Python para ingenieros ¶

OPTIMIZACIÓN

En esta notebook vamos a estudiar cómo resolver problemas de optimización en Python. Para ello, nos vamos a centrar fundamentalemnte en el módulo `SciPy` y, en concreto, su submódulo llamado `optimize`, si bien en Python existen otras librerías que proporcionan métodos para resolver este tipo de problemas.

- ***scipy.optimize (https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html)***: Este es el módulo de optimización de SciPy, y en él vamos a encontrar, entre otros métodos, varios algoritmos destinados a resolver problemas *no lineales* de optimización.

Contenido:

- Conceptos Generales.
- Ejemplos prácticos:
  - Optimización sin restricciones
  - Optimización con restricciones

Conceptos Generales

La optimización es fundamental en cualquier problema relacionado con la toma de decisiones, ya que en estos problemas se busca elegir la mejor decisión posible dentro de un conjunto de alternativas, tarea a la que nos ayudan los métodos de optimización.

Un problema de optimización consta de un ***función objetivo***, que es la función que se pretende maximizar o minimizar, que depende de un conjunto de ***variables independientes o parámetros***, que en muchos casos están sujetas a un conjunto de condiciones que definen los valores aceptables y limitaciones de dichas variables, y que se denominan ***restricciones del problema***.

La solución de dicho problema de optimización, si existe, consiste en el conjunto de valores de las variables independientes que, de acuerdo con las limitaciones del problema, proporcionan un valor óptimo de la función objetivo, lo que en términos matemáticos, implica maximizar o minimizar dicha función.

Hay que resaltar, además, que el éxito o no en el problema de optimización, vendrá determinado por el planteamiento de una formulación adecuada del mismo y por el conocimiento de las fortalezas y debilidades de los distintos métodos de optimización.

Ejemplos prácticos

Como siempre, lo primero que vamos a hacer es importar los paquetes que vamos a usar:

In [1]:

```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt

from scipy.optimize import minimize
```

scipy.optimize

El submódulo `optimize` del paquete `SciPy` proporciona, diferentes **métodos de optimización**, así como otros métodos destinados a cálculos de búsqueda de raices, mínimos cuadrados...

Método Nelder-Mead o Downhill Simplex.

El método de Nelder-Mead o Downhill Simplex, no debe ser confundido con el método Simplex, pues en este caso, se aplica para calcular el mínimo local de una función objetivo multidimensional y no lineal (pero que varía suavemente), cuyas derivadas pueden no ser conocidas.

Hace uso del concepto de simplex, un tipo de politopo de n+1 vértices en n dimensiones (ej: tetraedro en el espacio tridimensional, triángulo en un plano), y en función del valor de la función en esos puntos, cambia el punto del simplex con peor valor tras la evaluación, y modifica el simplex, y lo expande o encoje hasta llegar a la función final.

Vamos a considerar la ***función de Himmelblau***, que es usada para testear el rendimiento de algoritmos de optimización.

Su ecuación es:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

y presenta:

- un máximo local $f(x, y) = 181.617$ siendo $x = -0.270845$ e $y = -0.923029$.
- cuatro mínimos locales:
  - $f(x, y) = 0.0$ siendo $x = 3.0$ e $y = 2.0$.
  - $f(x, y) = 0.0$ siendo $x = -2.8051185$ e $y = 3.131312$.
  - $f(x, y) = 0.0$ siendo $x = -3.779310$ e $y = -3.283186$.
  - $f(x, y) = 0.0$ siendo $x = 3.584428$ e $y = -1.848126$.

In [2]:

```
# definimos la función
def himmelblau(x):
    return (x[0]** 2 + x[1] - 11)**2 + (x[0] + x[1]**2 -7)**2
```

In [3]:

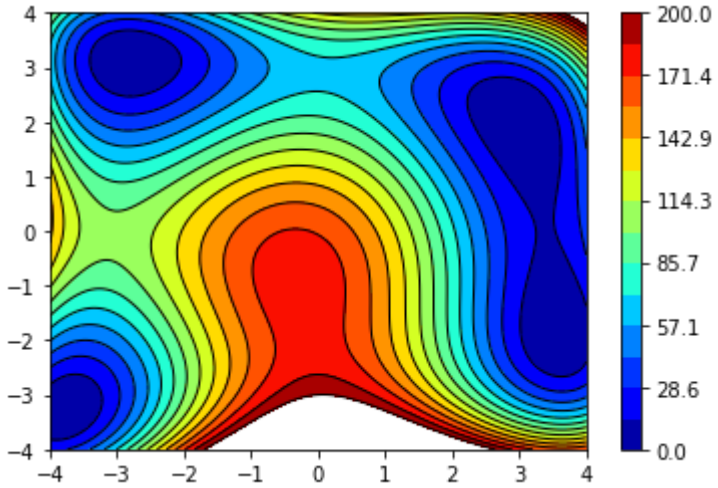
```
# datos para  representacion gráfica
x = np.linspace(-4, 4, 200)
y = np.linspace(-4, 4, 200)
xx, yy = np.meshgrid(x, y)
zz = himmelblau(np.array([xx, yy]))
```

In [4]:

```
# representacion contour y contourf
plt.contourf(xx, yy, zz, np.linspace(0, 200, 15), cmap=plt.cm.jet)
plt.colorbar()
plt.contour(xx, yy, zz, np.linspace(0, 200, 15),  colors='k', linewidths=1)
```

Out[4]:

