

EJE3DIRCON

April 3, 2023

1 Método de Direcciones conjugadas

```
[1]: import numpy as np
import matplotlib.pyplot as plt

# Definimos la función
def f(x, y):
    return -(4*x**2) - (57*y**2) + 4*x*y - 8*x + 10*y - 2

# Definimos el gradiente de la función
def grad_f(x, y):
    return np.array([-8*x + 4*y - 8, -114*y + 4*x + 10])

# Definimos la base para  $\mathbb{R}^2$ 
B = np.array([[ -10, 8], [ 6, 2]])

# Definimos el método de direcciones conjugadas
def conjugate_directions(point, max_iter=1000, tol=1e-6):
    # Inicializamos los vectores de búsqueda y gradiente
    d = -grad_f(point[0], point[1])
    g = d
    # Inicializamos el contador de iteraciones
    iter = 0
    # Inicializamos el historial de puntos para graficar la trayectoria
    path = [point]
    # Iteramos hasta alcanzar el máximo de iteraciones o la tolerancia
    while iter < max_iter and np.linalg.norm(g) > tol:
        # Calculamos el tamaño de paso óptimo en la dirección de búsqueda
        alpha = np.dot(g, g) / np.dot(np.dot(d, B), np.dot(grad_f(point[0],
        ↪point[1]), B))
        # Actualizamos el punto y el gradiente
        point = point + alpha * np.dot(d, B)
        g_new = np.dot(grad_f(point[0], point[1]), B)
        # Calculamos el coeficiente de corrección beta
        beta = np.dot(g_new, g_new) / np.dot(g, g)
        # Actualizamos el vector de búsqueda
        d = -g_new + beta * d
```

```

    g = g_new
    # Actualizamos el contador de iteraciones y el historial de puntos
    iter += 1
    path.append(point)

    return point, path

# Definimos los parámetros del método
point = np.array([2, -5]) # Punto semilla
max_iter = 1000 # Máximo de iteraciones
tol = 1e-6 # Tolerancia

# Ejecutamos el método de direcciones conjugadas
critical_point, path = conjugate_directions(point, max_iter=max_iter, tol=tol)

# Creamos la figura
fig, ax = plt.subplots(figsize=(8, 6))

# Creamos la malla de puntos para graficar la función
x_range = np.linspace(-10, 10, 100)
y_range = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x_range, y_range)
Z = f(X, Y)

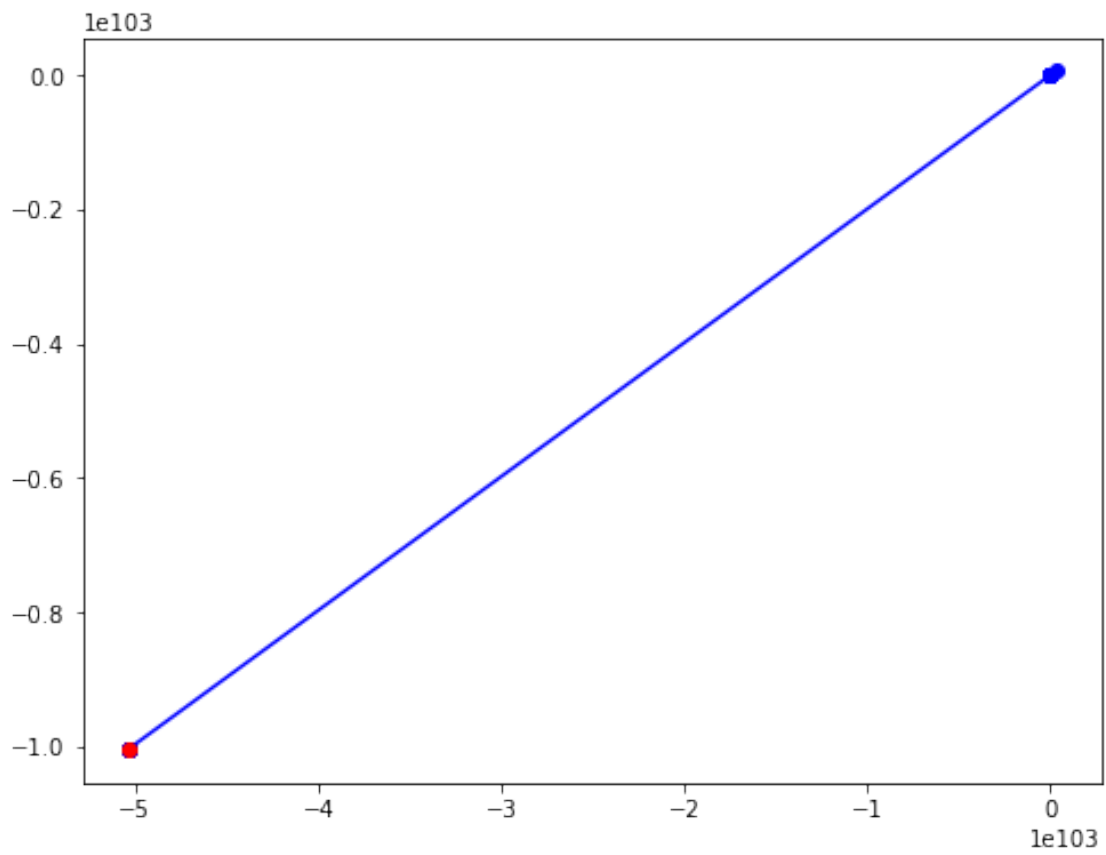
# Graficamos la función
ax.contour(X, Y, Z, levels=20, cmap='coolwarm')

# Graficamos la trayectoria
ax.plot([p[0] for p in path], [p[1] for p in path], 'bo-')

# Para mostrar el punto crítico en la gráfica
ax.plot(critical_point[0], critical_point[1], 'ro')

plt.show()

```



[]: