ARTÍCULO ORIGINAL

Revista de Investigación de Física **24(2)**, (Jul-Dic 2021)

Doi: 10.15381/rif.v24i2.20730

eISSN:1728-2977

Solución analítica y numérica de la ecuación de Laplace utilizando Python

Yhony Mamani A. [*1], J. Chacaliaza-Ricaldi[1], A. Leandro-Pelaez[1], O. Lanchipa[1] y Juan A. Ramos-Guivar[1,2]

[1] *Universidad Nacional Mayor de San Marcos, Unidad de Posgrado de Física, Lima, Perú*

[2] *Universidad Nacional Mayor de San Marcos, Facultad de Ciencias Físicas, Grupo de Investigación de Nanotecnología Aplicada para Biorremediación Ambiental, Energía, Biomedicina y Agricultura (NANOTECH), Lima Perú.*

Recibido 30 Jun 2021 – Aceptado 20 Jul 2021 – Publicado 27 Jul 2021

Resumen

La solución de la ecuación de Laplace dentro de un curso de Electrodinámica Clásica es fundamental para el entendimiento de los problemas de contorno en Electrostática. En el presente trabajo de investigación se obtiene la solución analítica y numérica por el método de relajación, de un problema de placas paralelas infinitas, de la ecuación de Laplace en coordenadas cartesianas en un espacio euclidiano bi-dimensional, específicamente usando el lenguaje de programación Python, para su uso en la enseñanza de la Física y su mejor comprensión fenomenológica. Así mismo, se compara y discute las soluciones obtenidas por ambas técnicas de solución.

Palabras clave: Ecuación de Laplace, Método de Relajación, Fisica Computacional, Problemas de Contorno.

Abstract

The solution of Laplace's equation within a Classical Electrodynamics course is fundamental for the understanding of boundary problems in Electrostatics. In the present research work the analytical and numerical solution is obtained by the relaxation method, of a problem of infinite parallel plates, of the Laplace equation in Cartesian coordinates in an Euclidean space of two dimensions, specifically using the Python programming language, for its use in the teaching of Physics and its better phenomenological understanding. Likewise, the solutions obtained by both solution techniques are compared and discussed.

Keywords: Laplace Equation, Relaxtion Method, Computational Physics, Boundaries Condition Problems.

Introducción

Las ecuaciones de Laplace y Poisson representan a las ecuaciones diferenciales parciales elípticas fundamentales (EDPs) de la Física Matemática EDPs [1]. De hecho, la teoría general de la solución de la ecuación de Laplace se conoce como teoría de potenciales vectoriales. La solución a esta ecuación se conoce como funciones armónicas, y estas funciones se destacan en diferentes campos como: Electrostática, Astronomía, Dinámica de Fluidos, Flujo Estacionario de Calor [2]. En Física la mayoría de problemas que involucran la resolución de ecuaciones diferenciales (EDOs) aparecen en casi todas las asignaturas de la especialidad, las cuales en muchos casos no cuentan con soluciones matemáticas exactas y son restringidas a través de condiciones de contorno y problemas de valores iniciales. Por tanto, debemos enfocar el problema en cuestión desde otro punto de vista. Es ahi donde aparecen los metodos numéricos como una potente herramienta capaz de resolver las EDOs y EDPs, donde nos permite encontrar el valor aproximado de la función incógnita para un conjunto de valores de las variables independientes discretas. Que pueden ser implementados en diferentes lenguajes y técnicas de programación que un ordenador puede interpretar [3]. Una forma de desarrollar la solución de la ecuación de Laplace es el método de separación de variables. Para tal caso, primeramente se establece el sistema de coordenadas en que se trabajará, dependiendo de la geometría del problema bajo estudio. Luego, las EDPs se reducen a un conjunto de EDOs y finalmen-

[*] yhony.mamani@unmsm.edu.pe

te las condiciones de contorno, Neumann o Dirichlet, se satisfacen superponiendo las soluciones encontradas por separación de variables. Otras técnicas para resolver estas EDOs son: el método de imágenes, expansiones multipolares, funciones de Green y variable compleja. Estos métodos resuelven analíticamente la ecuación de Laplace o Poisson y sólo tienen éxito cuando el problema en cuestión tiene ciertos grados de simetría [4]. En este caso, existen diversos métodos numéricos para la solución numérica de la ecuación de Laplace, entre los cuales tenemos: Las diferencias finitas (DF) [5], el método de Crank-Nicholson [6] y el método de relajación [7–9]. Estos dos últimos, son flexibles en las condiciones de contorno y condiciones iniciales del problema.

