Matemática: Una publicación del ICM – ESPOL
2016, Vol. 14, No. 1

MÉTODO DEL GRADIENTE DE MÁXIMO DESCENSO

Rodríguez Luis [1]

Resumen. *En esta contribución académica se expone el conocido método del gradiente de máximo descenso para encontrar un mínimo local de una función multivariada no lineal sin restricciones. Se destaca la importancia de conectar los conceptos matemáticos con la descripción algorítmica y su instrumentación computacional para la solución práctica de problemas. La instrumentación se la realizó en el lenguaje Python el cual provee soporte adecuado para manejo matemático simbólico, gráfico y numérico y está disponible como software libre.*
Palabras clave: Gradiente de máximo descenso. Mínimo de una función multivariada no lineal. Optimización numérica.

Abstract. *This academic contribution exposes the well known steepest descent gradient method used to find a local minimum of a nonlinear multivariate unconstrained function. It is emphasized the importance of connect mathematical concepts with the algorithmic description and its computational instrumentation for solving practical problems. The instrumentation was made in the Python language which provides adecuate support for symbolic, graphical and numerical mathematics and is available as free software.*
Keywords: Steepest descent gradient. Minimum of a non linear multivariate function. Numerical optimization.

Recibido: Abril 2016.
Aceptado: Abril 2016.

1. INTRODUCCIÓN

El método del gradiente de máximo descenso es un algoritmo iterativo para encontrar un mínimo local de una función multivariada no lineal sin restricciones, mediante aproximaciones sucesivas. La búsqueda de la solución sigue la dirección del gradiente descendente más pronunciado hasta llegar a su menor valor. El método funciona con la suposición que la función tiene forma convexa alrededor del mínimo y que la distancia de avance en cada paso es elegida apropiadamente.

Se revisan las definiciones básicas y se describe el algoritmo para su aplicación. La contribución relevante es la vinculación de la formulación con la instrumentación computacional, en este caso con el lenguaje Python por su simplicidad y facilidad para el manejo matemático simbólico. En este aspecto cabe resaltar la importancia de usar estos lenguajes computacionales modernos como una opción para que los investigadores puedan aplicar los métodos matemáticos de manera práctica y eficiente.

2. DEFINICIONES

Vector de n variables reales

$$\mathbf{v} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

Función de n variables reales

$$f: \mathbf{R}^n \rightarrow \mathbf{R}$$

Gradiente de f. Es el vector de las derivadas parciales de la función que se desea minimizar

$$\nabla \mathbf{f} = \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}_i} \right], \; i=1,2,\dots, n$$

Gradiente de f en un punto **v**. Es la dirección de mayor incremento de f en el punto **v**

$$\mathbf{c} = \nabla \mathbf{f}(\mathbf{v})$$

Magnitud del vector gradiente de f en un punto **v**. Es la mayor tasa de cambio del gradiente en el punto **v**

$$\| \mathbf{c} \| = \sqrt{\mathbf{c}^T \mathbf{c}}$$

Vector gradiente de f normalizado, en un punto **v**

$$\bar{\mathbf{c}} = \frac{\mathbf{c}}{\| \mathbf{c} \|}$$

Vector de la dirección de búsqueda descendente para obtener el mínimo de f en el punto **v**. Esta dirección debe ser la opuesta al gradiente de f en el punto **v**:

$$\mathbf{d} = -\bar{\mathbf{c}}$$

Longitud del paso de avance s para modificar el vector **v** en la dirección **d**:

Debe ser un valor escalar que garantice que $f(\mathbf{v} + s\mathbf{d}) < f(\mathbf{v})$.

La búsqueda del mínimo de la función f consiste en modificar el vector **v** agregando el vector s**d** repetidamente. En cada iteración se debe elegir el valor de s que optimice la búsqueda.

El método converge si se llega a un punto en el cual el gradiente ya no cambia significativamente. Este punto corresponde aproximadamente al mínimo de la función f.

[1] *Rodriguez Ojeda Luis., Profesor, Departamento de Matemáticas, Facultad de Ciencias Naturales y Matemáticas, ESPOL. (e_mail: lrodrig@espol.edu.ec).*

3. ALGORITMO DEL GRADIENTE DE MÁXIMO DESCENSO

Objetivo: Encontrar un mínimo local de una función multivariada f

1) Iniciar un conteo de iteraciones: k=0

Estimar un vector inicial para la solución: $\mathbf{v}^{(k)}$

Estimar un máximo de iteraciones m y un criterio de precisión ε

2) Calcular el vector gradiente de f evaluado en el punto $\mathbf{v}^{(k)}$

$$\mathbf{c}^{(k)} = \nabla \mathbf{f}(\mathbf{v}^{(k)})$$

3) Calcular la máxima tasa de cambio del gradiente de f en el punto $\mathbf{v}^{(k)}$:

$$\| \mathbf{c}^{(k)} \| = \sqrt{\mathbf{c}^{(k)^T} \mathbf{c}^{(k)}}$$

Si $\| \mathbf{c}^{(k)} \| < \varepsilon$ entonces

$\mathbf{v}^{(k)}$ es un punto mínimo de **f** con precisión ε

Finalizar (el método converge)

