Reporte 7:
Búsqueda Local

Jorge Torres

27 de marzo de 2022

1. Objetivo

La actividad consiste en maximizar una variante de una función bidimensional, $g(x, y)$, por medio de una búsqueda local aleatoria con restricciones en los ejes. La posición actual es un par x, y y se realizan 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. El resultado final es un compendio de los valores máximos encontrados para 10, 20 y 30 réplicas simultáneas de 100, 1000 y 10,000 pasos de la búsqueda. Además, en la sección 3 se visualizan algunos de los pasos que se realizaron para la búsqueda de 10 réplicas en una proyección plana de la función.

2. Desarrollo

El desarrollo de la práctica está basado en el código implementado por E. Schaeffer [1], con el cual realiza una búsqueda de valores mínimos para una función de una sola variable. En ésta actividad, se selecciona la ecuación 1 como función a evaluar por la particular cantidad de valores máximos y mínimos locales que presenta, lo que podría representar una dificultad para encontrar un máximo total. La ecuacuón se define en el código 1.

$$g(x, y) = \cos^2 x + 0{,}3x + \sin^2 y - 0{,}6y \tag{1}$$

Código 1: Función a Evaluar

```
def g(x, y):
    px = (cos(x))**2 + 0.3 * x
    py = (sin(y))**2 - 0.6 * y
    return px + py
```

En el código 2 se definen algunos parámetros con los que se evalúa la función, como el rango de los ejes, la resolución con que se le grafica y el valor máximo que pueden tomar los diferenciales de movimiento, Δx y Δy.

Código 2: Parámetros de Evaluación

```
low = -10
high = -low
step = 0.05
p = np.arange(low, high, step)
n = len(p)
z = np.zeros((n, n), dtype=float)
valores=[]
paso = 0.5
```

Para graficar la función en una proyección plana, se necesita evaluar la función en una cantidad finita de puntos, dada por la intrucción `p`, dentro del rango previamente definido, como se observa en el código 3.

Código 3: Puntos para Graficar la Función

```
for i in range(n):
    x = p[i]
    for j in range(n):
        y = p[n - j - 1] # voltear
        z[i, j] = g(x, y)
        valores.append(g(x, y))
```

Por medio del código 4 se definen las coordenadas de los agentes o puntos que realizarán la búsqueda. Además, también se definen las coordenadas iniciales para el mejor valor entre los agentes creados.

Código 4: Creación de Coordenadas para Búsqueda

```
tmax=10
for a in range(10, 31, 10):

    resultados = pd.DataFrame()
    for tiem in range(2, 5):
        agentes =  pd.DataFrame()
        agentes['x'] = [uniform(low, high) for i in range(a)]
        agentes['y'] = [uniform(low, high) for i in range(a)]
        agentes['best'] = [min(valores) for i in range(a)]
        bestx = agentes['x'][0]
        besty = agentes['y'][0]
        best = g(bestx, besty)
```

En el código 5, utilizando los valores de `tmax` y `tiem` definidos anteriormente, se consigue la cantidad de pasos que darán los agentes. Los diferenciales de paso se deciden de manera aleatoria y se agregan a la lista.

Código 5: Diferenciales de Paso

```
        for tiempo in range(tmax**tiem):
            agentes['dx'] = [uniform(0, paso) for i in range(a)]
            agentes['dy'] = [uniform(0, paso) for i in range(a)]
```

Para que cada agente realice movimientos en cada una de las ocho posibles direcciones, se implementa el código 6. Además, se evalúa la función en las ocho direcciones para posteriormente poder determinar cuál presenta un valor mayor. Las líneas `9` a `16` impiden que cualquiera de los agentes realice un movimiento que se encuentre fuera del rango evaluado.

Código 6: Movimiento de los Agentes

```
            for i in range(a):
                r = agentes.iloc[i]

                xl = r.x - r.dx
                xr = r.x + r.dx
                yd = r.y - r.dy
                yu = r.y + r.dy

                if  xl < low+step:
                    xl = r.x
                if  xr > high-step:
                    xr = r.x
                if  yd < low+step:
                    yd = r.y
                if  yu > high-step:
                    yu = r.y

                g1 = g(xl, yu)
                g2 = g(r.x, yu)
                g3 = g(xr, yu)
                g4 = g(xl, r.y)
                g5 = g(xr, r.y)
                g6 = g(xl, yd)
                g7 = g(r.x, yd)
                g8 = g(xr, yd)
                lista = [g1,g2,g3,g4,g5,g6,g7,g8]
```

Por último, se decide la dirección en que la función toma un valor menor y se reescriben las coordenadas de los agentes. Además, se toman las coordenadas del mayor de los valores evaluados de la función y se guardan para poder graficar el punto posteriormente, como se aprecia en el código 7.

