SciPy Optimize

Doherty Andrade

1 SciPy optimize: Programação Linear e equações não lineares

SciPy optimize tem um conjunto grande funções para tratar com minimização (ou maximização) funções objetivo, inclusive sujeitas a restrições. Inclui solucionadores de problemas não lineares (com suporte para algoritmos de otimização local e global), programação linear, mínimos quadrados restritos e não lineares, determinação de raiz de funções não lineares e ajuste de curva.

Neste post vamos apresentar exemplos em problemas de programação linear. Devemos colocar o ppl na forma

$$\min c.x$$

$$Ax \leq b.$$

Consideremos inicialmente um problema de minimização. Podemos resumir o problema da seguinte forma:

$$\begin{cases} \min(4x\_1 + 2x\_2 + x\_3 + 10x\_4 + 5x\_5) \\ x\_2 + 5x\_3 + 4x\_4 + 3x\_5 \geq 10 \\ 2x\_1 + x\_2 + 3x\_4 + 2x\_5 \geq 30 \\ 3x\_1 + x\_2 + 9x\_4 \geq 18 \\ x\_1, \geq 0, x\_2 \geq 0, x\_3 \geq 0, x\_4 \geq 0, x\_5 \geq 0. \end{cases}$$

```
In [1]: # Importando as bibliotecas necessárias
        import numpy as np
        from scipy.optimize import linprog

In [2]: # Entrando com o conjunto de restrições, todas da forma com <=

        #matrix A
        A = np.array([[0,-1, -5, -4,-3 ], [-2, -1, 0, -3,-2], [-3, -1, 0,-9,0], [-1, 0, 0, 0, 0]
                      [0, -1, 0, 0, 0], [0, 0, -1, 0 , 0], [0, 0, 0, -1, 0], [0,0,0,0,-1]])

        # vetor b
        b = np.array([-10, -30, -18, 0, 0, 0,0,0])

        # coeficientes da função objetivo
        c = np.array([4, 2, 1,10, 5])

        # Resolvendo o problema
        res = linprog(c, A_ub=A, b_ub=b)
```

```
        # Imprimindo resultados
        print('Valor ótimo:', round(res.fun, ndigits=2),
              '\nValores de x:', res.x,
              '\nNúmero de iterações realizadas:', res.nit,
              '\nStatus:', res.message)
```

```
Valor ótimo: 60.0
Valores de x: [10. 10.  0.  0.  0.]
Número de iterações realizadas: 8
Status: Optimization terminated successfully.
```

Determinar a solução ótima do seguinte PPL

$$\min(20x_1 + 30x_2 + 16x_3)$$

$$\begin{cases} 2.5x\_1 + 3x\_2 + x\_3 \geq 3 \\ x\_1 + 3x\_2 + 2x\_3 \geq 4 \\ x\_1 \geq 0, x\_2 \geq 0, x\_3 \geq 0. \end{cases}$$

```
In [3]: # Entrando com o conjunto de restrições, todas da forma com <=

        #matrix A
        A = np.array([[-2.5,-3,-1 ], [-1,-3,-2],[-1, 0, 0], [0, -1, 0], [0,0,-1]])

        # vetor b
        b = np.array([-3, -4, 0, 0, 0])

        # coeficientes da função objetivo
        c = np.array([20,30,16])

        # Resolvendo o problema
        res = linprog(c, A_ub=A, b_ub=b)

        # Imprimindo resultados
        print('Valor ótimo:', round(res.fun, ndigits=2),
              '\nValores de x:', res.x,
              '\nNúmero de iterações realizadas:', res.nit,
              '\nStatus:', res.message)
```

```
Valor ótimo: 36.0
Valores de x: [0.         0.66666667 1.        ]
Número de iterações realizadas: 2
Status: Optimization terminated successfully.
```

Para determinação de soluções equações não lineares usamos o comando solve que utiliza o método de Newton-Raphson. Veremos um exemplo.

Vamos determinar a solução da equação $x - cos(x) = 0$. Como o método de Newton-Raphson exige uma aproximação inicial, vamos tomar para este exemplo $x_0 = 0.5$.

```
In [4]: from scipy.optimize import fsolve
        from math import cos, sin, exp, pi

        def f(x):
            return x- cos(x)

        x = fsolve(f, 0.5)

        print("A raiz aproximada é x =%21.19g" % x)
```

```
A raiz aproximada é x =0.7390851332151601172
```

Pode-se utilizar o método da bissecção.

```
In [5]: from scipy.optimize import bisect
        from math import cos, sin, exp, pi

        def f(x):
            return x -cos(x)

        x = bisect(f, 0.5, 1, xtol=1e-6)

        print("A raiz é aproximadamente  x=%14.12g,\n"
              "o erro é menor do que 1e-6." % (x))
```

```
A raiz é aproximadamente  x=0.739085197449,
o erro é menor do que 1e-6.
```

Usando diretamente o método de Newton.

```
In [6]: #newton(func, x0, fprime, args, tol, ...])

        from math import cos, sin, exp, pi
        from scipy.optimize import newton

        def f(x):
            return x-cos(x)

        def fprime(x):
            return 1+sin(x)

        x = newton(f, 0.75, fprime, tol=1e-6)

        print("A raiz é aproximadamente  x=%14.12g,\n"
              "o erro é menor do que 1e-6." % (x))
```

```
A raiz é aproximadamente  x=0.739085133215,
o erro é menor do que 1e-6.
```