En la siguiente investigación, se ha abordado la solución analítica, por el método de separación de variables, y numérica, por el método de relajación, de la ecuación de Laplace, utilizando un problema particular de la Electrodinámica Clásica de dos placas conductoras de metal infinitas. Para ello, se ha hecho uso del lenguaje de Programación Python (version 3.7), cuyo entorno se desarrolló en Jupyter Lab de Anaconda y se hizo uso de las bibliotecas: **sympy, numpy, matplotlib**. Además, se discute la implementación de la solución numérica paso a paso hasta la obtención gráfica del potencial electrostático bajo ciertas condiciones de contorno establecidas en el código computacional. Así mismo, se hace una comparación de la solución analítica y la solución numérica obtenida por el método de relajación.

Metodología y simulación numérica

El problema elegido de manera didáctica para la solución numérica de la EDP de Laplace enuncia lo siguiente: "Dos placas de metal infinitas conectadas a tierra se encuentran paralelas al plano xz, una en $y = 0$, la otra en $y = a$ como se muestra en la Figura 1.

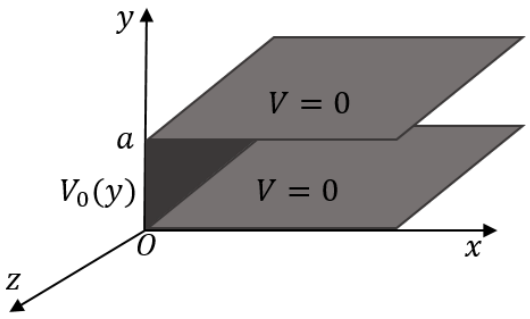

Figura 1: Placas conductoras infinitas conectadas a través de un aislante infinito.

El extremo izquierdo, en $x = 0$, se cierra con una tira aislante infinita entre las dos placas y se mantiene a un potencial específico $V_0(y)$. El problema solicita encontrar el potencial dentro de esta ranura".

Con condiciones de contorno:

1. $V = 0$ cuando $y = 0$
2. $V = 0$ cuando $y = a$
3. $V = V_0(y)$ cuando $x = 0$
4. $V \to 0$ como $x \to \infty$

Solución analítica

Para solucionar este problema hacemos uso de la ecuación de Laplace y por el método de separación de variables. Dado que la solución es independiente de z, en realidad es un problema 2D y, por lo tanto:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0 \tag{1}$$

El método de separación de variables facilita encontrar las soluciones en forma de productos:

$$V(x, y) = X(x)Y(y) \tag{2}$$

Reemplazando la ecuación (2) en la ecuación (1), se tiene:

$$Y\frac{d^2X}{dx^2} + X\frac{d^2Y}{dy^2}$$

Dividiendo por V, tenemos:

$$\frac{1}{X}\frac{d^2X}{dx^2} + \frac{1}{Y}\frac{d^2Y}{dy^2} = 0 \tag{3}$$

la cual tiene una correspondencia funcional del tipo::

$$f(x) + g(y) = 0$$

lo cual solo es posible si f y g son *constantes*. Entonces, el primer término y el segundo término del primer miembro de la ecuación (3) serán constnates:

$\frac{1}{X}\frac{d^2X}{dx^2} = C_1$ y $\frac{1}{Y}\frac{d^2Y}{dy^2} = C_2$ con $C_1 + C_2 = 0$

Entonces, C_1 o C_2 tiene que ser negativo (o ambos son cero). Para este caso hacemos que C_1 sea positivo (y por lo tanto C_2 negativo).

Entonces, convertimos las EDPs en dos EDOs:

$$\frac{d^2X}{dx^2} = k^2X \tag{4}$$

con k^2 siempre positivo, y

$$\frac{d^2Y}{dy^2} = -k^2Y \tag{5}$$

con $-k^2$ siempre negativo.

La solución de las dos EDOs (4) y (5) son fáciles de resolver. Sin embargo, las herramientas computacionales nos permiten solucionar de forma más rápida. En ese entender, implementamos el código para la solución de las ecuaciones (4) y (5) haciendo uso de la biblioteca **sympy**, que es una herramienta para el manejo computacional de sistemas algebraicos (CAS por sus siglas en inglés, *Computer Algebra System*), y algunas funciones que estan dentro de la biblioteca **sympy**, tal como se muestra en el siguiente código:

[1]:
```
from sympy.interactive import printing
printing.init_printing(use_latex=True)
from sympy import Function, dsolve, Eq,␣
↪Derivative, sin, cos, symbols, simplify,␣
↪real_roots
from sympy.abc import x,y
```

[2]:
```
X = Function('X',real=True)
k = symbols('k',positive=True)
f1_ode = Eq(Derivative(X(x), x, 2) -␣
↪k**2*X(x))
f1_ode
```

[2]:
$$-k^2X(x) + \frac{d^2}{dx^2}X(x) = 0$$

[3]:
```
dsolve(f1_ode, X(x))
```

[3]:
$$X(x) = C_1e^{-kx} + C_2e^{kx}$$

[4]:
```
Y = Function('Y',real=True)
f2_ode = Eq(Derivative(Y(y), y, 2) +␣
↪k**2*Y(y))
f2_ode
```

