Facultad de Ciencias

Universidad Autónoma de México

Electromagnetismo II – Proyecto 1

Profesores:

Dr. Alejandro Reyes Coronado

Ayud. Daniel Espinosa González

Ayud. Atzin López Tercero

Alumno: Sebastián González Juárez

sebastian_gonzalezj@ciencias.unam.mx

Proyecto: (100pts) Genera un programa de cómputo (en el lenguaje de tu preferencia) que calcule numéricamente la solución a la ecuación de Laplace en dos dimensiones para un conjunto discreto de puntos, tal y como se muestra en la figura. Los puntos de la frontera exterior se encuentran a un potencial $\phi = 0$, mientras que los puntos de la frontera interna están a un potencial igual a $\phi = 100$.

Te sugiero que antes de ponerte a programar entiendas con detalle cómo es que tiene que ser tu algoritmo. Para esto, te sugiero que resuelvas "manualmente" el problema, con el mallado que se propone en la figura. Puedes comenzar tomando valores arbitrarios para los puntos propuestos en la figura (intersecciones de las líneas punteadas), de manera sensata pues de otra manera tardará en converger tu cálculo, e ir reemplazando de manera sistemática el valor de cada punto por el **promedio de sus cuatro vecinos**.

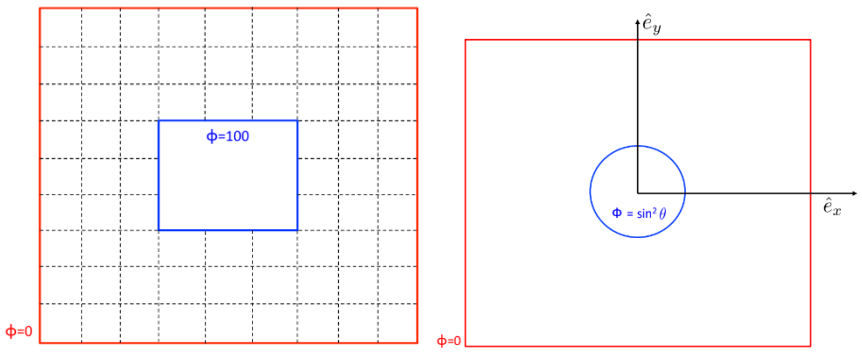
Como siempre, es mejor primero pensar que ponerse a calcular directamente! Pronto te darás cuenta de que los valores propuestos deberán estar entre 0 y 100, y que realmente sólo tienes que determinar siete (7) valores de los puntos en el interior (debido a la simetría), utilizando algún procedimiento sistemático para reemplazar cada punto por el promedio de sus cuatro vecinos inmediatos, repitiendo el proceso hasta que la diferencia entre los valores sucesivos de dos secuencias sea suficientemente pequeño.

Resultado que se espera de tu programa: un archivo en formato pdf con gráficas de las curvas equipotenciales (en tu programa puedes tomar un mallado mucho más fino), y gráficas del campo eléctrico (dibujar las flechitas en tu mallado que representen el campo eléctrico).

Una vez que hayas entendido el algoritmo del problema (que se conoce como *método de relajación*) y hayas practicado con la figura anterior, genera un programa para resolver el problema de un círculo en el interior y un cuadrado al exterior, como se muestra en la figura. El cuadrado exterior se encuentra aterrizado (potencial $\phi = 0$), mientras que el círculo interno tiene un potencial $\phi(\theta) = \sin^2\theta$.

También se espera que grafiques el potencial y el campo eléctrico para este caso. Tendrás que pensar para elegir el mallado que mejor convenga!

Una pregunta que seguramente se te ocurrirá es hasta qué punto debe parar el proceso de iteración. Una idea es repetir el proceso calculando la diferencia en cada punto de tu mallado para la iteración i y la $i-1$, y pedir que la diferencia más grande sea menor que un cierto valor (digamos 1×10^{-6}).



Método de relajación

Este método da un enfoque numérico iterativo y es utilizado en la resolución de ecuaciones diferenciales parciales, como lo son la ecuación de Laplace o la ecuación de Poisson, esto en regiones discretas. Es ampliamente utilizado en problemas donde es necesario encontrar la distribución de potenciales, temperaturas o campos en un dominio, como en el análisis de problemas electrostáticos, transferencia de calor y flujo de fluidos.

Se parte de asumir inicialmente una estimación del campo o potencial en todos los puntos del dominio y luego ir refinando estos valores de forma iterativa. En cada iteración, el valor de una variable en un punto se actualiza como el promedio de los valores en sus puntos vecinos. Este proceso se repite hasta que la solución converge, es decir, hasta que las diferencias entre los valores de iteraciones sucesivas sean lo suficientemente pequeñas.

Trabajaremos en dos dimensiones y en un espacio de cuadricula, la idea es que, en cualquier punto de una cuadrícula, el valor de ϕ puede aproximarse por el promedio de sus cuatro vecinos (arriba, abajo, izquierda y derecha).

Su proceso iterativo se puede resumir como:

1. Inicializa el potencial (o la cantidad de interés) en todos los puntos de la cuadrícula con un valor inicial.
2. Para cada punto de la cuadrícula que no sea un punto de frontera (los bordes donde el valor es conocido), actualiza su valor como el promedio de sus vecinos.
3. Repite el proceso hasta que la solución converja. La condición de convergencia se da cuando la máxima diferencia entre las actualizaciones sucesivas es menor que un valor de tolerancia predefinido.

Nunca había utilizado este método, pero encontré los siguientes usos:

- Electrostática: Resolver la ecuación de Laplace para determinar la distribución de potencial en un campo electrostático.
- Transferencia de calor: Resolver la ecuación de calor para encontrar la distribución de temperatura en una placa o en un sólido.
- Dinámica de fluidos: Resolver ecuaciones de flujo de fluidos en problemas de dinámica de fluidos computacional, como el flujo de aire alrededor de objetos.
- Mecánica de sólidos: Calcular tensiones y deformaciones en estructuras bajo ciertas condiciones de carga.

Antes de pasar a mi proyecto decidí buscar otro código donde se use este método para aclarar mis ideas.

Ejemplo del Método de relajación

Determinar la distribución de temperatura en una placa cuadrada con condiciones de borde 100 grados para el lado izquierdo y 50 grados para el lado derecho. Los bordes de la placa están a temperaturas constantes, el interior de la placa está inicialmente a temperatura cero.

Importamos la siguiente librería:

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

Definamos el espacio de trabajo, en nuestro caso la malla tiene cuadriculas y también propongamos el valor de tolerancia para concluir el ciclo.

```
grid_size = 20
tolerance = 1e-4
```

Hagamos la malla como una matriz de ceros, sus valores iniciales de temperatura.

```
temperature = np.zeros((grid_size, grid_size))
```

Ahora establezcamos las condiciones de frontera, las cuales son la temperatura de los bordes, ajustemos la matriz para estos valores.

```
temperature[:, 0] = 100  # Borde izquierdo a 100 grados
temperature[:, -1] = 50  # Borde derecho a 50 grados
temperature[0, :] = 0  # Borde superior a 0 grados
temperature[-1, :] = 0  # Borde inferior a 0 grados
```

A continuación, el método, para colocar la función completa lo que haremos será explicarla con su documentación en el código.

```
# Función de relajación
def relaxation_step(temperature, tolerance):
    max_diff = tolerance + 1  # Inicializamos con una diferencia mayor a la
tolerancia
    while max_diff > tolerance: #Ciclo que continuara hasta que se cumpla la
tolerancia
        new_temperature = temperature.copy()  # Crear una copia de los
valores actuales
        max_diff = 0  # Inicializar la diferencia máxima

        # Actualizar cada punto interno de la cuadrícula
        for i in range(1, grid_size - 1):
            for j in range(1, grid_size - 1):
                # Actualizar el valor de la temperatura en el punto (i, j)
                new_value = 0.25 * (temperature[i+1, j] + temperature[i-1,
j] +
                                    temperature[i, j+1] + temperature[i, j-
1])
                max_diff = max(max_diff, abs(new_value - temperature[i, j]))
                new_temperature[i, j] = new_value
```

```
                #Podemos notar que acá se obtiene el promedio de los lados
de la cuadricula.

        temperature = new_temperature  # Actualizar la cuadrícula con los
nuevos valores
        print(f"Máxima diferencia en esta iteración: {max_diff}")

    return temperature
```

Simplemente ejecutamos el código asignando nuestras variables.

