Ingeniería Matemática
FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

MA5701 Optimización no Lineal

Informe de la Tarea 3

Desarrollo y resultados

Autor:	Manuel Torres.
Profesor:	Jorge Amaya.
Auxiliar:	Aldo Gutiérrez.
Ayudantes:	Carolina Chiu
	Mariano Vazquez

Índice general

Capítulo 1

Instrucciones

1.- Preliminares

1A.- Objetivo

El objetivo de esta tarea es implementar un código computacional que haga operacional el *método de direcciones admisibles (Zountendijk)*, para resolver el problema de optimización no lineal[1]

$$
\begin{array}{ll}
(P) & \text{mín } f(x) \\
 & Ax \leq b \\
 & Ex = e.
\end{array}
$$

1B.- Problema

El algoritmo a implementar es el siguiente:

(0) Sean $\varepsilon > 0$, $k = 0$, $x_0 \in \mathbb{R}^n$ tal que $Ax_0 \leq b$, $Ex_0 = e$.

(1) Sea la descomposición

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

tal que $A_1 x_k = b_1$, $A_2 x_k < b_2$.

(2) Resolver el problema lineal

$$
(\mathcal{D}_k) \left\{ \begin{array}{ll}
\text{mín} & \nabla f(x_k)^T d \\
\text{s.a.} & A_1 d \leq 0 \\
 & Ed = 0 \\
 & -1 \leq d_j \leq 1, \quad j = 1, \ldots, n.
\end{array} \right.
$$

y sea d_k solución de $(\mathcal{D}_k)$.

- Si $\|\nabla f(x_k)^T d_k\| < \varepsilon$, entonces parar.
- En caso contrario, ir a (3).

(3) Determinar el paso, resolviendo aproximadamente el problema de minimización unidimensional

$$
(L) \left\{ \begin{array}{ll}
\text{mín} & f(x_k + \lambda d_k) \\
\text{s.a} & \lambda \in [0, \tilde{\lambda}_k]
\end{array} \right.
$$

[1] Aclaración: Cuando se habla de problema de optimización no lineal no quiere decir que necesariamente se excluyen los problemas lineales.

medinate el *método de Armijo*. Se usa

$$\tilde{\lambda}_k = \text{mín}\left\{ \frac{(b_2 - A_2 x_k)_i}{(A_2 d_k)_i} / (A_2 d_k)_i > 0 \right\},$$

y se considera $\tilde{\lambda}_k = +\infty$ cuando $(A_2 d_k)_i \leq 0$ para todo i.

Sea λ_k solución del subproblema (L). Hacer:

$$x_{k+1} = x_k + \lambda_k d_k, \quad k \leftarrow k+1$$

e ir a (1).

Luego se propone aplicar el algoritmo para testear los siguientes problemas de optimización:

1. Comenzando del punto $(0, 2)$,

$$(P_1) \begin{cases} \text{mín} & 8(x_1 - 6)^2 + (x_2 - 2)^4 & & \\ \text{s.a.} & -x_1 + 2x_2 & \leq & 4 \\ & 3x_1 + 2x_2 & \leq & 12 \\ & x_1, x_2 & \geq & 0 \end{cases}$$

2. Comenzando del punto $(2, 2, 3, 2)$,

$$(P_2) \begin{cases} \text{mín} & x_1^4 - 2x_2^2 + 10x_1x_2^2 + x_4^2 & & \\ \text{s.a.} & x_1 + x_2 - x_3 & = & 1 \\ & x_1 & = & 4 \\ & x_1 + x_4 & = & 0 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{cases}$$

2.- Comentarios sobre el modelo

Se desea codificar en Python el algoritmo del *método de direcciones admisibles* descrito antes, para ello se utilizarán métodos auxiliares para los pasos (1), (2) y (3). En el anexo puede encontrar la codificación del *método de direcciones admisibles* y de los métodos auxiliares empleados.

Capítulo 2

Respuestas

1.- Preguntas de la tarea

A continuación se testea el método de direcciones admisibles implementado en A.4 con dos problemas. Los resultados obtenidos son los presentados a continuación.

P_1.- Comenzando del punto $(0, 2)$,

$$(P_1)\left\{\begin{array}{llrcl} \text{mín} & 8(x_1-6)^2+(x_2-2)^4 & & & \\ \text{s.a.} & & -x_1+2x_2 & \leq & 4 \\ & & 3x_1+2x_2 & \leq & 12 \\ & & x_1, x_2 & \geq & 0 \end{array}\right.$$

Resultados: Para P_1 se alcanzó el óptimo en 3 iteraciones del método. El valor óptimo es $f(x^*) = 46{,}9041227202609$ con solución $x^* = (3{,}85346355 \quad 0{,}21980435)$.

P_2.- Comenzando del punto $(2, 2, 3, 2)$,

$$(P_2)\left\{\begin{array}{llrcl} \text{mín} & x_1^4-2x_2^2+10x_1x_2^2+x_4^2 & & & \\ \text{s.a.} & & x_1+x_2-x_3 & = & 1 \\ & & x_1 & = & 4 \\ & & x_1+x_4 & = & 0 \\ & & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}\right.$$

Resultados: Para P_2 se alcanzó el óptimo en 3 iteraciones del método. El valor óptimo es $f(x^*) = 13{,}047127462172996$ con solución $x^* = (0{,}5358624 \quad 0{,}5358624 \quad 0{,}07172479 \quad 3{,}4641376)$.

2.- Conclusión

Como se ha visto en los resultados presentados antes, el método de direcciones admisibles consigue resolver los problemas (P_1) y (P_2) en muy pocos pasos.

Apéndice A

Anexo

1.- Codificación de los métodos auxiliares

El primer método auxiliar recibe un sistema de desigualdades de la forma $Ax \leq b$ y entrega una descomposición

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

tal que $A_1 x_k = b_1$, $A_2 x_k < b_2$.

```python
def desigualdades_activas_inactivas(A,b,x):
    """
    -Input: Sistema de inecuaciones Ax <= b.
    -Output: A1,A2,b1,b2 typo np.array.
    -Descripcion: Particiona el sistema de inecuaciones Ax <= b en las desigualdades
    activas y las desigualdades inactivas.
    """
    # Declara los arreglos para guardar la particion
    A1,A2,b1,b2 = [],[],[],[]
    for i in range(len(A)):
        # Igualdades alcanzadas
        if np.isclose(A[i]@x, b[i]):
            A1.append(A[i].tolist())
            b1.append(b[i].tolist())
        # Desigualdades que no alcanzan igualdad
        elif A[i]@x<b[i]:
            A2.append(A[i].tolist())
            b2.append(b[i].tolist())
    # Output en formato de arreglos
    #return = A1,A2,b1,b2
    # Output en formato de arreglos de numpy (se transforman de array a np.array)
    return np.array(A1), np.array(A2), np.array(b1), np.array(b2)
```

Listing A.1: Método auxiliar (1)

El siguiente método resuelve el problema de optimización lineal

$$(\mathcal{D}_k) \left\{ \begin{array}{cl} \text{mín} & \nabla f(x_k)^T d \\ \text{s.a.} & A_1 d \leq 0 \\ & Ed = 0 \\ & -1 \leq d_j \leq 1, \quad j = 1, \ldots, n. \end{array} \right.$$

y sea d_k solución de $(\mathcal{D}_k)$.

```python
def problema_d_k(f,xk,A1,E):
    """
    -Input:
    -Output:
    -Descripcion:
    """
    # Calcula el gradiente de f utilizando la libreria numdifftools
    gradiente        = nd.Gradient(f)
```

```python
    # Funcion objetivo: Se define a partir del vector gradiente
    funcion_objetivo = lambda d: gradiente(xk)@d
    # Cotas para xk
    cotas            = tuple([i for i in zip([-1 for _ in range(len(xk))],
                                             [1 for _ in range(len(xk))])])
    restricciones    = []
    restricciones_E  = []
    # Si no hay restricciones del estilo Ex = 0
    if E is None:
        for i in range(len(A1)):
            restricciones.append(LinearConstraint(A1,[-np.inf]*A1.shape[0],[0]*A1.shape[0]))
    # Si hay restricciones del estilo Ex = 0
    else:
        for i in range(len(E)):
            restricciones_E.append(0)
        cons.append(LinearConstraint(E,restricciones_E,restricciones_E))
    # Metodo optimizador: Se utiliza scipy.optimize.minimize
    resultado = minimize(funcion_objetivo,[1]*len(xk),method='trust-constr',bounds=cotas,
    constraints=restricciones)
    argmin    = resultado.x # Output
    return argmin
```

