FACULTAD DE CIENCIAS

Tercera Práctica Calificada

Esaú Flores Villar

Universidad Nacional de Ingeniería

9 de octubre de 2024

Índice

Penalty Methods

Se usan para convetir problemas de optimizacion con restricciones. a problemas sin restricciones añadiendo terminos de penalizacion al objetivo lo que nos permite usar los metodos vistos anteriormente.

$$\begin{gathered} \text{minimice } f(x) \\ \text{sujeto a } g(x) \leq 0 \\ h(x) = 0 \end{gathered}$$

un método de penalizacion simple consta de un numero de restricciones

$$\rho(x) = \sum(g_i(x) > 0) + \sum(h_j(x) \neq 0)$$

resultando un problema de optimizacion sin restricciones que penaliza la impracticabilidad

$$\text{minimice } f(x) + \mu\rho_{count}(x) \text{ donde } \mu > 0$$

Teoría

$$\text{minimice } sin(x)$$
$$\text{sujeto a } x^2 < 1$$

$\mathcal{L}(x, \mu) = sin(x) + \mu(x^2 - 1)$
la funcion objetivo tiene una traza negra
la region factible esta en azul
las lineas del $L(x, \mu)$ se trazan en morado, para diferentes valores de μ tiene un minimo en $x = -1$ para $\mu = -0{,}841$