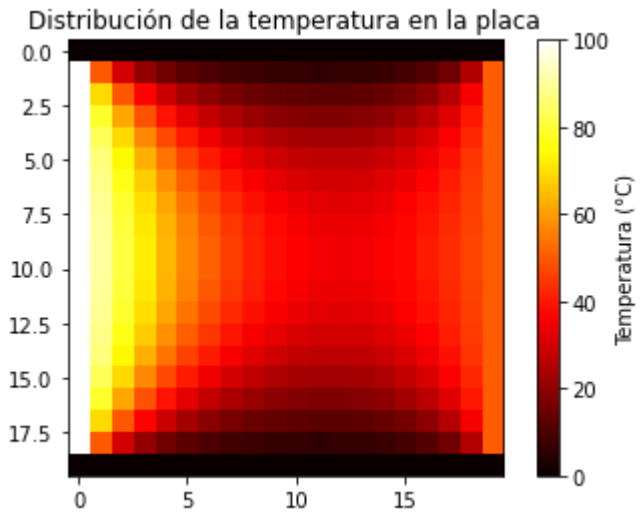
```
final_temperature = relaxation_step(temperature, tolerance)
```

Las líneas que siguen son solo para la realización de la grafica, en las cuales le agregamos color, nombres a los ejes y el título.

```
plt.imshow(final_temperature, cmap='hot', interpolation='nearest')
plt.colorbar(label="Temperatura (°C)")
plt.title("Distribución de la temperatura en la placa")
plt.show()
```

Corramos el código:

```
Máxima diferencia en esta iteración: 25.0
Máxima diferencia en esta iteración: 12.5
Máxima diferencia en esta iteración: 7.8125
Máxima diferencia en esta iteración: 5.46875
Máxima diferencia en esta iteración: 4.6875
Máxima diferencia en esta iteración: 4.0283203125
Máxima diferencia en esta iteración: 3.4912109375
Máxima diferencia en esta iteración: 3.0548095703125
Máxima diferencia en esta iteración: 2.69775390625
Máxima diferencia en esta iteración: 2.4026870727539062
Máxima diferencia en esta iteración: 2.1560192108154297
Máxima diferencia en esta iteración: 1.9481360912322998
Máxima diferencia en esta iteración: 1.8262669444084167
Máxima diferencia en esta iteración: 1.7129983752965927
Máxima diferencia en esta iteración: 1.6098838299512863
Máxima diferencia en esta iteración: 1.513678824994713
Máxima diferencia en esta iteración: 1.427066745236516
Máxima diferencia en esta iteración: 1.3453946099616587
Máxima diferencia en esta iteración: 1.2725002037768718
Máxima diferencia en esta iteración: 1.2028265333356103
Máxima diferencia en esta iteración: 1.1411082349241042
Máxima diferencia en esta iteración: 1.0812817908885108
Máxima diferencia en esta iteración: 1.0286397251576318
Máxima diferencia en esta iteración: 0.9792394385055303
Máxima diferencia en esta iteración: 0.9418137343071713
...
Máxima diferencia en esta iteración: 0.0001030935521413312
Máxima diferencia en esta iteración: 0.00010168748998751198
Máxima diferencia en esta iteración: 0.00010030060506949212
Máxima diferencia en esta iteración: 9.893263509752614e-05
```



Vemos justamente lo esperado del problema, los bordes tal cual se asignaron con diferentes temperaturas y en el centro cada vez les llegaría menos. Aunque al parecer no se actualizaron los otros bordes y solo corrimos la línea donde se asignan que son 0, habría que revisar eso.

Pasemos a lo que de verdad nos interesa, el código del potencial, sin embargo, es muy similar a lo anterior, por eso decidí primero aprender que se necesitaba hacer con otro ejemplo más clásico.

Ya vimos la idea del método y el cómo este va construyendo la malla, ahora plantemos que esperamos de ambas situaciones (el cuadrado y el circulo).

a) Función potencial en el cuadrado interno

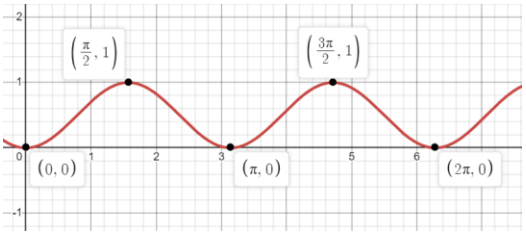
Se dan las condiciones de frontera, las cuales son:

- Los bordes tienen potencial nulo, i.e. $\phi = 0$
- La región central, que es el cuadrado, tiene potencial constante $\phi = 100$

Con eso es suficiente para imaginarnos como el potencial va a decrecer conforme nos alejemos del centro, hasta llegar a 0. Como si dejáramos caer una roca en una alberca, la onda provocada será menos notoria entre más nos alejemos. Evidentemente imaginarlo con fluctuaciones de cuadradas, solo es una idea.

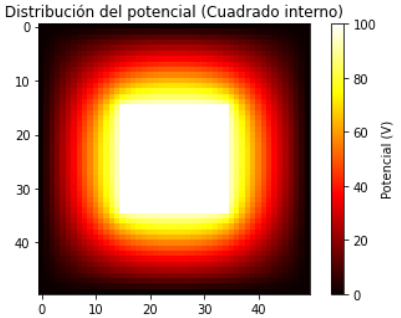
b) Función potencial en el circulo interno

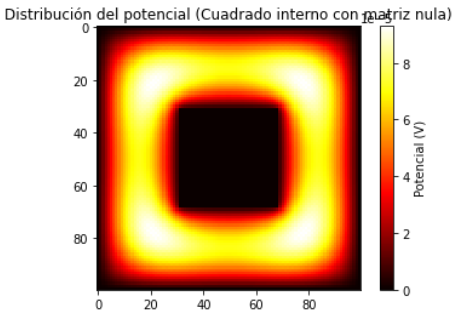
Se espera, como el comportamiento de dicha función $\sin^2\theta$, dos extremos opuestos alcancen el máximo potencial, mientras que los otros dos extremos tengan un potencial nulo. Tal como su grafica la cual muestro a continuación:



Como vemos esta oscila entre los valores 0 y 1, y cada $\frac{\pi}{2}$ se da dicho cambio en los valores. Y, es decir, potencial en el contorno varía con $\sin^2\theta$, el interior del círculo tendrá una distribución de potencial que refleja esta dependencia angular.

A continuación, presento dos ideas de código, uno donde la figura dentro tiene potencial y otro donde solo la frontera.





Código.

Los pasos del código son casi los mismos al ejemplo anterior, por eso fue buena idea hacerlo antes. Importamos la siguiente librería:

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

La última versión del código que compile fue con una malla de 100x100, así que lo dejaré así, luego veremos más a detalle las implicaciones de otras medidas y porque escogí esta.

```
grid_size = 100  # Tamaño de la cuadrícula
```

Cuadrado interno.