4) Calcular el vector normalizado de la dirección de búsqueda en el punto $\mathbf{v}^{(k)}$

$$\mathbf{d}^{(k)} = -\frac{\mathbf{c}^{(k)}}{\| \mathbf{c}^{(k)} \|}$$

5) Determinar el tamaño del paso de avance $s^{(k)}$ en la iteración k

6) Actualizar el vector de búsqueda

$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + \mathbf{s}^{(k)} \mathbf{d}^{(k)}$$

7) Actualizar el conteo de iteraciones

k = k + 1

Si k < m entonces

Regresar al paso 2)

Sinó

Finalizar (el método no converge)

3.1 Procedimiento para determinar el tamaño óptimo del paso de avance

Es necesario elegir el tamaño del paso para avanzar en la búsqueda de la solución. Un tamaño muy grande puede hacer que no se pueda localizar la solución. Un tamaño muy pequeño puede hacer que la búsqueda sea ineficiente. Es preferible que el tamaño del paso se pueda modificar en cada iteración.

Un método para optimizar el tamaño del paso de búsqueda $s^{(k)}$ consiste en elegir el valor de s tal que $\mathbf{f}(\mathbf{v}^{(k+1)})$ sea mínimo. Este valor de s puede encontrarse resolviendo en cada iteración k la ecuación:

$$\frac{\mathbf{df}(\mathbf{v}^{(k+1)})}{\mathbf{ds}} = \frac{\mathbf{df}(\mathbf{v}^{(k)} + \mathbf{sd}^{(k)})}{\mathbf{ds}} = 0$$

4. INSTRUMENTACIÓN COMPUTACIONAL DEL ALGORITMO DEL GRADIENTE DE MÁXIMO DESCENSO

Es conveniente instrumentar computacionalmente la formulación del algoritmo de máximo descenso para experimentar y faciltar su uso en la resolución de problemas. Se utilizó el lenguaje Python por las facilidades que ofrece para manejo matemático simbólico, numérico y gráfico.

4.1 Descripción y uso de las funciones de la instrumentación computacional

La instrumentación está encapsulada en un módulo denominado **gradiente** el cual contiene funciones de utilidad que se pueden llamar separadamente y también son los componentes de soporte para una función principal con el nombre **metodo_gradiente** la cual corresponde al lineamiento del algoritmo propuesto.

El código Python de las funciones del módulo **gradiente** está al final de esta contribución académica.

obtener_gradiente(f,v)

Entra

f: Función multivariada

v: Vector de variables definidas en forma simbólica

Sale

g: Vector gradiente con las derivadas parciales de f

evaluar_gradiente(g,v,u)

Entra **g:** Vector gradiente con las derivadas parciales de f

v: Vector de variables definidas en forma simbólica

u: Vector con valores escalares para las variables en **v**

Sale

c: Vector gradiente con los componentes evaluados en el punto **u**

magnitud_del_gradiente(c)

Entra

c: Vector gradiente evaluado en el punto **u**

Sale

norma: Máxima tasa de cambio del gradiente en el punto **u**

gradiente_normalizado(c)

Entra

c: Vector gradiente evaluado en el punto **u**

Sale

cn: Vector gradiente normalizado evaluado en el punto **u**

calcular_paso(f,g,v,u)

Entra

f: Función multivariada

g: Vector gradiente con las derivadas parciales de f
v: Vector de variables definidas en forma simbólica
u: Vector con valores escalares para las variables en **v**

Sale

s: Valor estimado para el tamaño del paso de avance

evaluar_solucion(f,v,u)
Entra

f: Función multivariada
v: Vector de variables definidas en forma simbólica
u: Vector con valores escalares para las variables en **v**

Sale

fm: Valor actual de la solución en el punto **u**

metodo_gradiente(f,v,u,e,m,imp=0)
Entra

f: Función multivariada
v: Vector de variables definidas en forma simbólica
u: Vector con valores escalares iniciales para las variables en **v**
e: Criterio de convergencia y precisión para el gradiente
m: Cantidad máxima permitida para las iteraciones
imp: Parámetro opcional. Si se lo omite no se mostrarán los resultados intermedios, pero si se le asigna algún valor mayor a cero, se mostrarán los valores calculados en cada iteración.