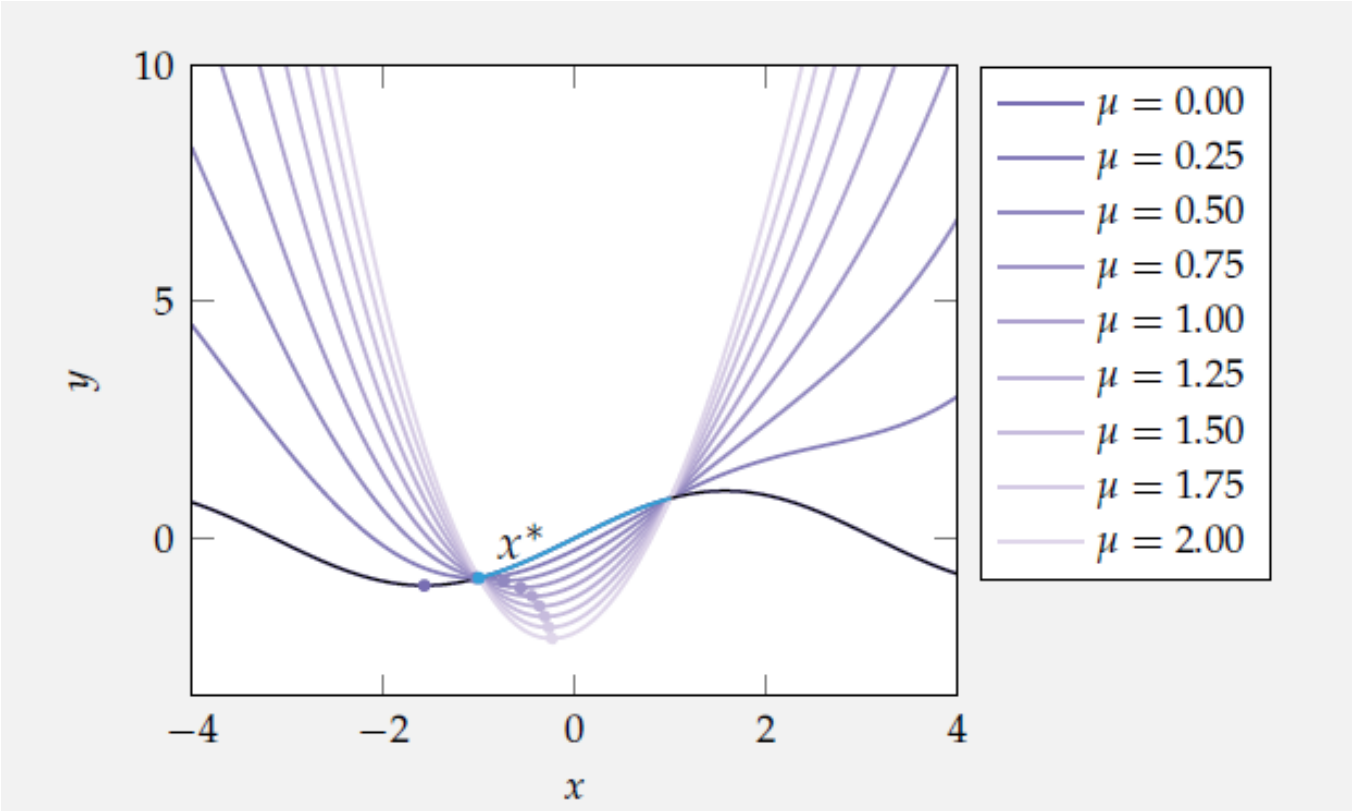

Figura:

Los metodos de penalizacion requieren un punto inicial y un valor pequeño de μ en la ecuacion del problema de optimizacion. El punto de diseño se utiliza luego como punto de partida para otra optmizacion con una pena aumentada. El algoritmo proporciona la implementacion. Y se sigue con el procedimiento hasta que el punto resutante sea factible o se haya alcanzado un numero maximo de iteraciones. La penalizacion preservara la solucion del problema para valores grandes de μ pero introduce una marcada discontinuidad Los puntos que no estan dentro del conjunto factible carecen de gradiente, es decir de informacion que oriente la busqueda hacia la viabilidad. Se puede usar unas penalizaciones cuadráticas para producir una funcion objetivo suave.

$$\rho_{cuadratico}(x) = \sum max(g_i(x), 0)^2 + \sum (h_i(x))^2$$

Algoritmo del metodo con penalizacion

1: **Input** x_0, μ, β,ϵ $nMax$, f
2: Inicializar contador $i = 0$
3: **for** $i = 1, 2, \ldots, nMax$ **do**
4: $x_{i+1} \leftarrow$ minimizar$(f(x) + \mu \sum max(g_i(x), 0)^2 + \mu \sum (h_i(x))^2)$
5: **if** $\mu_{i+1} * P(x_{i+1}) < \epsilon$ **then**
6: Retornar: x_{i+1}
7: **else**
8: $\mu_{i+1} \leftarrow \beta * \mu_i$
9: $i \leftarrow i + 1$
10:

Podemos usar penalizaciones cuadraticas para producir una funcion objetivo fluida.

Codigo

```
"""Minimizar
 f (x y) = x^2 +xy +y^2 - 2y
 sujeto a
 x + y + 2 <= 0
 x + y - 5 = 0

 """
def P(x):
    return (x[0]+x[1]-2)**2 + (x[0]+x[1]-5)**2

def penalizacion(x,mu,epsilon,beta,nMax):
    i = 1
    while i < nMax :
        x = grad_desc_for_penalizacion(x,mu)
        if mu * P(x) < epsilon:
            return x
        else:
            mu = mu * beta
            i = i + 1
    print("maximo numero de iteracciones")
```

Codigo

```
1  def grad_desc_for_penalizacion(x:np.array,mu, lr=0.1,nMax=100,epsilon
       =0.0001):
2      points = []
3      i = 0
4      gr = gF(x,mu)
5      while i<nMax and np.linalg.norm(gr)>=epsilon:
6          x = x -lr*gr
7          gr = gF(x,mu)
8          points.append(x)
9          i = i + 1
10     return x
```

Codigo

```
1 if __name__=='__main__':
2     x1 = np.array([3,4])
3     mu = 0.5
4     beta = 0.5
5     epsilon = 0.001
6     nMax = 50
7     x=penalizacion(x1,mu,epsilon,beta,nMax)
8     print(x)
9
```

Salida en consola

```
el minimo en :
[-0.66621479  1.33378521]
lo que da scipy
[-0.66666668  1.33333333]
```