Hagamos la malla como una matriz de ceros, i. e. inicializamos la cuadrícula con ceros.

```
potential_square = np.zeros((grid_size, grid_size))
```

Coloquemos sus condiciones de frontera, que son los bordes, los cuales tienen potencial nulo.

```
potential_square[:, 0] = 0  # Borde izquierdo
potential_square[:, -1] = 0  # Borde derecho
potential_square[0, :] = 0  # Borde superior
potential_square[-1, :] = 0  # Borde inferior
```

Definamos el tamaño del cuadrado interno, el cual le podemos hacer modificaciones evidentemente, para mejorar la ilustración. (tomar en cuenta esto al correr el código, evidentemente no puedes poner una malla de 10x10 con un cuadrado de 10x10, pues genera discrepancias). En mi caso la ultima vez que corrí el código probe con 10x10.

```
internal_square_size = 10  # Tamaño del cuadrado interno
```

Empecemos con definiendo las variables donde estaremos dividiendo para sustituir en el método, además con esto ponemos el potencial del cuadrado, el cual tiene un potencial constante de $\phi = 100$.

```
start = (grid_size - internal_square_size) // 2
end = start + internal_square_size
potential_square[start:end, start:end] = 100  # Región interna a ϕ = 100
```

A continuación, el método, para colocar la función completa lo que haremos será explicarla con su documentación en el código.

```
def relaxation_step_square(potential, tolerance=1e-6):
    max_diff = tolerance + 1  # Iniciar con una diferencia mayor al criterio
    while max_diff > tolerance:
        new_potential = potential.copy()  # Crear una copia de los valores
actuales
        max_diff = 0  # Inicializar la diferencia máxima

        # Actualizar cada punto interno de la cuadrícula
        for i in range(1, grid_size - 1):
            for j in range(1, grid_size - 1):
```

```
                if not (start <= i < end and start <= j < end):  # Saltar la
región de ϕ = 100
                    new_value = 0.25 * (potential[i+1, j] + potential[i-1,
j] +
                                        potential[i, j+1] + potential[i, j-
1])
                    max_diff = max(max_diff, abs(new_value - potential[i,
j]))
                    new_potential[i, j] = new_value

        potential = new_potential  # Actualizar la cuadrícula con los nuevos
valores
        print(f"Máxima diferencia en esta iteración (cuadrado): {max_diff}")

    return potential
```

Como en el caso anterior, vemos que imprimirá cada iteración del ciclo, a mi me funciona visualizarlo para ver que el código no se congelara y ver como si va avanzando. Puedes quitar ese print, para solo ver las imágenes al ejecutar.

Para este momento nos damos cuenta lo útil que fue agregar el ejemplo con este método, pues en ejemplo logramos comprender como funcionaba para un caso más sencillo y vemos que es casi idéntica a la ocupada en este nuevo código, solo cambiando que fue acá dentro de la función que se agregó la tolerancia y el salto de la región del potencial constante.

Si decides correr el código, revisar la tolerancia, pues nuevamente, la última vez que corrí el código que es tal cual como lo copio, le puse una tolerancia de 1e-6, si buscas más rapidez tendrás que cambiar eso, o en su defecto aumentarlo para mayor cambio.

Pasemos a la función que genera la gráfica del potencial, la cual no tiene tanta explicación, nombre los ejes y cambie los colores.

```
def plot_potential_square(potential):
    plt.imshow(potential, cmap='hot', interpolation='nearest')
    plt.colorbar(label="Potencial (V)")
    plt.title("Distribución del potencial (Cuadrado interno)")
    plt.show()
```

Ejecutemos el código con las dos funciones.

```
relaxed_potential_square = relaxation_step_square(potential_square)
plot_potential_square(relaxed_potential_square)
```

Después de esto me percate que también pedía graficar el campo eléctrico, obtengámoslo por la relación que tenemos con el potencial.

```
Ex_square, Ey_square = np.gradient(-relaxed_potential_square)
```

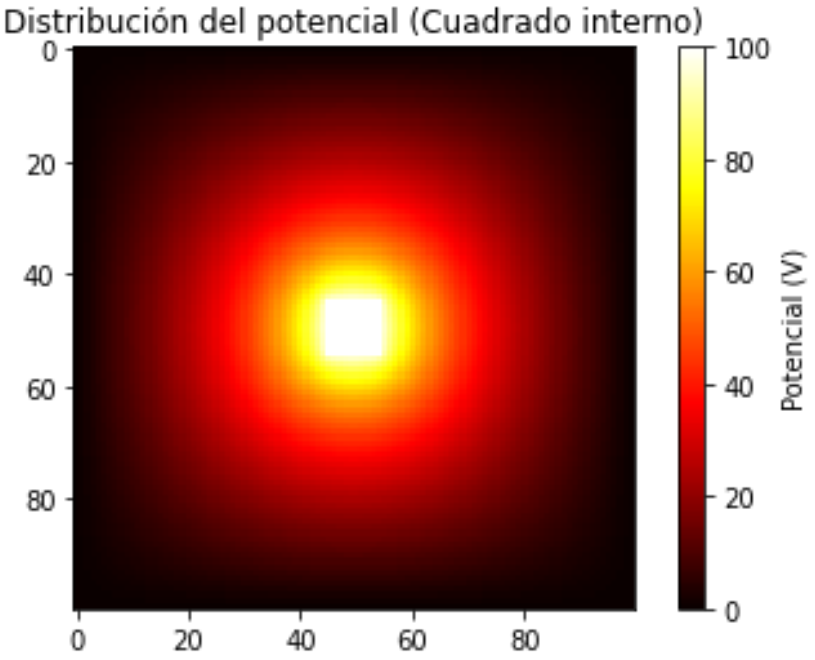
Solo falta hacer su gráfica, tal como se hizo para el potencial,

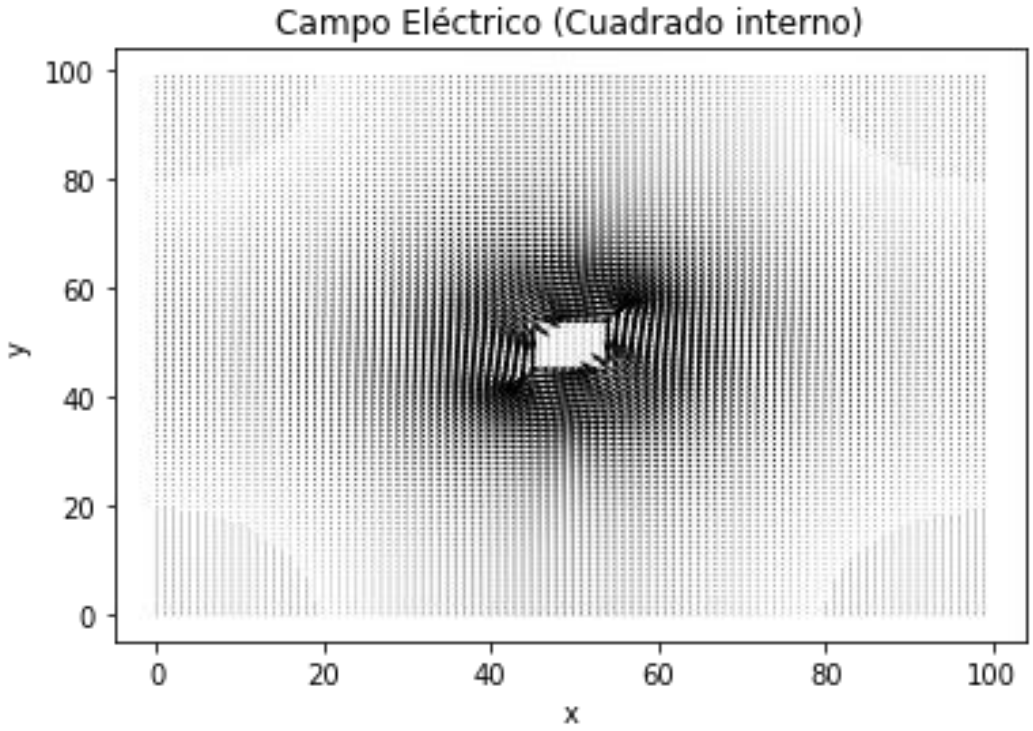
```
def plot_electric_field_square(Ex, Ey):
    plt.quiver(Ex, Ey)
    plt.title("Campo Eléctrico (Cuadrado interno)")
```

```
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```

Ejecutemos esta última función,

```
plot_electric_field_square(Ex_square, Ey_square)
```





Circulo interno.

Continuemos con el código, hagamos la malla como una matriz de ceros, i. e. inicializamos la cuadrícula con ceros. Obs. son las mismas variables de tamaño, que en este caso use 100x100.

```
potential_circle = np.zeros((grid_size, grid_size))
```

Coloquemos sus condiciones de frontera, que son los bordes, los cuales tienen potencial nulo.

```
potential_circle[:, 0] = 0  # Borde izquierdo
potential_circle[:, -1] = 0  # Borde derecho
potential_circle[0, :] = 0  # Borde superior
potential_circle[-1, :] = 0  # Borde inferior
```

Definamos el tamaño del círculo, en mi caso lo hice con radio 10. Recordar cambiar este valor si deseas hacer la cuadricula de la malla más pequeña, para no llegar a incoherencias. Además, ubicamos el centro de la cuadricula.

```
radius = 10  # Radio del círculo interno
center = grid_size // 2  # Centro del círculo en la cuadrícula
```

Como vimos en la gráfica, tendrá puntos máximos y mínimos, estos dependerán de como agarre el ángulo tu programa. A continuación, voy a realizar las particiones de estas mediciones evaluando la función, para asignar el potencial en cada parte del círculo.

```
for i in range(grid_size):
    for j in range(grid_size):
        dist = np.sqrt((i - center)**2 + (j - center)**2)
        if dist <= radius:
            theta = np.arctan2(j - center, i - center)
            potential_circle[i, j] = np.sin(theta)**2  # ϕ(θ) = sin²(θ)
```

Pasemos a la función que llamara al método de relajación, el cual nuevamente es casi el mismo que las anteriores vistas. Le pasamos la tolerancia con los mismos comentarios que dijimos par el caso anterior. La única diferencia es que ahora analiza con base del centro del círculo y su radio, se salta la región ya calculada que es la del propio circulo.

```
def relaxation_step_circle(potential, tolerance=1e-6):
    max_diff = tolerance + 1
    while max_diff > tolerance:
        new_potential = potential.copy()
        max_diff = 0

        for i in range(1, grid_size - 1):
            for j in range(1, grid_size - 1):
                dist = np.sqrt((i - center)**2 + (j - center)**2)
                if dist > radius:  # Saltar la región del círculo
                    new_value = 0.25 * (potential[i+1, j] + potential[i-1,
j] +
                                        potential[i, j+1] + potential[i, j-
1])
```

```
                    max_diff = max(max_diff, abs(new_value - potential[i,
j]))
                    new_potential[i, j] = new_value

        potential = new_potential
        print(f"Máxima diferencia en esta iteración (círculo): {max_diff}")

    return potential
```

Igual, si no quieres que imprima cada iteración, borra esa línea, yo la dejo para ver que no se trabe mi lap.

Pasemos a la función que nos generara el potencial, que es lo mismo que ya hemos explicado.

```
def plot_potential_circle(potential):
    plt.imshow(potential, cmap='hot', interpolation='nearest')
    plt.colorbar(label="Potencial (V)")
    plt.title("Distribución del potencial (Círculo interno)")
    plt.show()
```

Ejecutemos el código con las dos funciones.

```
relaxed_potential_circle = relaxation_step_circle(potential_circle)
plot_potential_circle(relaxed_potential_circle)
```

Calculemos el campo eléctrico.

```
Ex_circle, Ey_circle = np.gradient(-relaxed_potential_circle)
```

Grafiquemos con una función el campo eléctrico.

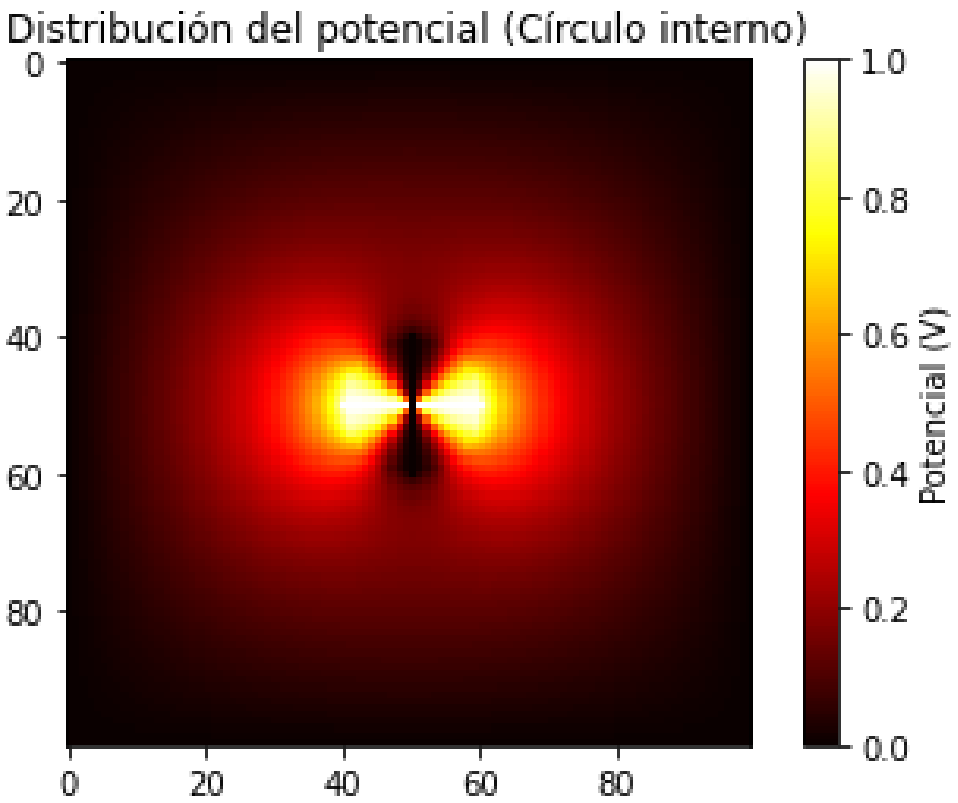
```
def plot_electric_field_circle(Ex, Ey):
    plt.quiver(Ex, Ey)
    plt.title("Campo Eléctrico (Círculo interno)")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```

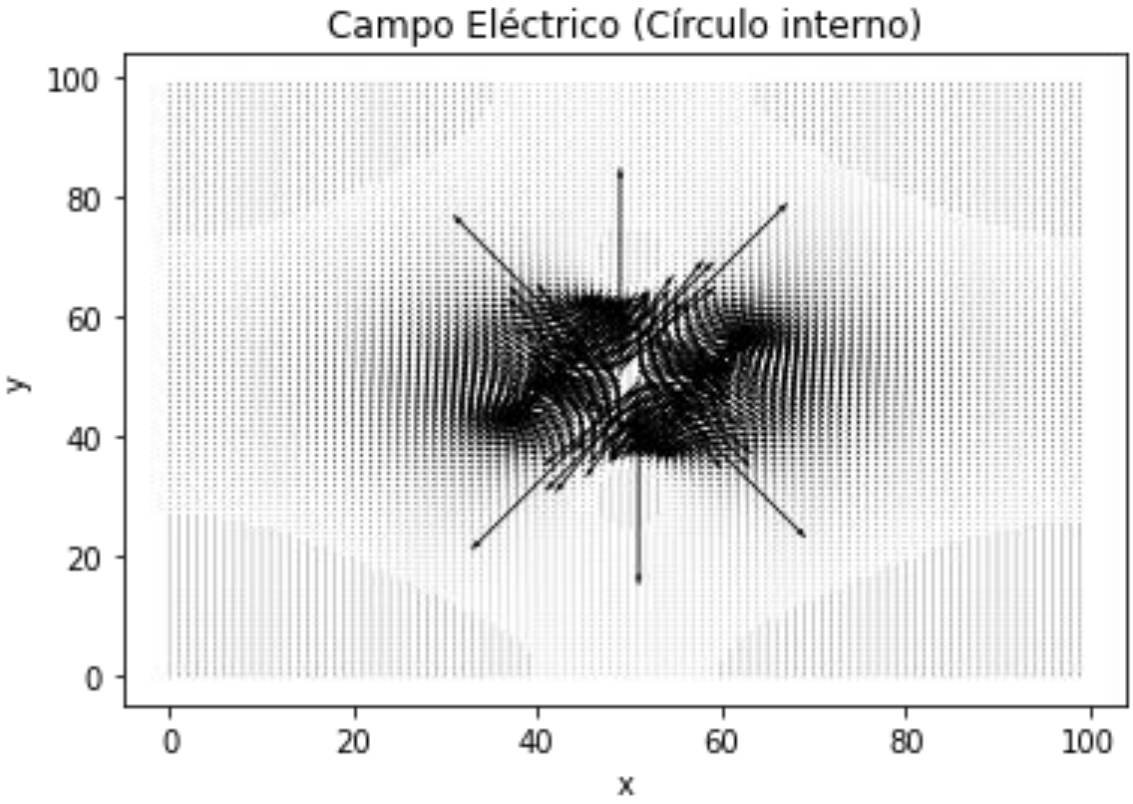
Ejecutemos esta última función,

```
plot_electric_field_circle(Ex_circle, Ey_circle)
```

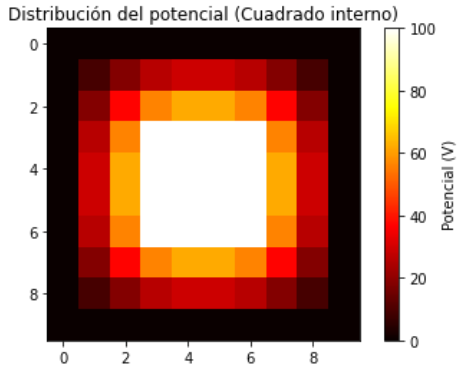
En la siguiente hoja se muestra el resultado.

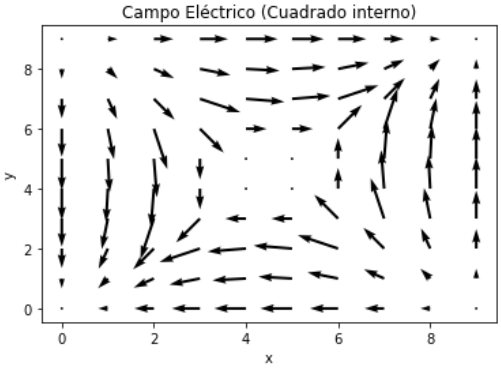
Vemos que las imágenes satisfacen lo que esperábamos con el breve análisis realizado. También ejecutaremos el código con otros valores de las variables para poder apreciar y ejemplificar mejor.

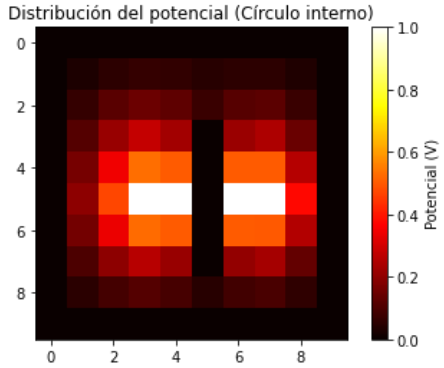

Distribución del potencial (Círculo interno)
0
20
40
60
80
0
20
40
60
80
1.0
0.8
0.6
0.4
0.2
0.0
Potencial (V)

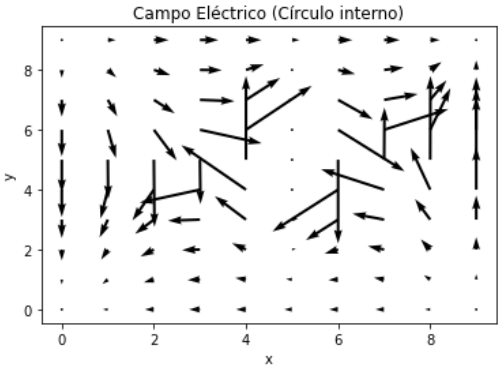

Campo Eléctrico (Círculo interno)
100
80
60
40
20
0
y
0
20
40
60
80
100
x

Malla 10x10 con cuadrado lado 4 y circulo radio 2.

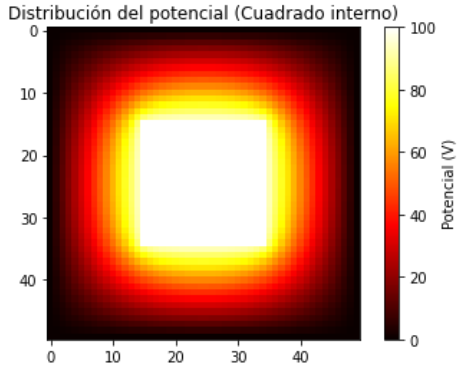


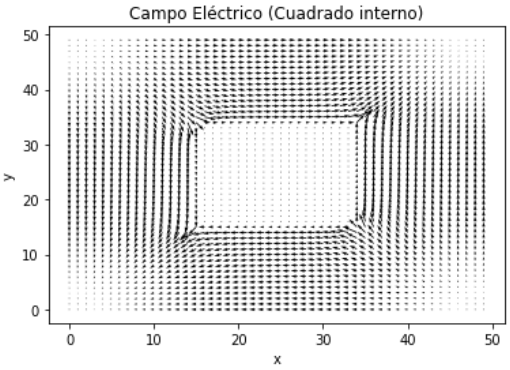


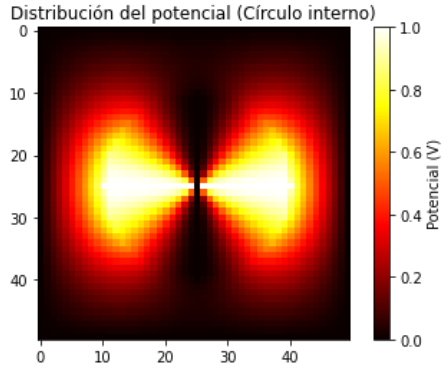


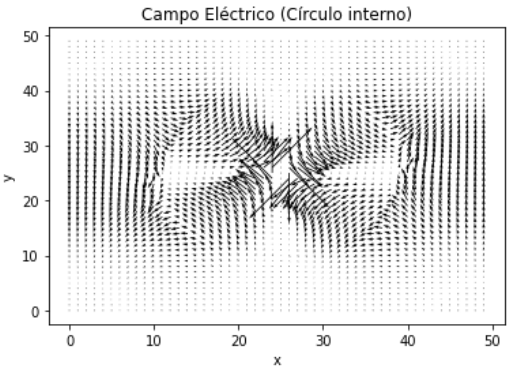


Malla 50x50 con cuadrado lado 20 y circulo radio 15, baje la tolerancia a 1e-4.









