Práctica 7:Búsqueda local

Raul L.

29 de marzo de 2022

1. Introducción

En la séptima práctica implementamos una optimización heurística sencilla para encontrar máximos locales de funciones, los ejemplos siendo de Womersley [1] — los matemáticamente inclinados pueden consultar su trabajo por métodos de búsqueda más sofisticados, guiados por métodos matemáticos de mayor rigor [2].

Buscamos minimizar la función unidimensional, a partir de un punto seleccionado al azar, realizando movimientos locales. Estando en x, seleccionará al azar un $\Delta x > 0$ pequeño, calculará los valores $f((x \pm \Delta x)$ y seleccionará el menor de los dos como el siguiente valor de x. Esto se repite k veces y aquel x que produjo el menor valor de $f(x)$ se regresa como el resultado. Se realizarán n réplicas, y el menor de ellos es el resultado de la búsqueda en sí. La primera versión es sencilla, ineficiente y con una sola réplica para poder entender el comportamiento de la búsqueda y visualizarla.

2. Objetivo

La tarea se trata de maximizar algún variante de la función bidimensional ejemplo, $g(x, y)$, con restricciones tipo $-3 \leq x$,y ≤ 3, con la misma técnica del ejemplo unidimensional. La posición actual es un par x, y y se ocupan dos movimientos aleatorios, Δx y Δy, cuyas combinaciones posibles proveen ocho posiciones vecino, de los cuales aquella que logra el mayor valor para g es seleccionado. Dibujado en tres dimensiones, $g(x, y)$ se ve así:[2].

3. Código

Para este código se utilizó como base el código de la doctora donde se hicieron modificaciones variando la función y los parámetros base como también se crearon los 5 puntos al mismo tiempo y se revisó con una x el mejor valor conocido.

Código en Python

`https://satuelisa.github.io/simulation/p7.html`

Código creado en Python

`https://github.com/Raullr28/Resultados/blob/main/P7/practica_7.py`

```
def g(x, y):
    return np.exp(-x**2)+ np.exp(-y**2)

low = -6
high = -low
step = 0.20
tmax = 500

x = np.arange(low, high, step)
y = np.arange(low, high, step)
x, y = np.meshgrid(x, y)
z =np.exp(-x**2)+ np.exp(-y**2)

fig = plt.figure()
ax = plt.axes(projection='3d')
s = ax.plot_surface(x, y, z, cmap=cm.coolwarm, linewidth=0, antialiased=False)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.01f'))
fig.colorbar(s, shrink=0.5, aspect=5)
plt.savefig("p7_3dinicial.png")
plt.show()
```

Código 1: Representación de la función y parámetros utilizados.

```
 for iteracion in range(tmax):#entra a hacer ciclo de la particula
        #mueve particula en x+right  y x-left
        deltax = uniform(0, step/5)#movimiento en x
        leftx = currx - deltax
        leftx = low if leftx < low+step else leftx #asegurar que la particula esta dentro
        rightx = currx + deltax
        rightx = high if rightx > high-step else rightx

        deltay = uniform(0, step/5)
        lefty = curry - deltay
        lefty = low if lefty < low+step else lefty
        righty = curry + deltay
        righty = high if righty > high-step else righty

        lista=[(leftx, righty),(currx, righty),(rightx, righty),(leftx, curry),(rightx, curry),(leftx, lef
        v1 = g(leftx, righty)#valores evaluados en g
        v2 = g(currx, righty)
        v3 = g(rightx, righty)
        v4 = g(leftx, curry)
        v5 = g(rightx, curry)
        v6 = g(leftx, lefty)
        v7 = g(currx, lefty)
        v8 = g(rightx, lefty)
        vecinos=[v1, v2, v3, v4, v5, v6, v7, v8]
        vecino_mayor=vecinos.index(max(vecinos))# guarda la posicion del vecino mayor
        [currx, curry]=lista[vecino_mayor]#actualiza particula en posicion nueva
        if g(currx, curry) > g(bestx, besty):#Actualiza si es una mejor posicion
            [bestx, besty] = [currx, curry]
```

Código 2: Representación ciclo de la partícula.