Figura: el optimo

Codigo

```
1  def scipy_test():
2      restriccion = dict(type = 'ineq',fun = g)
3      x_opt = optimize.minimize(f, (0, 0), method='BFGS').x
4      x_cons_opt = optimize.minimize(f, (0, 0), method='SLSQP',constraints
       =restriccion).x
5      return x_cons_opt
6
7  def grafica_de_fxy():
8      fig = plt.figure ()
9      ax = fig.add_subplot(111,projection = "3d")
10     x = np.linspace(-5,5,50)
11     y = np.linspace(-5,5,50)
12     X,Y = np.meshgrid(x,y)
13     Z = X**2 +X*Y +X**2 -2*Y
14     x_cons_opt=scipy_test()
15     ax.plot_surface(X,Y,Z,cmap = "viridis")
16     ax.plot(x_cons_opt[0], x_cons_opt[1], 'r*', markersize=15)
17     plt.show()
18
```

The augmented Lagrange Method

Este metodo es una variacion del metodo de penalizacion para restricciones de igualdad A diferencia del metodo anterior donde μ debe crecer este metodo funciona con valores mas pequeños de μ Utiliza una penalizacion cuadratica y una lineal Para problemas de optimizacion con igualdad

$$\rho_{lagrange} = \frac{1}{2}\sum(h_i)^2 - \sum \lambda_i h_i x$$

donde λ tiende a los multiplicadores de lagrange. Ademas de μ λ tambien se actualiza de acuerdo a esta regla

$$\lambda_{i+1} = \lambda_i - \mu * h(x)$$

Algoritmo Augmented Lagrange Method

1: **Input** x_0, μ, β,ϵ $nMax$, f
2: Inicializar contador $i = 1$
3: **for** $i = 1, 2, \ldots, nMax$ **do**
4: $\quad x_{i+1} \leftarrow minimizar(f(x) + \frac{\mu}{2} h(x)^2 - \lambda h(x))$
5: $\quad \lambda_{i+1} \leftarrow \lambda_i - \mu * h(x)$
6: $\quad \mu_{i+1} \leftarrow \beta * \mu_i$
7: $\quad i \leftarrow i + 1$
8: $\quad x \leftarrow x_{i+1}$
9: **end for**
10: **Retornar:** x
11: $=0$

Codigo

```
"""  minimizar f (x y) = (x - 4)^2 + (y - 4)^2
        sujeto a x + y - 5 = 0                  """
def augmented_lagrange_method(x,nMax,mu,beta):
    lam = 0
    for i in range(nMax):
        x = grad_desc_for_lagrange_aumented(x,mu,lam)
        lam = lam - mu*h(x)
        mu = mu*beta
    return x
```

Codigo

```
#F = f + uh(x)^2/ 2 -lam*h(x)
#F = (x - 4)**2+(y - 4)**2 + 0.5*mu *(x + y - 5)^2-lam*(x + y - 5)

def gF(x,mu,lam):
    dx = 2*(x[0]-4) + 2*0.5*mu*(x[0]+x[1]-5)-lam
    dy = 2*(x[1]-4) + 2*0.5*mu*(x[0]+x[1]-5)-lam
    return np.array([dx,dy])

def grad_desc_for_lagrange_aumented(x:np.array,mu,lam, lr=0.1,nMax=100,
     epsilon=0.0001):
    points = []
    i = 0
    gr = gF(x,mu,lam)
    while i<nMax and np.linalg.norm(gr)>=epsilon:
        x = x -lr*gr
        gr = gF(x,mu,lam)
        points.append(x)
        i = i + 1
    return x

```

Codigo

```
1
2
3     def scipy_test():
4     restriccion = dict(type = 'eq',fun = h)
5     x_opt = optimize.minimize(f, (0, 0), method='BFGS').x
6     x_cons_opt = optimize.minimize(f, (0, 0), method='SLSQP',constraints
       =restriccion).x
7     return x_cons_opt
8
9     def h(X):
10    x,y= X
11    return x + y - 5
12
13    if __name__=='__main__':
14    x1 = np.array([1,1])
15    x = augmented_lagrange_method(x1,50,1,1) #depende demasiado de los
       valores de los paramtros
16    print("minimo en ")
17    print(x)
18    print(scipy_test())
19    grafica_de_fxy()
20
```