A continuación, el código completo documentado,

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

# Definimos el tamaño de la cuadrícula
grid_size = 100  # Tamaño de la cuadrícula

# ---- CASO 1: CUADRADO INTERNO ----

# Inicializamos la cuadrícula con ceros
potential_square = np.zeros((grid_size, grid_size))

# Condiciones de frontera
potential_square[:, 0] = 0  # Borde izquierdo
potential_square[:, -1] = 0  # Borde derecho
potential_square[0, :] = 0  # Borde superior
potential_square[-1, :] = 0  # Borde inferior

# Definir un cuadrado interno con potencial ϕ=100
internal_square_size = 10  # Tamaño del cuadrado interno
start = (grid_size - internal_square_size) // 2
end = start + internal_square_size

potential_square[start:end, start:end] = 100  # Región interna a ϕ = 100

# Función de relajación para el cuadrado
def relaxation_step_square(potential, tolerance=1e-6):
    max_diff = tolerance + 1  # Iniciar con una diferencia mayor al criterio
    while max_diff > tolerance:
        new_potential = potential.copy()  # Crear una copia de los valores
actuales
        max_diff = 0  # Inicializar la diferencia máxima

        # Actualizar cada punto interno de la cuadrícula
        for i in range(1, grid_size - 1):
            for j in range(1, grid_size - 1):
                if not (start <= i < end and start <= j < end):  # Saltar la
región de ϕ = 100
                    new_value = 0.25 * (potential[i+1, j] + potential[i-1,
j] +
                                        potential[i, j+1] + potential[i, j-
1])
                    max_diff = max(max_diff, abs(new_value - potential[i,
j]))
                    new_potential[i, j] = new_value
```

```
        potential = new_potential  # Actualizar la cuadrícula con los nuevos
valores
        print(f"Máxima diferencia en esta iteración (cuadrado): {max_diff}")

    return potential

# Graficar el potencial para el cuadrado
def plot_potential_square(potential):
    plt.imshow(potential, cmap='hot', interpolation='nearest')
    plt.colorbar(label="Potencial (V)")
    plt.title("Distribución del potencial (Cuadrado interno)")
    plt.show()

# Ejecutar el método de relajación para el cuadrado y graficar
relaxed_potential_square = relaxation_step_square(potential_square)
plot_potential_square(relaxed_potential_square)

# Calcular el campo eléctrico para el cuadrado y graficar
Ex_square, Ey_square = np.gradient(-relaxed_potential_square)

def plot_electric_field_square(Ex, Ey):
    plt.quiver(Ex, Ey)
    plt.title("Campo Eléctrico (Cuadrado interno)")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

plot_electric_field_square(Ex_square, Ey_square)

# ---- CASO 2: CÍRCULO INTERNO ----

# Inicializamos la cuadrícula con ceros
potential_circle = np.zeros((grid_size, grid_size))

# Condiciones de frontera (cuadrado exterior a ϕ = 0)
potential_circle[:, 0] = 0  # Borde izquierdo
potential_circle[:, -1] = 0  # Borde derecho
potential_circle[0, :] = 0  # Borde superior
potential_circle[-1, :] = 0  # Borde inferior

# Definir un círculo interno con ϕ(θ) = sin²(θ)
radius = 10  # Radio del círculo interno
center = grid_size // 2  # Centro del círculo en la cuadrícula

# Aplicamos las condiciones de frontera en el círculo interno
```

```
for i in range(grid_size):
    for j in range(grid_size):
        dist = np.sqrt((i - center)**2 + (j - center)**2)
        if dist <= radius:
            theta = np.arctan2(j - center, i - center)
            potential_circle[i, j] = np.sin(theta)**2  # ϕ(θ) = sin²(θ)

# Función de relajación para el círculo
def relaxation_step_circle(potential, tolerance=1e-6):
    max_diff = tolerance + 1
    while max_diff > tolerance:
        new_potential = potential.copy()
        max_diff = 0

        for i in range(1, grid_size - 1):
            for j in range(1, grid_size - 1):
                dist = np.sqrt((i - center)**2 + (j - center)**2)
                if dist > radius:  # Saltar la región del círculo
                    new_value = 0.25 * (potential[i+1, j] + potential[i-1,
j] +
                                        potential[i, j+1] + potential[i, j-
1])
                    max_diff = max(max_diff, abs(new_value - potential[i,
j]))
                    new_potential[i, j] = new_value

        potential = new_potential
        print(f"Máxima diferencia en esta iteración (círculo): {max_diff}")

    return potential

# Graficar el potencial para el círculo
def plot_potential_circle(potential):
    plt.imshow(potential, cmap='hot', interpolation='nearest')
    plt.colorbar(label="Potencial (V)")
    plt.title("Distribución del potencial (Círculo interno)")
    plt.show()

# Ejecutamos el método de relajación para el círculo y graficamos
relaxed_potential_circle = relaxation_step_circle(potential_circle)
plot_potential_circle(relaxed_potential_circle)

# Calcular el campo eléctrico para el círculo y graficarlo
Ex_circle, Ey_circle = np.gradient(-relaxed_potential_circle)
```

```python
def plot_electric_field_circle(Ex, Ey):
    plt.quiver(Ex, Ey)
    plt.title("Campo Eléctrico (Círculo interno)")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