Código 7: Maximización de la Función

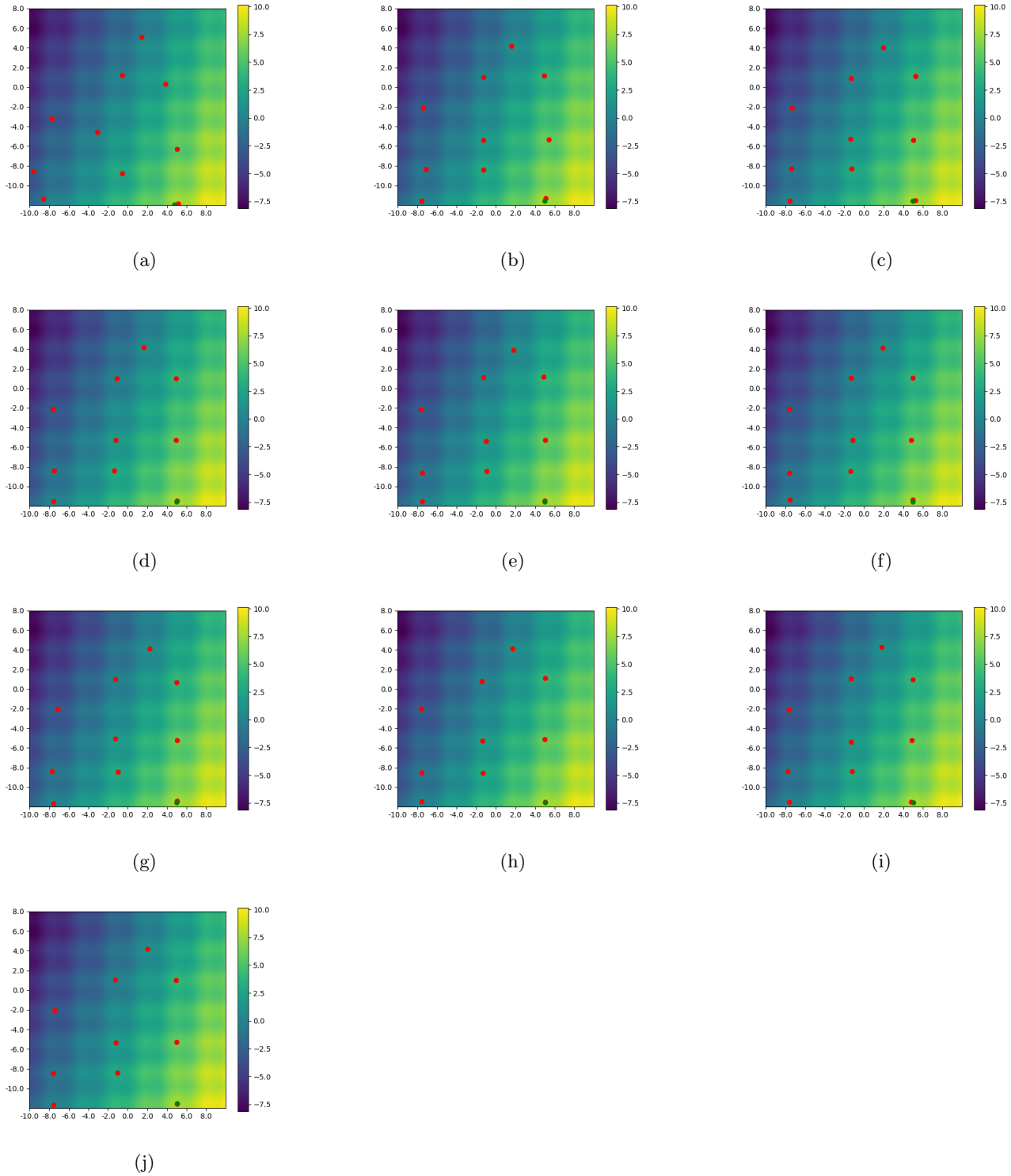
```
                mayor = lista.index(max(lista))+1

                if mayor == 1:
                    agentes.at[i, 'x'] = xl
                    agentes.at[i, 'y'] = yu
                elif mayor ==2:
```

```
                    agentes.at[i, 'x'] = r.x
                    agentes.at[i, 'y'] = yu
                elif mayor ==3:
                    agentes.at[i, 'x'] = xr
                    agentes.at[i, 'y'] = yu
                elif mayor ==4:
                    agentes.at[i, 'x'] = xl
                    agentes.at[i, 'y'] = r.y
                elif mayor ==5:
                    agentes.at[i, 'x'] = xr
                    agentes.at[i, 'y'] = r.y
                elif mayor ==6:
                    agentes.at[i, 'x'] = xl
                    agentes.at[i, 'y'] = yd
                elif mayor ==7:
                    agentes.at[i, 'x'] = r.x
                    agentes.at[i, 'y'] = yd
                elif mayor ==8:
                    agentes.at[i, 'x'] = xr
                    agentes.at[i, 'y'] = yd

                mejor = g(r.x, r.y)
                if mejor > best:
                    best = g(r.x, r.y)
                    bestx = r.x
                    besty = r.y

                if mejor > r.best:
                    agentes.at[i, 'best'] = mejor
```

El desarrollo completo del código puede encontrarse en el repositorio de J. Torres en GitHub [2].

3. Resultados

En su repositorio de GitHub, J. Torres presenta una animación tipo GIF del proceso completo que realizan 10 agentes en 1000 pasos para encontrar los valores máximos de la función evaluada, mientras que en la figura 1 se pueden observar algunos de estos pasos. La proyección plana está codificada por colores, en donde el azul oscuro representa los menores valores y el amarillo brillante representa los mayores valores que toma la función al ser evaluada en el rango definido. El eje horizontal representa los valores de las x, mientras que el eje vertical representa los valores de las y. Cada uno de los puntos rojos que se observan es un agente que realiza un movimiento, Δx y Δy, en cada paso de la iteración, mientras que el punto verde es el máximo valor encontrado en dicho paso.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j)

Figura 1: Movimientos de agentes y maximización de la función.

4. Conclusiones

Como se puede observar en la figura 1, ésta técnica es bastante versátil y logra encontrar los valores máximos (o mínimos) de una función. Sin embargo, presenta algunas dificultades al tratarse de una función oscilatoria, pues los puntos pueden caer en un ciclo en donde no pueden salir de una cresta o un valle y, por lo tanto, no encuentran un máximo o mínimo absoluto.

Referencias

[1] E. Schaeffer. Localsearch, 2019. URL https://github.com/satuelisa/Simulation/tree/master/LocalSearch.

[2] J. Torres. P7, 2022. URL https://github.com/FeroxDeitas/Simulacion-Nano/tree/main/Tareas/P7.