Listing A.2: Método auxiliar (2)

Y finalmente, el siguiente método determina el paso visto en la parte (3) del método de direcciones admisibles, resolviendo aproximadamente el problema de minimización unidimensional

$$(L)\begin{cases} \text{mín} & f(x_k+\lambda d_k) \\ \text{s.a} & \lambda \in [0,\tilde{\lambda}_k] \end{cases}$$

medinate el *método de Armijo*. Se usa

$$\tilde{\lambda}_k = \text{mín}\left\{\frac{(b_2 - A_2x_k)_i}{(A_2d_k)_i} / (A_2d_k)_i > 0\right\},$$

y se considera $\tilde{\lambda_k} = +\infty$ cuando $(A_2d_k)_i \leq 0$ para todo i.

```python
def metodo_de_armijo(f,gradiente,xk,dk,A2,b2,A,b):
    """
    -Input:
    -Output:
    -Descripcion:
    """
    sig = 0.6
    h   = 0.01
    m   = 1
    lambdas = []
    uwu     = []
    for i in range(b2.shape[0]):
        if np.all(A2@dk <= 0) == True:
            lambdas.append(np.inf)
        elif A2[i]@dk > 0:
            uwu.append((b2[i] - np.array(A2[i])@ xk)/(A2[i]@dk))
    lambdas.append(min(uwu))
    while h * m <= lambdas[0]:
        if f(xk) + sig*m*h*gradiente(xk) @ dk >= f(xk + m*h*dk):
            m+=1
        else:
            if np.all(A@(xk+h*(m-1)*dk) <= b) == True:
                return h*(m-1)
            else:
                m-=1
    else:
        return lambdas[0]
```

Listing A.3: Método auxiliar (3)

1A.- Codificación del método de direcciones admisibles

```
def metodo_direcciones_admisibles(eps,x0,f,A,b,E=None,e=None,max_cantidad_iteraciones=100):
    """
    -Input:
    -Output:
    -Descripcion:
    """
    # Contador de cantidad de iteraciones realizadas / conv = True mientras k no exceda
    # la cantidad maxima de iteraciones
    k, conv = 0, True
    # Punto inicial (viene propuesto un punto para comenzar junto con el problema)
    xk = x0
    "Iteraci n inicial"
    # Paso (1)
    A1, A2, b1, b2 = desigualdades_activas_inactivas(A, b, xk)
    # Paso (2)
    dk = problema_d_k(f, xk, A1, E)
    "Iteraciones hasta cumplir la condicion \|\nabla f(x_{k})^{T}d_{k}\|<\epsilon"
    while np.abs(nd.Gradient(f)(xk) @ dk) > eps:
        # Pasada una cantidad maxima de iteraciones se supondra que el problema
        # es irresoluble con el metodo de direcciones admisibles.
        if k > max_cantidad_iteraciones:
            print('Se agot  el m ximo de iteraciones ({})'.format(max_iter))
            conv = False
            break
        # Paso (3)
        tk = metodo_de_armijo(f, nd.Gradient(f), xk,dk, A2, b2, A, b)
        if tk == 0:break
        xk = xk + tk*dk
        # Aumenta el contador de iteraciones
        k = k+1
        # Paso (1)
        A1, A2, b1, b2 = desigualdades_activas_inactivas(A, b, xk)
        # Paso (2)
        dk = problema_d_k(f, xk, A1, E)
    output = xk
    return output
```

Listing A.4: Método de direcciones admisibles