Sale

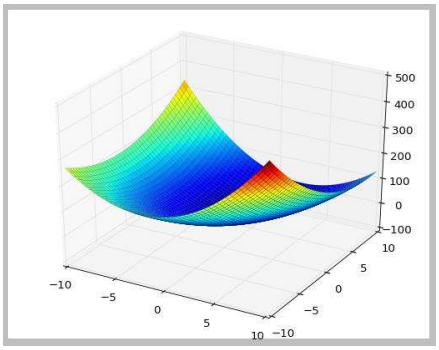
Si el método converge
uk: Vector solución calculado en la última iteración
fm: Valor de la función evaluada en la solución

Si el método no converge entrega un vector nulo

5. EJEMPLOS DE APLICACIÓN

Ejemplo 5.1 Calcular el mínimo de $f(x,y)=2x^2-xy+y^2-7y$. Graficar y mostrar las iteraciones y resultados

```
>>> from gradiente import*
>>> x,y=symbols('x,y')
>>> f=2*x**2-x*y+y**2-7*y
>>> plot3d(f,(x,-10,10),(y,-10,10))
```



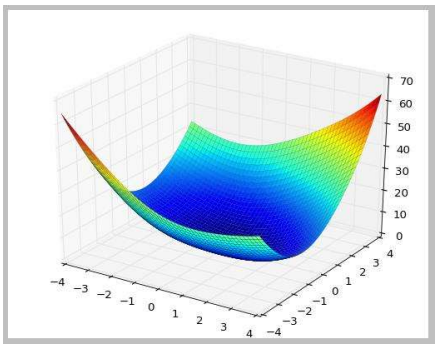

```
>>> v=[x,y]
>>> u=[0,0]
>>>
uk,fm=metodo_gradiente(f,v,u,0.01,2
0,1)
k=1  s=3.5 vector= [0.0, 3.5]
k=2  s=0.875 vector= [0.875, 3.5]
k=3   s=0.4375   vector=   [0.875,
3.9375]
k=4  s=0.109375  vector=  [0.984375,
3.9375]
k=5  s=0.0546875  vector=  [0.984375,
3.9921875]
k=6      s=0.013671875      vector=
[0.998046875, 3.9921875]
k=7   s=   0.0068359375      vector=
[0.998046875, 3.9990234375
>>> uk
[0.998046875, 3.9990234375]
>>> fm
-13.9999933242798
```

Ejemplo 5.2 Calcular el mínimo de $f(x,y)=2x^2-xy+y^2-7y$. Elegir otro vector inicial. Mostrar resultados.

```
>>> from gradiente import*
>>> x,y=symbols('x,y')
>>> v=[x,y]
>>> u=[5,5]
>>>
uk,fm=metodo_gradiente(f,v,u,0.01,2
0)
>>> uk
[1.001147654264687,
4.000286913566172]
>>> fm
-13.9999976127376
```

Ejemplo 5.3 Calcular el mínimo de $f(x,y)=x^2+2y^2+\cos(x+y+1)+xy$. Graficar y mostrar resultados

```
>>> from gradiente import*
>>> x,y=symbols('x,y')
>>> f=x**2+2*y**2+cos(x+y+1)+x*y
>>> plot3d(f,(x,-4,4),(y,-4,4))
```



```
>>> v=[x,y]
>>> u=[1,1]
>>>
uk,fm=metodo_gradiente(f,v,u,0.001,
20)
>>> uk
[0.42864627450129655,
0.14293529459696494]
>>> fm
0.285082064827950
```

Ejemplo 5.4 Calcular el mínimo de $f(x,y,z)=5(x-1)^2+3(y+2)^2+4(z+3)^2+xyz+1$. Mostrar resultados

```
>>> from gradiente import*
>>> x,y,z=symbols('x,y,z')
>>>                          f=5*(x-
1)**2+3*(y+2)**2+4*(z+3)**2+x*y*z+1
>>> v=[x,y,z]
>>> u=[0,0,0]
>>>
uk,fm=metodo_gradiente(f,v,u,0.01,2
0)
>>> uk
[0.4909226771078404,                -
1.7628499677299507,                 -
2.8919840590397894]
>>> fm
5.01397838490301
```

Ejemplo 5.5 Calcular el mínimo de $f(x_1,x_2,x_3,x_4)=5(x_1-1)^2+3(x_2-2)^2+4(x_3+3)^2+(x_4-1)^4-x_1x_2x_3x_4+5$.