plot_electric_field_circle(Ex_circle, Ey_circle)
```

Si se busca

Su buscamos que dentro de las figuras el potencial sea cero, es decir, los puntos de la frontera interna no tengan potencial, realizamos las siguientes modificaciones al poner el potencial:

Para el cuadrado:

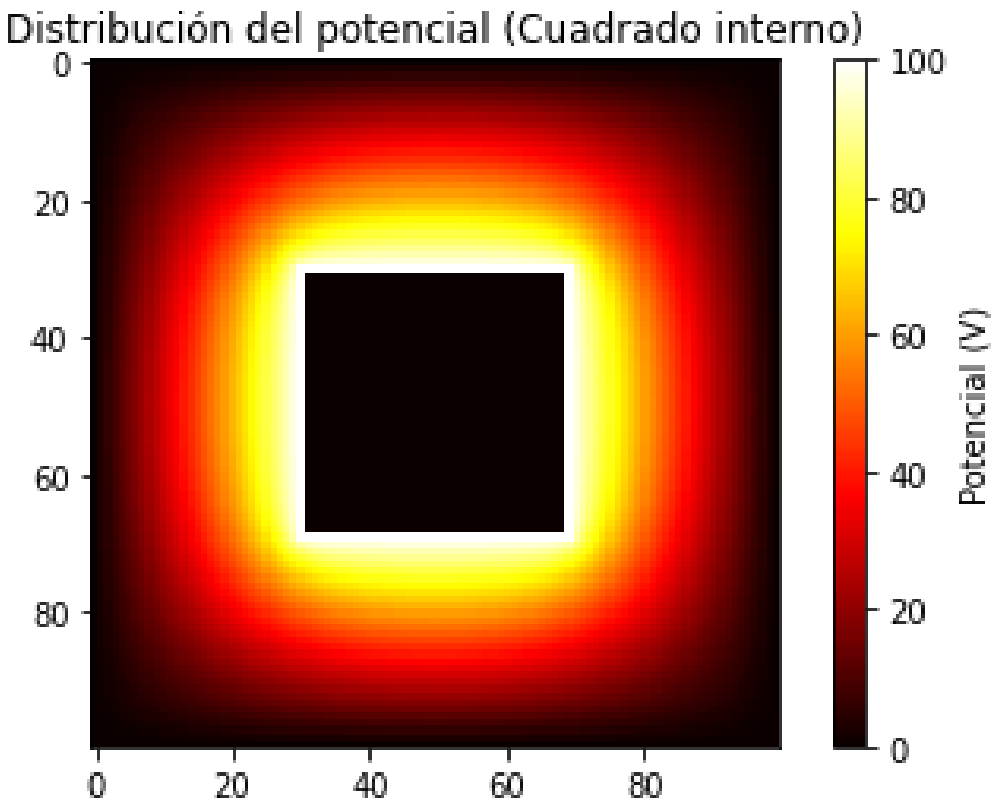
```
potential_square[:, 0] = 0  # Borde izquierdo
potential_square[:, -1] = 0  # Borde derecho
potential_square[0, :] = 0  # Borde superior
potential_square[-1, :] = 0  # Borde inferior
potential_square[start+1:end-1, start+1:end-1] = 0
```

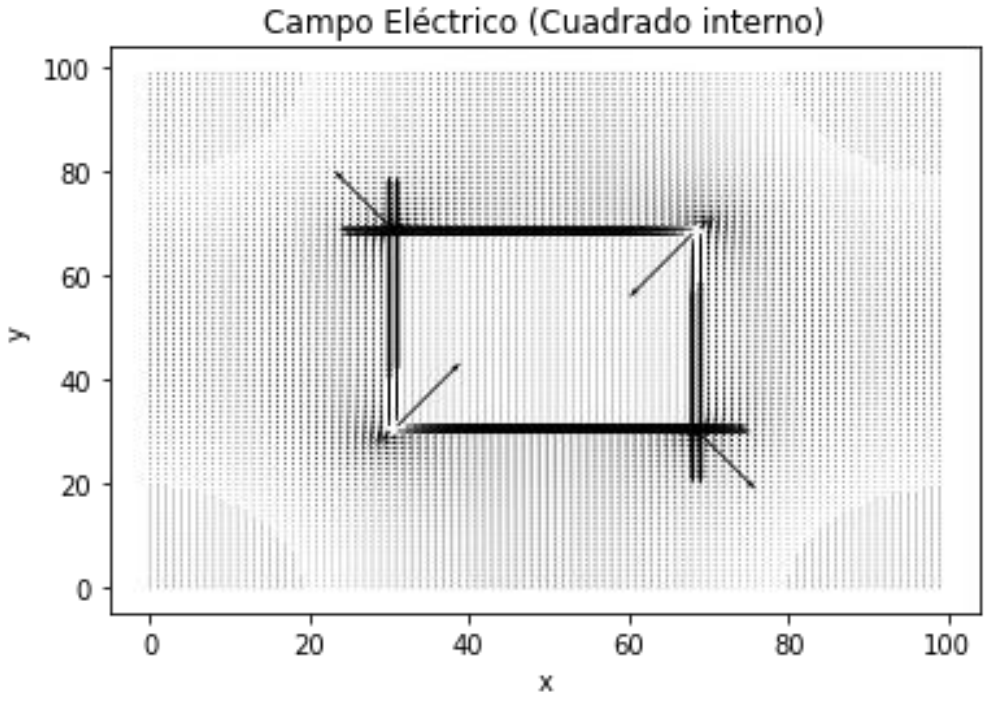
Para el cirulo:

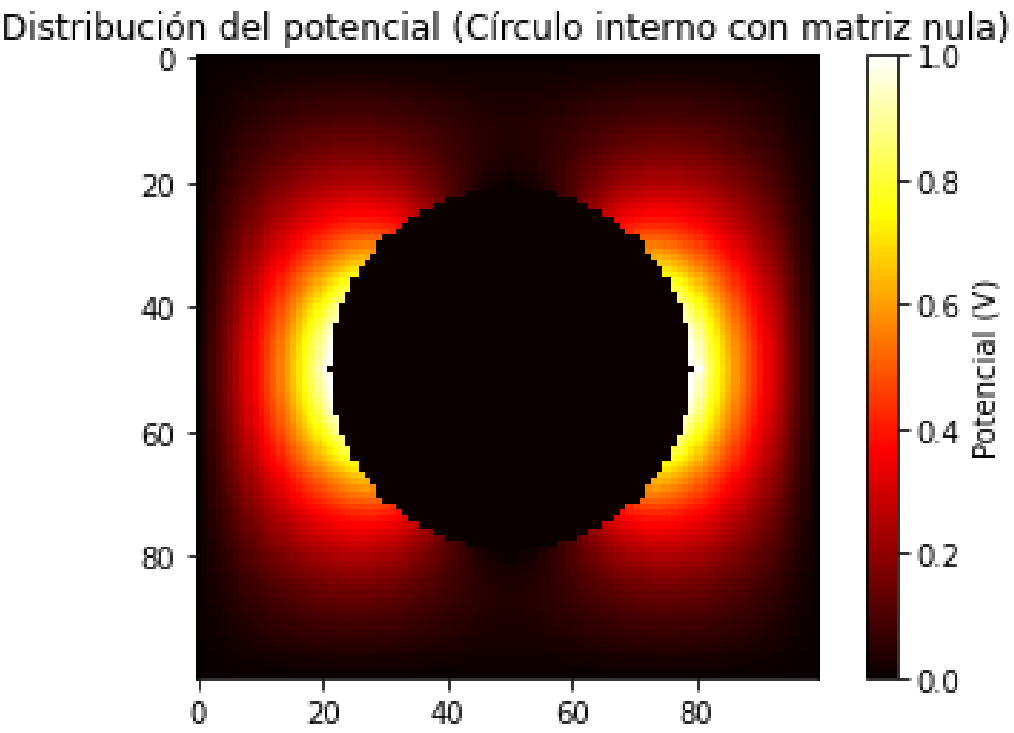
```
for i in range(grid_size):
    for j in range(grid_size):
        dist = np.sqrt((i - center)**2 + (j - center)**2)
        if radius - 1 < dist <= radius:
            theta = np.arctan2(j - center, i - center)
            potential_circle[i, j] = np.sin(theta)**2  # ϕ(θ) = sin²(θ)
```

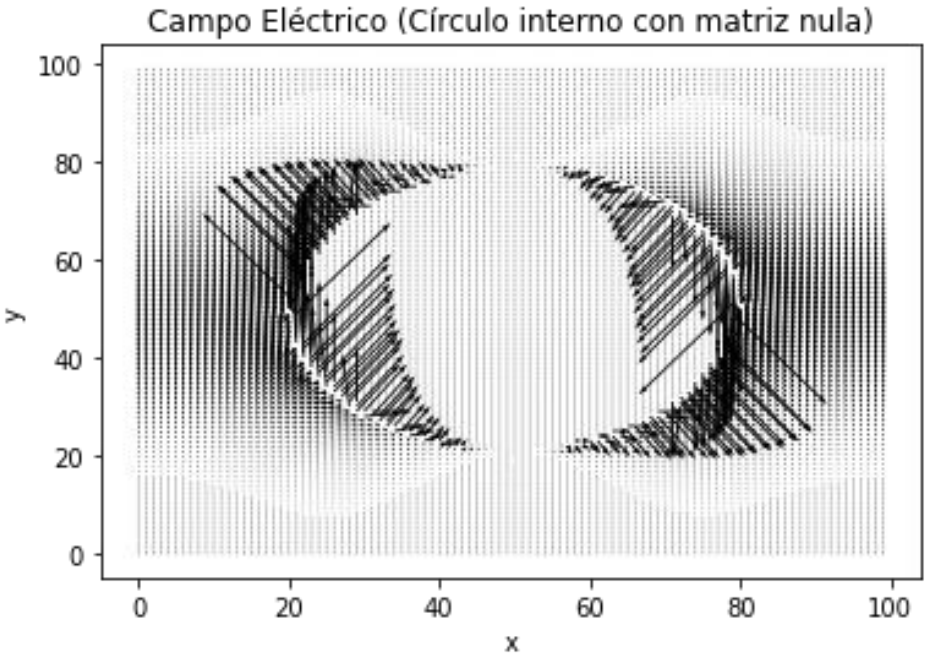
Observamos que lo que se hizo fue solo dejar los valores nulos de la matriz a esa parte.

Lo vamos a correr en la siguiente hoja, tomar en cuenta que fue una malla de 100x100, el cuadrado con lado 40 y el circulo radio 30.


Distribución del potencial (Cuadrado interno)
0
20
40
60
80
0
20
40
60
80
100
80
60
40
20
0
Potencial (V)


Campo Eléctrico (Cuadrado interno)
100
80
60
40
20
0
y
0
20
40
60
80
100
x





Es la misma explicación, solo modificando que en el potencial se asigna solo en los bordes.

A continuación, el código modificado con esas partes y de igual modo documentado.

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

# Definimos el tamaño de la cuadrícula
grid_size = 100  # Tamaño de la cuadrícula

import numpy as np
import matplotlib.pyplot as plt

# Definimos el tamaño de la cuadrícula
grid_size = 100  # Tamaño de la cuadrícula

# ---- CASO 1: CUADRADO INTERNO ----

# Inicializamos la cuadrícula con ceros
potential_square = np.zeros((grid_size, grid_size))

# Condiciones de frontera externas (cuadrícula)
potential_square[:, 0] = 0  # Borde izquierdo
potential_square[:, -1] = 0  # Borde derecho
potential_square[0, :] = 0  # Borde superior
potential_square[-1, :] = 0  # Borde inferior

# Definir un cuadrado interno con potencial solo en los bordes
internal_square_size = 40  # Tamaño del cuadrado interno
start = (grid_size - internal_square_size) // 2
end = start + internal_square_size

# Bordes del cuadrado con potencial ϕ = 100
potential_square[start, start:end] = 100  # Lado superior
potential_square[end-1, start:end] = 100  # Lado inferior
potential_square[start:end, start] = 100  # Lado izquierdo
potential_square[start:end, end-1] = 100  # Lado derecho

# El interior del cuadrado será 0
potential_square[start+1:end-1, start+1:end-1] = 0

# Función de relajación para el cuadrado con interior nulo
def relaxation_step_square(potential, tolerance=1e-4):
    max_diff = tolerance + 1  # Iniciar con una diferencia mayor al criterio
    while max_diff > tolerance:
        new_potential = potential.copy()  # Crear una copia de los valores
actuales
        max_diff = 0  # Inicializar la diferencia máxima
```

```
        # Actualizar cada punto interno de la cuadrícula
        for i in range(1, grid_size - 1):
            for j in range(1, grid_size - 1):
                # Saltar la región del cuadrado interno (mantener 0 en el
interior)
                if not (start <= i < end and start <= j < end):
                    new_value = 0.25 * (potential[i+1, j] + potential[i-1,
j] +
                                        potential[i, j+1] + potential[i, j-
1])
                    max_diff = max(max_diff, abs(new_value - potential[i,