4. Resultados

En la figura principal se muestra la función en 3D creada, en las imágenes siguientes se muestra el comportamiento de los 5 puntos al mismo tiempo marcando con una x el mejor valor en el transcurso de las 500 repeticiones modificándose cada vez que algún punto mejore su valor.

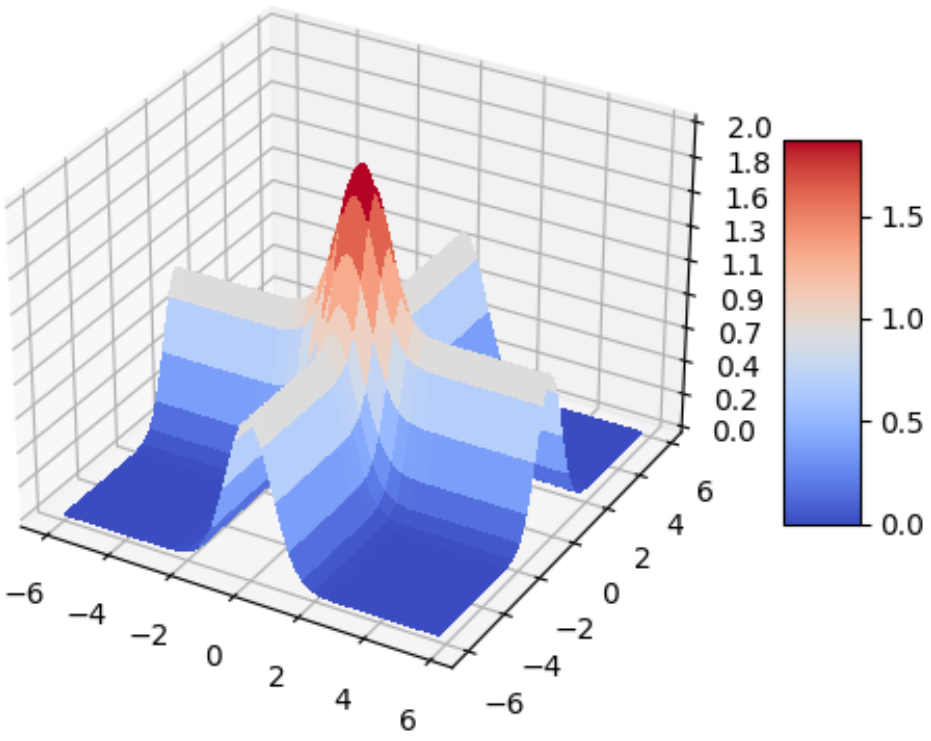
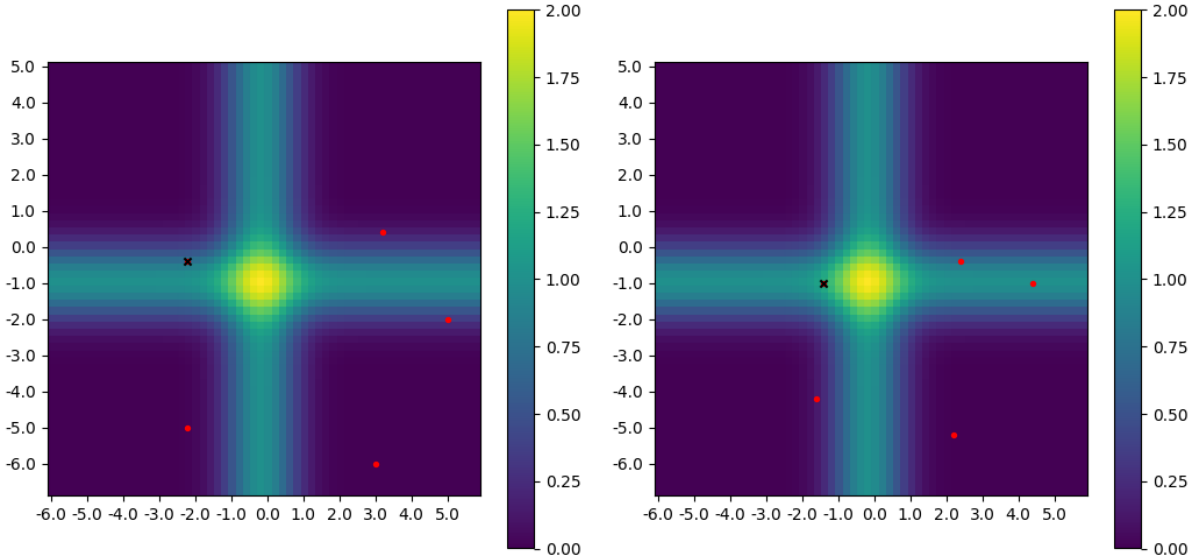
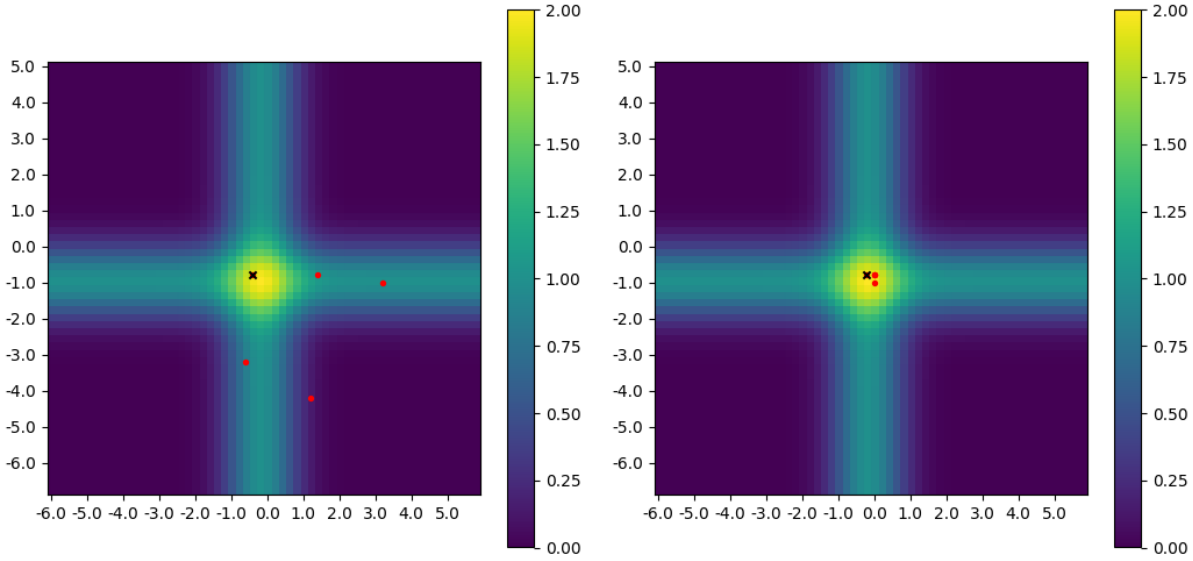