Mostrar resultados

```
>>> from gradiente import*
>>>
x1,x2,x3,x4=symbols('x1,x2,x3,x4')
>>>              f=5*(x1-1)**2+3*(x2-
2)**2+4*(x3+3)**2+(x4-1)**4-
x1*x2*x3*x4+5
>>> v=[x1,x2,x3,x4]
>>> u=[0,0,0,0]
>>>
uk,fm=metodo_gradiente(f,v,u,0.01,2
0)
>>> uk
[1.1570991258533814,
2.141077839354268,
3.0738289454327443,
 -0.23935943405879354]
>>> fm
5.74146881516544
```

6. CONCLUSIONES

En esta contribución se ha tratado un conocido método matemático de optimización y se ha resaltado la vinculación entre los enunciados y las fórmulas matemáticas con su instrumentación computacional. Esta relación tiene mucha importancia pues permite utilizar el desarrollo matemático en la investigación y en la resolución práctica de problemas de aplicación.

El método también se puede usar para encontrar máximos locales de una función multivariada f aplicándolo a la función -f. Igualmente puede usarse para el caso básico del cálculo de mínimos o máximos locales de funciones univariadas.

La instrumentación computacional usa como soporte el lenguaje Python, el cual por ser software de uso libre no requiere licencia. Este lenguaje tiene características adecuadas para manejo matemático numérico, simbólico y gráfico siendo además muy simple de usar.

BIBLIOGRAFÍA BÁSICA

[1]. Using Gradient Descent for Optimization and Learning Nicolas Le Roux, 2009 http://www.gatsby.ucl.ac.uk/teaching/courses/ml2-2008/graddescent.pdf

[2]. Gradient Descent Kris Hauser, 2012 http://homes.soic.indiana.edu/classes/spring2012/csci/b553-hauserk/gradient_descent.pdf

[3]. Introducción a la optimización numérica http://rua.ua.es/dspace/bitstream/10045/16373/8/Microsoft%20Word%20-%208.%20INTRODUCCION%20A%20LA%20OPTIMIZACION%20NUMERICA-1.pdf

[4]. Análisis Numérico Básico Rodríguez Ojeda, Luis Libro digital disponible en la FCNM, ESPOL, 2014 http://www.fcnm.espol.edu.ec/publicaciones

[5]. Python Programación Rodríguez Ojeda, Luis Libro digital disponible en la FCNM, ESPOL, 2014 http://www.fcnm.espol.edu.ec/publicaciones

7. CÓDIGO DEL MÓDULO GRADIENTE EN LENGUAJE PYTHON

```
#Método del gradiente de máximo descenso
from sympy import*
from sympy.plotting import*
import numpy as np

def obtener_gradiente(f,v):
    n=len(v)
    g=[]
    for i in range(n):
        d=diff(f,v[i])
        g=g+[d]
    return g

def evaluar_gradiente(g,v,u):
    n=len(v)
    c=[]
    for i in range(n):
        t=g[i]
        for j in range(n):
            t=t.subs(v[j],u[j])
        c=c+[float(t)]
    return c

def magnitud_del_gradiente(c):
    norma=sqrt(np.dot(c,c))
    return norma

def gradiente_normalizado(c):
    norma=magnitud_del_gradiente(c)
    t=list(np.array(c)/norma)
    cn=[]
    for i in range(len(c)):
        cn=cn+[float(t[i])]
    return cn

def calcular_paso(f,g,v,u):
    c=evaluar_gradiente(g,v,u)
    cn=gradiente_normalizado(c)
    t=Symbol('t')
    xt=[]
    for i in range(len(v)):
        xt=xt+[float(u[i])-t*float(cn[i])]
    fs=f.subs(v[0],xt[0])
    for i in range(1,len(v)):
        fs=fs.subs(v[i],xt[i])
    df=diff(fs,t)
    ddf=diff(df,t)
    s=1
    for i in range(5):
        s=s-float(df.subs(t,s))/float(ddf.subs(t,s))
    return s
def evaluar_solucion(f,v,u):
    fm=f.subs(v[0],u[0])
    for i in range(1,len(v)):
        fm=fm.subs(v[i],u[i])
    return fm

def metodo_gradiente(f,v,u,e,m,imp=0):
    u0=u.copy()
    g=obtener_gradiente(f,v)
    for k in range(m):
        c=evaluar_gradiente(g,v,u0)
        norma=magnitud_del_gradiente(c)
        if norma<e:
            fm=evaluar_solucion(f,v,u0)
            return u0,fm
        s=calcular_paso(f,g,v,u0)
        cn=gradiente_normalizado(c)
        uk=[]
        for i in range(len(c)):
            uk=uk+[float(u0[i])-s*float(cn[i])]
        u0=uk.copy()
        if imp>0:
            print('k=',k+1,' s=',s,' vector=',u0)
    return [],None
```