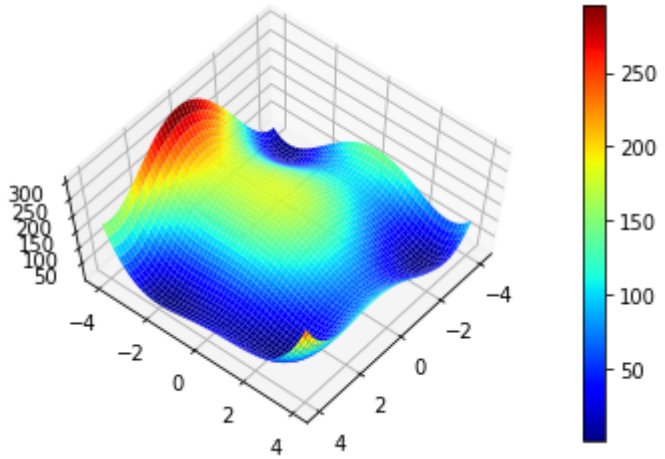
```
<matplotlib.contour.QuadContourSet at 0x1fac0a862e8>
```



In [5]:

```
#representacion surface
from mpl_toolkits.mplot3d.axes3d import Axes3D
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.view_init(65, 40)
surf = ax.plot_surface(xx, yy, zz, cmap=plt.cm.jet, linewidths=1)
fig.colorbar(surf)
plt.show()
```



In [6]:

```
# definimos la posición inicial
x0 = np.array([3.5, 2.1])
```

In [7]:

```
# aplicamos el método de optimización
sol = minimize(himmelblau, x0, method="Nelder-Mead")
sol
```

Out[7]:

```
 final_simplex: (array([[2.99996255, 2.00000681],
       [2.99999247, 1.99994762],
       [3.00001357, 2.00005231]]), array([4.75758877e-08, 5.66274259e-08, 6.
75309789e-08]))
           fun: 4.757588770698023e-08
       message: 'Optimization terminated successfully.'
          nfev: 64
           nit: 33
        status: 0
       success: True
             x: array([2.99996255, 2.00000681])
```

In [8]:

```
# aplicamos el método de optimización ajustando la tolerancia
sol = minimize(himmelblau, x0, method="Nelder-Mead", tol= 1e-10)
sol
```

Out[8]:

```
 final_simplex: (array([[3., 2.],
       [3., 2.],
       [3., 2.]]), array([7.62624100e-21, 4.26791443e-20, 6.96226450e-20]))
           fun: 7.626241003048843e-21
       message: 'Optimization terminated successfully.'
          nfev: 147
           nit: 75
        status: 0
       success: True
             x: array([3., 2.])
```

Comportamiento según el punto inicial

In [9]:

```
def plot_conv_himmelblau(x0, y0):
    plt.contourf(xx, yy, zz, np.linspace(0, 200, 15), cmap=plt.cm.jet)
    plt.colorbar()
    plt.contour(xx, yy, zz, np.linspace(0, 200, 15),  colors='k', linewidths=1)

    sol = minimize(himmelblau, np.array([x0, y0]), method="Nelder-Mead", callback=lambda x:
    plt.show()
    print(sol)
```

In [10]:

```
from ipywidgets import interact
```

In [11]:

```
interact(plot_conv_himmelblau, x0=(-4, 4), y0=(-4, 4))
```

Failed to display Jupyter Widget of type `interactive`.

If you're reading this message in the Jupyter Notebook or JupyterLab Notebook, it may mean that the widgets JavaScript is still loading. If this message persists, it likely means that the widgets JavaScript library is either not installed or not enabled. See the Jupyter Widgets Documentation (https://ipywidgets.readthedocs.io/en/stable/user_install.html) for setup instructions.

If you're reading this message in another frontend (for example, a static rendering on GitHub or NBViewer (https://nbviewer.jupyter.org/)), it may mean that your frontend doesn't currently support widgets.

Out[11]:

```
<function __main__.plot_conv_himmelblau>
```

Optimización con restricciones

En los problemas de optimización con restricciones, no sólo hay que definir la función objetivo, sin o también las restricciones a las que se ven sujetas las varaibles, y que deben pasarse al método de optimizacion.

A continuación, vamos a resolver el siguiente método utilizando el método 'L-BFGS-B':

$$ \begin{aligned} & \text{min} & & f(x) = (x -1)^2 - (y -2)^2 \\ & \text{sujeto a} & & 2 <= x <= 3 \\ &&& 0 <= y <= 1 \end{aligned} $$

In [12]:

```
# función a minimizar
def fun(x):
    return (x[0] -1)**2 + (x[1] -2)**2
```

In [13]:

```
# condición inicial
x0 = np.array([1, 1])
```

In [14]:

```
# minimizar la función sin restricciones

x_opt = minimize(fun, x0, method='L-BFGS-B')

x_opt
```

Out[14]:

```
      fun: 0.0
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([9.99999988e-09, 9.99999988e-09])
  message: b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL'
     nfev: 6
      nit: 1
   status: 0
  success: True
        x: array([1., 2.])
```

In [15]:

```
# minimizar la función con restricciones
r_x, r_y = (2, 3), (0, 1)
x_opt_r = minimize(fun, x0, method='L-BFGS-B', bounds=[r_x, r_y])
x_opt_r
```

Out[15]:

```
      fun: 2.0
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([ 1.99999999, -1.99999999])
  message: b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL'
     nfev: 3
      nit: 0
   status: 0
  success: True
        x: array([2., 1.])
```

Hemos aprendido:

- Como resolver problemas de optimización usando diferentes métodos que nos proporciona `SciPy` .

Referencias:

Mabel Delgado, Alejandro Sáez, Jesús Espinola