Figura 1: función 3D.



(a) Paso 29 (b) Paso 69

Figura 2: Comportamiento de el mejor punto.



(a) Paso 119 (b) Paso 299

Figura 3: Comportamiento de el mejor punto.

5. Reto 1

El primer reto es cambiar la regla del movimiento de una solución x (un vector de dimensión arbitraria) a la siguiente a la de recocido simulado: para optimizar una función $f(x)$, se genera para la solución actual x un sólo vecino $x{=}x + \Delta x$ (algún desplazamiento local). Se calcula $\delta = f(x\,') - f(x)$ (para minimizar; maximizando la resta se hace al revés). Si $\delta > 0$, siempre se acepta al vecino $x\,'$como la solución actual ya que representa una mejora. Si $\delta < 0$, se acepta a $x\,'$ con probabilidad $exp(\delta/T)$ y rechaza en otro caso. Aquí T es una temperatura que decrece en aquellos pasos donde se acepta una empeora; la reducción se logra multiplicando el valor actual de T con $\xi < 1$, como por ejemplo 0,995. Examina los efectos estadísticos del valor inicial de T y el valor de ξ en la calidad de la solución, es decir, qué tan bajo (para minimizar; alto para maximizar) el mejor valor termina siendo.

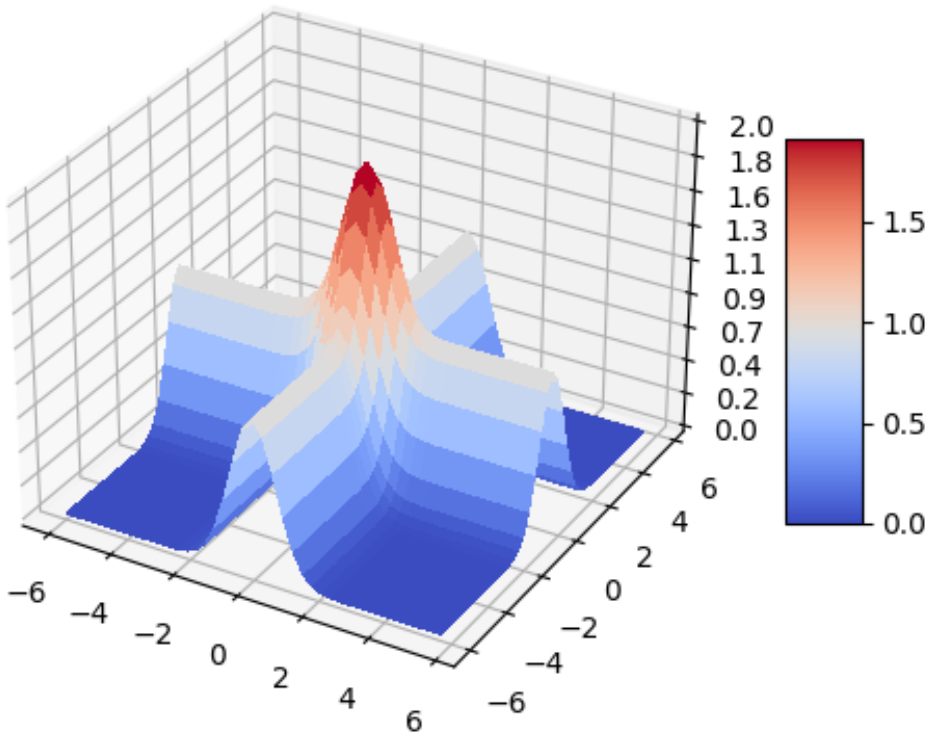
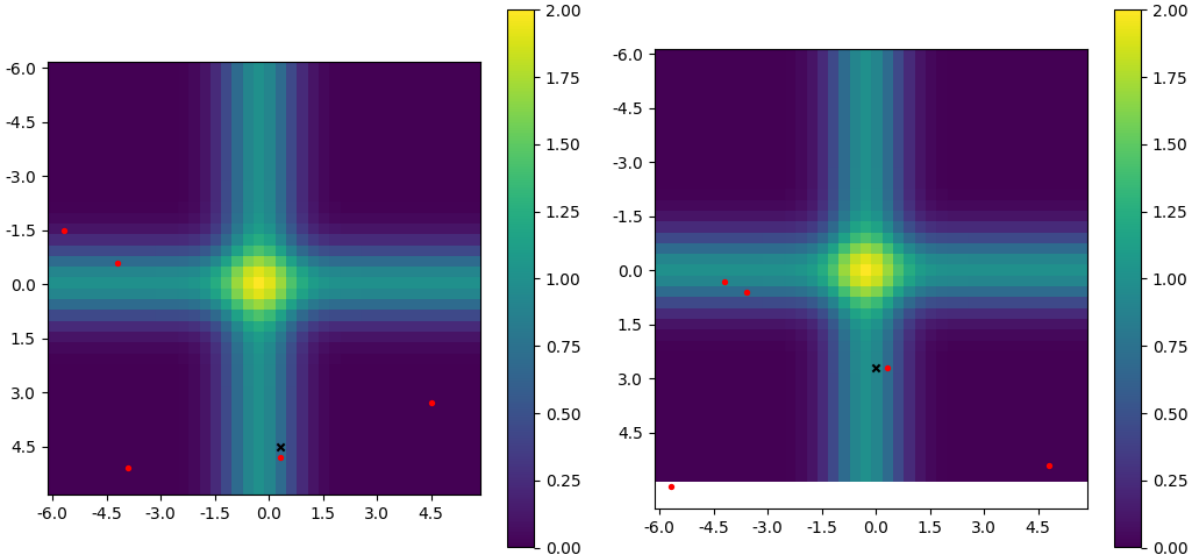


