Método del gradiente reducido generalizado

Penadillo Lazares Wenses Johan

UNIVERSIDAD NACIONAL DE INGENIERÍA

15 de noviembre de 2021

Índice

Introducción

Este método es una extensión del método del gradiente reducido.

$$\text{Minimize } f(\mathbf{X})$$

$$h_j(\mathbf{X}) \le 0, \quad j = 1, 2, \ldots, m$$

$$l_k(\mathbf{X}) = 0, \quad k = 1, 2, \ldots, l$$

$$x_i^{(l)} \le x_i \le x_i^{(u)}, \quad i = 1, 2, \ldots, n$$

Introducción

$$\text{Minimize } f(\mathbf{X})$$

$$g_j(\mathbf{X}) = 0, \quad j = 1, 2, \ldots, m + l$$
$$x_i^{(l)} \leq x_i \leq x_i^{(u)}, \quad i = 1, 2, \ldots, n + m$$

Donde el limite inferior y superior de las x_i sera 0 e ∞ respectivamente.

GRG

El método esta basado en la idea de eliminación de variables.

$$\mathbf{X} = \begin{Bmatrix} \mathbf{Y} \\ \mathbf{Z} \end{Bmatrix}$$

$$\mathbf{Y} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-l} \end{Bmatrix}$$

$$\mathbf{Z} = \begin{Bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{m+l} \end{Bmatrix}$$

Donde Y serán variables independientes y Z variables dependientes.

GRG

Considerando la primera derivada de la función objetivo y las restricciones.

$$df(\mathbf{X}) = \sum_{i=1}^{n-l} \frac{\partial f}{\partial y_i} dy_i + \sum_{i=1}^{m+l} \frac{\partial f}{\partial z_i} dz_i = \nabla_{\mathbf{Y}}^{\mathrm{T}} f \, d\mathbf{Y} + \nabla_{\mathbf{Z}}^{\mathrm{T}} f \, d\mathbf{Z}$$

$$dg_i(\mathbf{X}) = \sum_{j=1}^{n-l} \frac{\partial g_i}{\partial y_j} dy_j + \sum_{j=1}^{m+l} \frac{\partial g_i}{\partial z_j} dz_j$$

$$d\mathbf{g} = [C] \, d\mathbf{Y} + [D] \, d\mathbf{Z}$$

GRG

Donde:

$$\nabla_{\mathbf{Y}} f = \begin{Bmatrix} \dfrac{\partial f}{\partial y_1} \\ \dfrac{\partial f}{\partial y_2} \\ \vdots \\ \dfrac{\partial f}{\partial y_{n-l}} \end{Bmatrix} \qquad [C] = \begin{bmatrix} \dfrac{\partial g_1}{\partial y_1} & \cdots & \dfrac{\partial g_1}{\partial y_{n-l}} \\ \vdots & & \vdots \\ \dfrac{\partial g_{m+l}}{\partial y_1} & \cdots & \dfrac{\partial g_{m+l}}{\partial y_{n-l}} \end{bmatrix} \qquad d\mathbf{Y} = \begin{Bmatrix} dy_1 \\ dy_2 \\ \vdots \\ dy_{n-l} \end{Bmatrix}$$

$$\nabla_{\mathbf{Z}} f = \begin{Bmatrix} \dfrac{\partial f}{\partial z_1} \\ \dfrac{\partial f}{\partial z_2} \\ \vdots \\ \dfrac{\partial f}{\partial z_{m+l}} \end{Bmatrix} \qquad [D] = \begin{bmatrix} \dfrac{\partial g_1}{\partial z_1} & \cdots & \dfrac{\partial g_1}{\partial z_{m+l}} \\ \vdots & & \vdots \\ \dfrac{\partial g_{m+l}}{\partial z_1} & \cdots & \dfrac{\partial g_{m+l}}{\partial z_{m+l}} \end{bmatrix} \qquad d\mathbf{Z} = \begin{Bmatrix} dz_1 \\ dz_2 \\ \vdots \\ dz_{m+l} \end{Bmatrix}$$

GRG

La solución a la ecuación anterior seria:

$$d\mathbf{Z} = -[D]^{-1}[C]d\mathbf{Y}$$

Reemplazando dZ tenemos:

$$df(\mathbf{X}) = (\nabla_{\mathbf{Y}}^{\mathrm{T}} f - \nabla_{\mathbf{Z}}^{\mathrm{T}} f[D]^{-1}[C])d\mathbf{Y}$$

$$\frac{df}{d\mathbf{Y}}(\mathbf{X}) = \mathbf{G}_R$$

$$\mathbf{G}_R = \nabla_{\mathbf{Y}} f - ([D]^{-1}[C])^{\mathrm{T}} \nabla_{\mathbf{Z}} f$$

Donde G_R seria el gradiente reducido.

