Si	1	n/a	0	(0,0)	Global
Newton Mod.	Si	1	0	0	(0,0)	Global
BGFS	Si	29	2	$9 * 10^{-6}$	$(-10^{-6}, -2 * 10^{-6})$	Global
f_2, punto inicial $(3,0)^t$						
Newton	Si	11	n/a	$6 * 10^{-6}$	(2.011561,1.005781)	Global
Newton Mod.	Si	11	0	$6 * 10^{-6}$	(2.011561,1.005781)	Global
BGFS	Si	25	3	$4 * 10^{-6}$	(1.997240,0.998621)	Global
f_2, punto inicial $(0,10)^t$						
Newton	Si	13	-	$4 * 10^{-6}$	(1.989724,0.994862)	Global.
Newton Mod.	Si	13	0	$4 * 10^{-6}$	(1.989724,0.994862)	Global.
BGFS	Si	27	3	$2 * 10^{-6}$	(1.998472,0.999236)	Global.
f_3, punto inicial $(0,0)^t$						
Newton	No	-	-	-	-	-
Newton Mod.	Si	14	0	0	(0.695884,-1.347942)	Pto.Est.
BGFS	Si	12	2	$6 * 10^{-6}$	(0.695883,-1.347942)	Pto.Est.

Nota: Tanto en BFGS como en Newton modificado se utilizó búsqueda lineal regresiva (como la implementada en la tarea anterior).
Nota 2: Para BFGS todas las direcciones generadas fueron de descenso.

Análisis de los resultados

En concordancia con la pregunta teórica 2, para la función cuadrática f_1, el algoritmo de Newton y Newton Modificado consigue la solución en una sola iteración. Por otra parte, para las demás funciones que no son cuadráticas, se tiene que los algoritmos tardan más de 1 iteraciones en acercarse a la solución.
En general, cuando el algoritmo de Newton converge lo hace más rápido que el BFGS. Sin embargo, el algoritmo de Newton no siempre converge mientras que BFGS si lo hace.