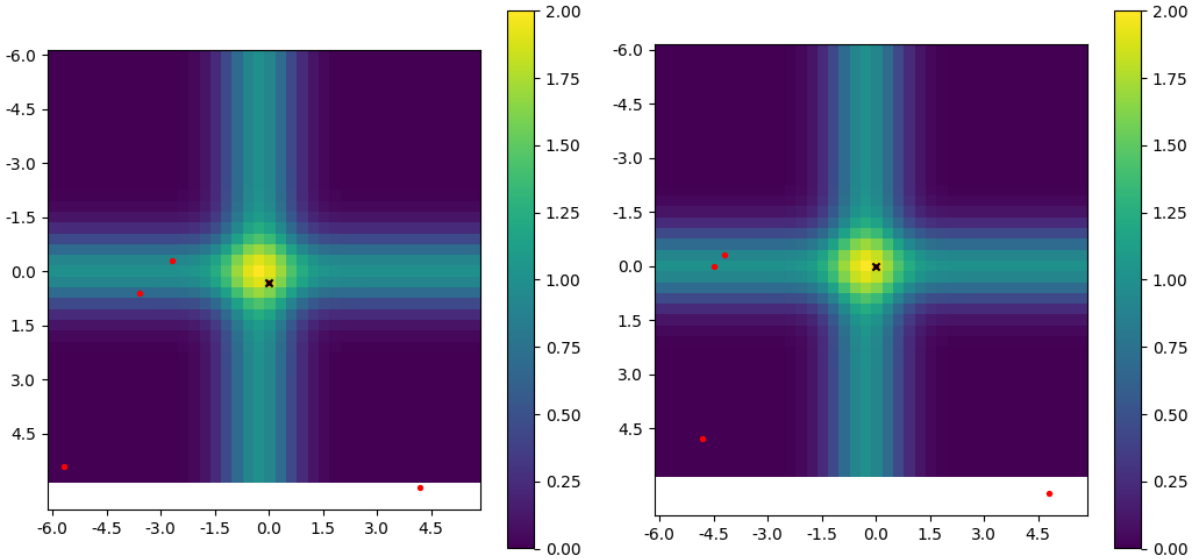
Figura 4: función 3D.

6. Resultados



(a) Paso 50 (b) Paso 500

Figura 5: Comportamiento de el mejor punto.



(a) Paso 750 (b) Paso1000

Figura 6: Comportamiento de el mejor punto.

7. Conclusión

Se demostró que en la tarea base le costo menos a los puntos maximizar las posiciones de los puntos para poder encontrar el mejor lugar mientras que en el reto 1 les costo más.

Referencias

[1] R. Womersley. Local and global optimization. 2008. URL `https://web.maths.unsw.edu.au/~rsw/lgopt.pdfl`.

[2] E. Schaeffer. Práctica 7: búsqueda local. 2022. URL `https://satuelisa.github.io/simulation/p7.html`.