Programação Não Linear : Metodos de Otimização

Jorge Hans Alayo Matlab

6 de Maio de 2013

1 Ejercicio I

Se utilizo Matlab para la implementación de los algoritmos de Sección Aurea, Newton y Bisección. El ejercicio pide utilizar los programas desenvolvidos para encontrar mínimo de la siguiente función:

$$\text{Min } f(\lambda) = e^{-\lambda} + \lambda^2$$

Los algoritmos fueron programados de forma general, para aplicarlos a otros problemas solo se necesita cambiar las funciones de cada código programado respectivamente.

1.1 El algoritmo de sección aurea

El algoritmo desenvolvido se presenta en el apéndice A. La primera parte del algoritmo inicializa los datos del problema y escoje el intervalo de busqueda inicial. Luego, se empieza con las iteraciones que encuentran los nuevos intervalos. El algoritmo converge en $k = 31$ interaciones y el valor óptimo encontrado fue de $\lambda^* = 0,3517$.

1.2 El algoritmo de Newton

El algoritmo desenvolvido se presenta en el apéndice B. El algoritmo apenas necesita un punto inicial y las dos primeras derivadas de la función. El algoritmo converge en $k = 4$ iteraciones y el valor óptimo encontrado fue de $\lambda^* = 0,3517$.

1.3 El algoritmo de bisseción

El algoritmo desenvolvido se presenta en el apéndice C. El algoritmo solo necesta de la primera derivada de la función a minimizar, y después se seleciona los nuevos intervalos a partir de la comparación del signo de la derivada. El algoritmo converge en $k = 21$ iteraciones y el valor óptimo encontrado fue de $\lambda^* = 0,3517$.

2 Ejercício II

Se pide implementar los algoritmos de Gradiente Óptimo, Newton, y Metodo Híbrido. Se debe aplicar los algoritmos desenvolvidos en el siguiente problema de optimización:

$$\text{Min } F(x) = 100 \cdot (x_2 - 3 \cdot x_1^2)^5 + (1 - x_1)^2$$

2.1 El algoritmo de Gradiente Óptimo

El algoritmo se presenta en el apéndice D. En la primera parte del algoritmo se calcula la dirección de mejora con el gradiente de la función, luego se usa el método de bisección para encontrar el tamano de paso óptimo. El algoritmo no converge para el problema.

2.2 El algoritmo de Newton

El algoritmo se presenta en el apéndice D. El algoritmo de Newton apenas necesita del gradiente y el Hessiano de la función a minimizar. El algoritmo converge en $k = 24$ iteraciones y el valor óptimo encontrado fue de $x_1 = 1$ y $x_2 = 2,998$.

2.3 Ejercício III

Se pide utilizar el metodo de penalidades para resolver el siguiente problema de optimización:

$$\begin{array}{ll} \text{Minimizar} & (x_1 - 1)^2 + (x_2 - 2)^2 \\ \text{Sujeto a:} & x_2 - x_1 = 1 \\ & x_1 + x_2 \leq 2 \\ & x \geq 0 \end{array}$$

El problema se puede escribir como el siguiente problema irrestricto:

$$\text{Minimizar } (x_1 - 1)^2 + (x_2 - 2)^2 + \\ \mu \cdot \{[\text{Max}(0, x_1 + x_2 - 2)]^2 + (x_2 - x_1 - 1)^2 + [\text{Max}(0, -x_1)]^2 + [\text{Max}(0, -x_2)]^2\}$$

El gradiente y Hessiano de la función son los siguiente utilizando las variables binarias d_1, d_2, d_3 para representar las condiciones de desigualdad:

$$\nabla f = \begin{bmatrix} 2 \cdot (x_1 - 1) + \mu\,(2 \cdot d_1 \cdot (x_1 + x_2 - 2) - 2 \cdot (x_2 - x_1 - 1) + 2 \cdot d_2 \cdot x_1) \\ 2 \cdot (x_2 - 2) + \mu\,(2 \cdot d_1 \cdot (x_1 + x_2 - 2) + 2 \cdot (x_2 - x_1 - 1) + 2 \cdot d_3 \cdot x_2) \end{bmatrix}$$

$$H_f = \begin{bmatrix} 2 + \mu\,(2 \cdot d_1 + 2 + 2 \cdot d_2) & \mu(2d_1 - 2) \\ \mu(2d_1 - 2) & 2 + \mu\,(2 \cdot d_1 + 2 + 2 \cdot d_3) \end{bmatrix}$$

Se utilizó el método de Newton y un parametro de $\mu = 1000$. El algoritmo se presenta en el apéndice F. El algoritmo converge en $k = 4$ iteraciones y se encontro que $x_1 = 0,5002$ y $x_2 = 1,5002$.