j]))
                    new_potential[i, j] = new_value

        potential = new_potential  # Actualizar la cuadrícula con los nuevos
valores
        print(f"Máxima diferencia en esta iteración (cuadrado): {max_diff}")

    return potential

# Graficar el potencial para el cuadrado
def plot_potential_square(potential):
    plt.imshow(potential, cmap='hot', interpolation='nearest')
    plt.colorbar(label="Potencial (V)")
    plt.title("Distribución del potencial (Cuadrado interno)")
    plt.show()

# Ejecutar el método de relajación para el cuadrado y graficar
relaxed_potential_square = relaxation_step_square(potential_square)
plot_potential_square(relaxed_potential_square)

# Calcular el campo eléctrico para el cuadrado y graficar
Ex_square, Ey_square = np.gradient(-relaxed_potential_square)

def plot_electric_field_square(Ex, Ey):
    plt.quiver(Ex, Ey)
    plt.title("Campo Eléctrico (Cuadrado interno)")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

plot_electric_field_square(Ex_square, Ey_square)


# ---- CASO 2: CÍRCULO INTERNO CON MATRIZ NULA ----
```

```
# Inicializamos la cuadrícula con ceros
potential_circle = np.zeros((grid_size, grid_size))

# Condiciones de frontera (cuadrado exterior a ϕ = 0)
potential_circle[:, 0] = 0  # Borde izquierdo
potential_circle[:, -1] = 0  # Borde derecho
potential_circle[0, :] = 0  # Borde superior
potential_circle[-1, :] = 0  # Borde inferior

# Definir un círculo interno con ϕ(θ) = sin²(θ) solo en la frontera
radius = 30  # Radio del círculo interno
center = grid_size // 2  # Centro del círculo en la cuadrícula

# Aplicamos las condiciones de frontera solo en el borde del círculo
for i in range(grid_size):
    for j in range(grid_size):
        dist = np.sqrt((i - center)**2 + (j - center)**2)
        if radius - 1 < dist <= radius:
            theta = np.arctan2(j - center, i - center)
            potential_circle[i, j] = np.sin(theta)**2  # ϕ(θ) = sin²(θ)

# Función de relajación para el círculo con matriz nula en el centro
def relaxation_step_circle(potential, tolerance=1e-6):
    max_diff = tolerance + 1
    while max_diff > tolerance:
        new_potential = potential.copy()
        max_diff = 0

        for i in range(1, grid_size - 1):
            for j in range(1, grid_size - 1):
                dist = np.sqrt((i - center)**2 + (j - center)**2)
                if dist > radius:  # Saltar la región del círculo interno
                    new_value = 0.25 * (potential[i+1, j] + potential[i-1,
j] +
                                        potential[i, j+1] + potential[i, j-
1])
                    max_diff = max(max_diff, abs(new_value - potential[i,
j]))
                    new_potential[i, j] = new_value

        potential = new_potential
        print(f"Máxima diferencia en esta iteración (círculo): {max_diff}")

    return potential
```

```python
# Graficar el potencial para el círculo
def plot_potential_circle(potential):
    plt.imshow(potential, cmap='hot', interpolation='nearest')
    plt.colorbar(label="Potencial (V)")
    plt.title("Distribución del potencial (Círculo interno con matriz
nula)")
    plt.show()

# Ejecutamos el método de relajación para el círculo y graficamos
relaxed_potential_circle = relaxation_step_circle(potential_circle)
plot_potential_circle(relaxed_potential_circle)

# Calcular el campo eléctrico para el círculo y graficarlo
Ex_circle, Ey_circle = np.gradient(-relaxed_potential_circle)

def plot_electric_field_circle(Ex, Ey):
    plt.quiver(Ex, Ey)
    plt.title("Campo Eléctrico (Círculo interno con matriz nula)")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

plot_electric_field_circle(Ex_circle, Ey_circle)
```