GRG

Se debe cumplir:

$$g(X) + dg(X) = 0$$

Reemplazando dg

$$dZ = [D]^{-1}(-g(X) - [C]dY)$$

Este valor sera usado para actualizar Z.

$$Z_{update} = Z_{current} + dZ$$

Ejemplo

$$\begin{aligned}
\text{minimize} \quad & x_1^2 + x_2^2 + x_3^2 + x_4^2 - 2x_1 - 3x_4 \\
\text{subject to} \quad & 2x_1 + x_2 + x_3 + 4x_4 = 7 \\
& x_1 + x_2 + 2x_3 + x_4 = 6 \\
& x_i \geqslant 0, \quad i = 1, 2, 3, 4.
\end{aligned}$$

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import *


def generalized_reduced_gradient():

    x1, x2, x3, x4 = symbols('x1 x2 x3 x4')
    xvars = [x1, x2, x3, x4]

    fx = x1**2 + x2**2 + x3**2 + x4**2 - 2*x1 - 3*x4
    hxs = [2*x1 + x2 + x3 + 4*x4 - 7, x1 + x2 +
           2*x3 + x4 - 6]                                  # Constraints to be ob
    alpha_0 = 1
    gamma = 0.4
    max_iter = 100
    max_outer_iter = 50
    eps_1, eps_2, eps_3 = 0.001, 0.001, 0.001

    xcurr = np.array([2, 2, 1, 0])

    dfx = np.array([diff(fx, xvar) for xvar in xvars])
    dhxs = np.array([[diff(hx, xvar) for xvar in xvars] for hx in hxs])
```

```
    nonbasic_vars = len(xvars) - len(hxs)
    opt_sols = []

    for outer_iter in range(max_outer_iter):

        print('\n\nOuter loop iteration: {0}, optimal solution: {1}'.fo
            outer_iter + 1, xcurr))
        opt_sols.append(fx.subs(zip(xvars, xcurr)))

        # Step 1

        delta_f = np.array([df.subs(zip(xvars, xcurr)) for df in dfx])
        delta_h = np.array([[dh.subs(zip(xvars, xcurr)) for dh in dhx]
                           for dhx in dhxs])                # Value of
        # Computation of J and C matrices
        J = np.array([dhx[nonbasic_vars:] for dhx in delta_h])
        C = np.array([dhx[:nonbasic_vars] for dhx in delta_h])
        delta_f_bar = delta_f[nonbasic_vars:]
        delta_f_cap = delta_f[:nonbasic_vars]

        J_inv = np.linalg.inv(np.array(J, dtype=float))
        delta_f_tilde = delta_f_cap - delta_f_bar.dot(J_inv.dot(C))

        # Step 2
```

```
    if abs(delta_f_tilde[0]) <= eps_1:
        break

    d_bar = - delta_f_tilde.T
    d_cap = - J_inv.dot(C.dot(d_bar))
    d = np.concatenate((d_bar, d_cap)).T

    # Step 3

    alpha = alpha_0

    while alpha > 0.001:

        print('\nAlpha value: {0}\n'.format(alpha))

        # Step 3(a)

        v = xcurr.T + alpha * d
        v_bar = v[:nonbasic_vars]
        v_cap = v[nonbasic_vars:]
        flag = False

        for iter in range(max_iter):
```

```
                print('Iteration: {0}, optimal solution obtained at x =
                    iter + 1, v))
                h = np.array([hx.subs(zip(xvars, v)) for hx in hxs])
                # Check if candidate satisfies all constraints
                if all([abs(h_i) < eps_2 for h_i in h]):
                    if fx.subs(zip(xvars, xcurr)) <= fx.subs(zip(xvars,
                        alpha = alpha * gamma
                        break
                    else:
                        xcurr = v
                        flag = True
                        break

                # Step 3(b)

                delta_h_v = np.array([[dh.subs(zip(xvars, v))
                                       for dh in dhx] for dhx in dhxs])
                J_inv_v = np.linalg.inv(
                    np.array([dhx[nonbasic_vars:] for dhx in delta_h_v]
                v_next_cap = v_cap - J_inv_v.dot(h)

                # Step 3(c)

                if abs(np.linalg.norm(np.array(v_cap - v_next_cap, dtyp
```

```
        v_cap = v_next_cap
        v = np.concatenate((v_bar, v_cap))
    else:
        v_cap = v_next_cap
        v = np.concatenate((v_bar, v_cap))
        h = np.array([hx.subs(zip(xvars, v)) for hx in hxs]
        if all([abs(h_i) < eps_2 for h_i in h]):

            # Step 3(d)

            if fx.subs(zip(xvars, xcurr)) <= fx.subs(zip(xv
                alpha = alpha * gamma
                break
            else:
                xcurr = v
                flag = True
                break
        else:
            alpha = alpha * gamma
            break

if flag == True:
    break

```

```python
    print('\n\nFinal solution obtained is: {0}'.format(xcurr))
    print('Value of the function at this point: {0}\n'.format(
        fx.subs(zip(xvars, xcurr))))

    # Plot the solutions obtained after every iteration
    plt.plot(opt_sols, 'ro')
    plt.show()


if __name__ == '__main__':
    generalized_reduced_gradient()
```