A Algoritmo de Sección Aurea

```
%Algoritmo seccion aurea
% Min F(x),    a<x<b

error=1e-5;
a=-10;
b=10;
alpha=(-1+sqrt(5))/2

lambda=a+(1-alpha)*(b-a);
mu=a+alpha*(b-a);

theta_lambda=exp(-lambda)+lambda^2; %F(lambda)
theta_mu=exp(-mu)+mu^2;             %F (mu)

k=0;
while(1)
   if (b-a)< error
       break
   end

   if theta_lambda>theta_mu
       a=lambda;
       b=b;
       lambda=mu;
       mu=a+alpha*(b-a);
    else
       a=a;
       b=mu;
       mu=lambda;
       lambda=a+(1-alpha)*(b-a);

   end

    theta_lambda=exp(-lambda)+lambda^2; %F(lambda)
    theta_mu=exp(-mu)+mu^2;             %F (mu)
    k=k+1;

end
```

B Algoritmo de Newton Unidimensional

```
%Metodo de Newton
% Min F(X)    a<x<b

error=1e-5;
x=10;

while(1)
    dF= -exp(-x)+2*x;       %dF/dx
    if(abs(dF)<error)
        break
    end
    d2F=exp(-x)+2;          %d^2F/dx^2
    x=x-dF/d2F;
end
```

C Algoritmo de Bisección

```
%Metodo de Biseccion
% Min F(x)    a<x<b
error=1e-5;
k=0;
a=-10;
b=10;
N=round(log(error/(b-a))/log(0.5));
```

```
while(1)
    lambda=(b+a)/2;
    dF= -exp(-lambda)+2*lambda; %dF/dx(lambda)
    if dF==0
        break
    end

    if (dF>0)
        a=a;
        b=lambda;
    end
    if (dF<0)
        a=lambda;
        b=b;
    end

    if (k==N)
        break
    end
    k=k+1;
    end
```

D Algoritmo de Gradiente Óptimo

```
%Gradiente optimo
% Min f(x)  x pertenece R^n
clear
error=1e-6;
x1=1;
x2=1;
k=0;
kmax=50;
  while(1)

          grad_F=[-500*((x2-3*x1^2)^4)*(6*x1)-2*(1-x1)
              500*(x2-3*x1^2)^4];
          if (norm(grad_F)<error)
              break
          end
          d=-grad_F;
```

```
        alpha=0;
        beta=10;
        N=round(log(error/(beta-alpha))/log(0.5));
        k1=0;
        while(1)
            lambda=(beta+alpha)/2;
            u1=x1+lambda*d(1);
            u2=x2+lambda*d(2);
            grad=[-500*((u2-3*u1^2)^4)*(6*u1)-2*(1-u1)
                    500*(u2-3*u1^2)^4];

            dF=grad'*d;
            if dF==0
                break
            end
            if (dF>0)
                alpha=alpha;
                beta=lambda;
            end
            if (dF<0)
                alpha=lambda;
                beta=beta;
            end
            if (k1==N)
                break
            end
            k1=k1+1;
        end
        alpha;
        x=[x1;x2];
        x=x + alpha*d;
        x1=x(1);
        x2=x(2);

        k=k+1;
        if k>kmax
            break
        end
end
```

E Algoritmo de Newton

```
% Metodo de Newton
% Min F(x)
error=1e-7;
x1=1;
x2=1;
k=0;
while(1)
        grad_F=[2*(1-x1)+100*5*(x2-3*x1^2)^4*(-6*x1)
            100*5*(x2-3*x1^2)^4];
        if (max(abs(grad_F))<error)
            break
        end
        H_f=[-2+100*20*(x2-3*x1^2)^3*(6*x1)^2+...
            100*5*(x2-3*x1^2)^4*(-6) 100*20*(x2-3*x1^2)^3*(-6*x1);
            100*20*(x2-3*x1^2)^3*(-6*x1) 100*20*(x2-3*x1^2)^3];

        x=[x1;x2];

        x=x-inv(H_f)*grad_F;

        x1=x(1);
        x2=x(2);
        k=k+1;
 end
```

F Algoritmo de Penalidades

```
% Metodo de Penalidades - Newton
error=1e-7
x1=0;
x2=0;
k=0;
u=1000;
while(1)
    d1=1;
    d2=1;
    d3=1;
    if x1+x2<2
       d1=0;
    end
    if -x1<0
        d2=0;
    end
    if -x2<0
        d3=0;
    end
       grad_F=[2*(x1-1)+u*(2*d1*(x1+x2-2)-2*(x2-x1-1)+2*x1*d2)
               2*(x2-2)+u*(2*d1*(x1+x2-2)+2*(x2-x1-1)+2*x2*d3)];
       if (max(abs(grad_F))<error)
           break
       end
       H_f=[2+u*(2*d1+2+2*d2) u*(2*d1-2)
            u*(2*d1-2) 2+u*(2*d1+2+2*d3)];
       x=[x1;x2];
       x=x-inv(H_f)*grad_F;
       x1=x(1);
       x2=x(2);
       k=k+1;
end
```