Salida en consola

```
minimo en
[2.50000431 2.50000431]
[2.5 2.5]
```

Figura: resultados,primero de acuerdo al codigo, segundo scipY

Metodo de Barrera o Metodo de puntos interiores

Los metodos de punto interior , en ocasiones denominados de barrera, son metodos de optmizacion que garantizan que los puntos de busqueda siempre sigan siendo factibles. Estos metodos usan una funcion de barrera que se acerca al infinito cuando uno se acerca al limite de la restriccion Dicha funcion de barrera debera satisfacer las siguientes propiedades:

1. $\rho_{barrera}(x)$ es continuo
2. $\rho_{barrera}(x)$ es no negativa en la region factible
3. $\rho_{barrera}(x)$ tiende al infinito cuando x se acerca al limite de la restriccion

algunos ejemplos de la funcion de funciones barrera

$$\rho_{barrera}(x) = -\sum \frac{1}{g_i(x)}$$

$$\rho_{barrera} = -\sum \begin{cases} \log\left(-g_i(x)\right) & \text{if } g_i(x) \geq -1 \\ 0 & \text{otros casos} \end{cases}$$

Un problema de restricciones con desigualdad se puede convertir en uno sin desigualdad

$$\text{minimizar } f(x) + \frac{1}{\mu}\ p_{barrera}(x)$$

cuando μ aumenta la penalizacion cerca del limite disminuye

Se debera tener especial cuidado para que las busquedas de lineas no salgan de la zona factible. Las busquedas lineales $f(x + \alpha d)$ estan restringidas al intervalo $\alpha = [0, \alpha_u]$ donde α_u es el paso hacia el limite mas cercano, en la practica se elige $x + \alpha d$ de modo que este dentro del limite para evitar la singularidad del limite Al igual que el metodo de penalizacion , el metodo del punto interior comienza con un valor μ y lo levanta hasta la convergencia . El metodo del punto interior suele terminar cuando la diferencia entre puntos subsiguientes es menor que un ϵ El metodo del punto interior requiere un punto factible desde el cual iniciar la busqueda.Un metodo conveniente para encontrar un punto factible es optimizar la funcion de penalizacion cuadratica.

$$\text{minimizar } \rho_{cuadratica}(x)$$

Algoritmo Puntos Interiores

1: **Input** x_1, ϵ, $nMax$, μ, β
2: Inicializar $\Delta = inf$, contador $i = 1$
3: **while** $i = 1, 2, \ldots, nMax$ $\Delta > \epsilon$ **do**
4: $\quad x_{i+1} \leftarrow \text{minimizar}(f(x) - \frac{1}{\mu}\frac{1}{g(x)})$
5: $\quad \Delta \leftarrow \|x_{i+1} - x_i\|$
6: $\quad x_i \leftarrow x_{i+1}$
7: $\quad \mu \leftarrow \mu * \beta$
8: **end while**
9: **Retornar:** x

Codigo

```
""" f (x) = -(x-4)^2 + 4
        x <=5
        x >=3
        """
#F(x,mu) = f(x) -1/mu*g1(x) -1/mu*g2(x)
#         =-(x-4)^2 +4 -1/mu*(x-5) -1/mu*(3-x)

def puntos_interiores(x,mu=1,epsilon=0.01,beta=0.5,nMax=100):
    delta = np.inf
    i = 1
    while delta > epsilon :
        x1 = metodo_newton(x,mu)
        delta = np.linalg.norm(x1-x)
        x = x1
        mu = mu * beta
        i = i + 1
    return x

```

Codigo

```
1  def dx(x,mu):
2      return -2*(x-4) + 1/(mu*(x-5)**2)-1/(mu*(3-x)**2)
3
4  def dxx(x,mu):
5      return  -2 -2/(mu*(x-5)**3)-2/(mu*(3-x)**3)
6
7  def metodo_newton(x,mu,epsilon=0.01,nMax=100):
8      delta = np.inf
9      i = 1
10     while i<nMax and delta >epsilon:
11         x1 = x - dx(x,mu)/dxx(x,mu)
12         delta =  mt.fabs(x1 -x)
13         x  = x1
14         i = i + 1
15     return x
16
```

Codigo

```
1  if __name__=='__main__':
2      x0 = 0
3      x = puntos_interiores(x0)
4      print("minimo")
5      print(x)
6
```

Salida en consola

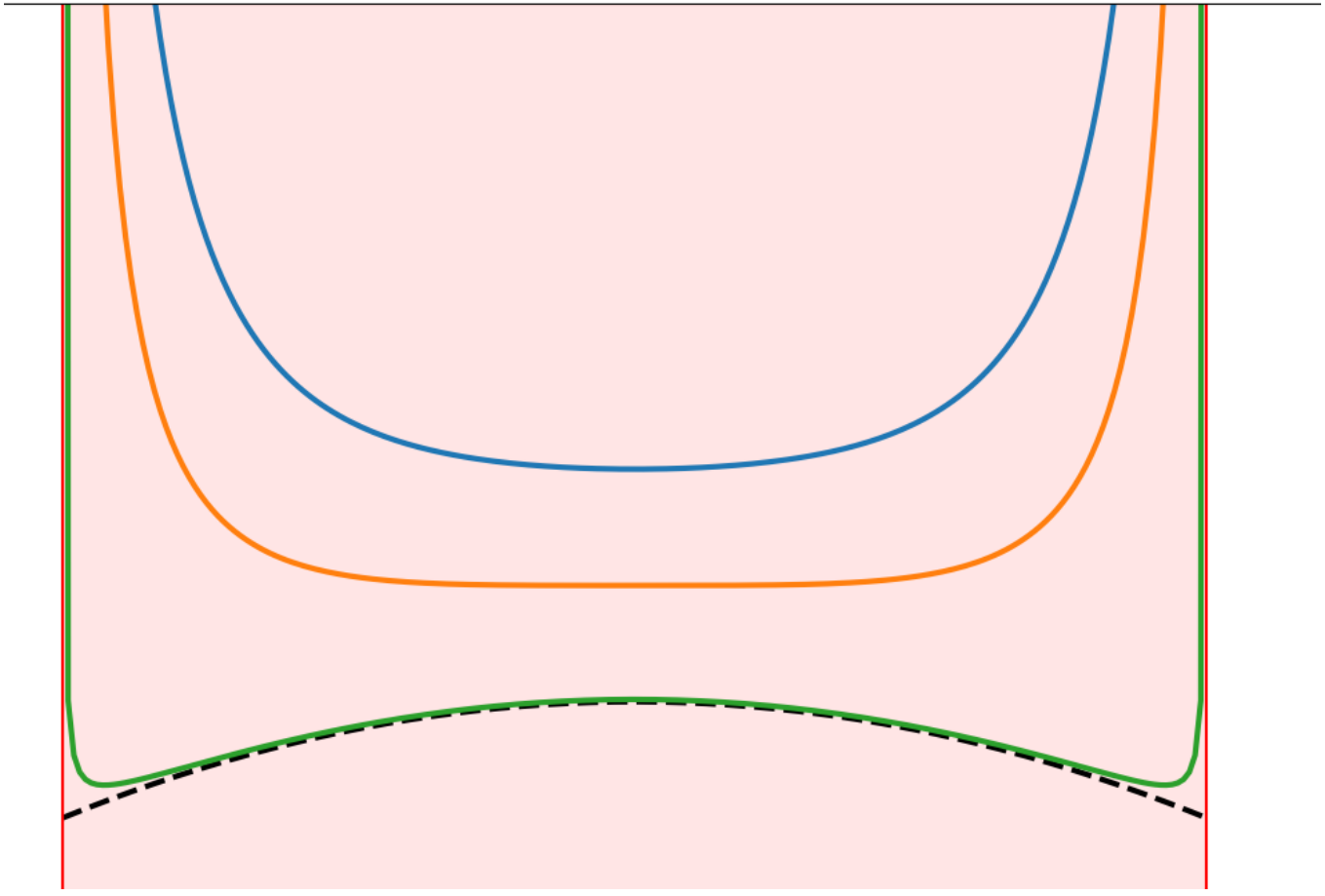

Figura: de acuerdo a distintos valores de μ