[4]:
$$k^2Y(y) + \frac{d^2}{dy^2}Y(y) = 0$$

[5]:
```
dsolve(f2_ode, Y(y))
```

[5]:
$$Y(y) = C_1\sin(ky) + C_2\cos(ky)$$

Ahora reemplazamos, las soluciones obtenidas en la ecuación (2), tenemos:

$$V(x,y) = \left(Ae^{kx} + Be^{-kx}\right)(C\sin ky + D\cos ky)$$

De las condiciones de contorno podemos obtener las constantes A, B, C y D.
De la condición 4, se tiene; $V \to 0$ como $x \to \infty$. Entonces: $A = 0$ y V queda:

$$V(x,y) = e^{-kx}(C/B\sin ky + D/B\cos ky)$$

De la condicion 1, se tiene; $V = 0$ cuando $y = 0$. Entonces: $D/B = 0$ y V queda:

$$V(x,y) = C/Be^{-kx}\sin ky$$

De la condición 2, se tiene; $V = 0$ cuando $y = a$. Entonces: $\sin ka = 0$ y de aquí k, será:

$$k = \frac{n\pi}{a}, \quad (n = 1, 2, 3, \ldots)$$

De la condición 3, tenemos $V = V_0(y)$ cuando $x = 0$ esto nos da la solución para un $V_0(y) \propto \sin(n\pi y/a)$ específico.

La separación de variables nos da un conjunto infinito de soluciones (una para cada n), y mientras que ninguna de ellas por sí sola satisface la condición de frontera 3, es posible combinarlas en una sola solución. Entonces, sabiendo que la ecuación de Laplace es lineal y por lo tanto:

$$\nabla^2V = \alpha_1\nabla^2V_1 + \alpha_2\nabla^2V_2 + \ldots = 0\alpha_1 + 0\alpha_2 + \ldots = 0$$

con α_1, α_2, ... constantes arbitrarias.

Teniendo este hecho, podemos construir una solución más general para el potencial eléctrico, que nos da todas las posibles soluciones para un $V_0(y)$ arbitrario

$$V(x,y) = \sum_{n=1}^{\infty} C_n e^{-n\pi x/a}\sin(n\pi y/a)$$

que satisface las tres primeras condiciones de contorno.

De la condición 3 cuando $x = 0$, tenemos:

$$V(0,y) = \sum_{n=1}^{\infty} C_n\sin(n\pi y/a) = V_0(y)$$

Para encontrar los coeficientes C_n, multiplicando $V(0,y)$ por $\sin(n'\pi y/a)$ con n' un entero positivo e integrando de 0 a a:

$$\sum_{n=1}^{\infty} C_n \int_0^a \sin(n\pi y/a)\sin(n'\pi y/a)\,dy = \int_0^a V_0(y)\sin(n'\pi y/a)\,dy$$

Para solucionar la integral del primer miembro utilizamos la biblioteca **sympy** importando paquetes necesarios, y obtenemos dos posibles soluciones, estas son:

[6]:
```
from sympy import integrate, pi
n, m = symbols('n m', positive = True,␣
↪integer=True)
a = symbols('a', constant = True)
integrate (sin(n*pi*y/a)*sin(m*pi*y/a), (y,␣
↪0, a))
```

[6]:
$$\begin{cases} 0 & \text{para } m \neq n \\ \frac{a}{2} & \text{para } m = n \end{cases}$$

y por lo tanto, todos los términos se eliminan, excepto para $m = n = n'$, por lo que obtenemos los coeficientes C_n:

$$C_n = \frac{2}{a}\int_0^a V_0 \sin(n\pi y/a)\, dy$$

Ilustremos ahora un ejemplo, para ver como se comporta el potencial eléctrico cuando $x = 0$, es decir, en la tira aislante con potencial constante V_0, la constante C_n queda:

$$C_n = \frac{2V_0}{a}\int_0^a \sin(n\pi y/a)dy = \frac{2V_0}{n\pi}(1 - \cos n\pi)$$

$$C_n = \begin{cases} 0, & \text{si } n \text{ es par.} \\ \frac{4V_0}{n\pi}, & \text{si } n \text{ es impar.} \end{cases}$$

Finalmente, el potencial eléctrico cuando n es par, será:

$$V(x,y) = \frac{4V_0}{\pi}\sum_{n=1,3,5,\ldots}\frac{1}{n}\mathrm{e}^{-n\pi x/a}\sin(n\pi y/a) \qquad (6)$$

Para visualizar el comportamiento del potencial eléctrico, ecuación (6), en las placas paralelas conductoras, implementamos el código. Para ello importamos la biblioteca **matplotlib** y la biblioteca **numpy** que permite graficar funciones, y crear vectores y matrices multidimensionales para almacenar datos, respectivamente, como se muestra en la entrada (7) del script Python en el apéndice. De igual manera, declaramos el rango de variación de x, y, y para el caso cuando $n = 1$, $a = 30$, $V_0 = 1$ V en la entrada (8) y (10) del script, respectivamente.

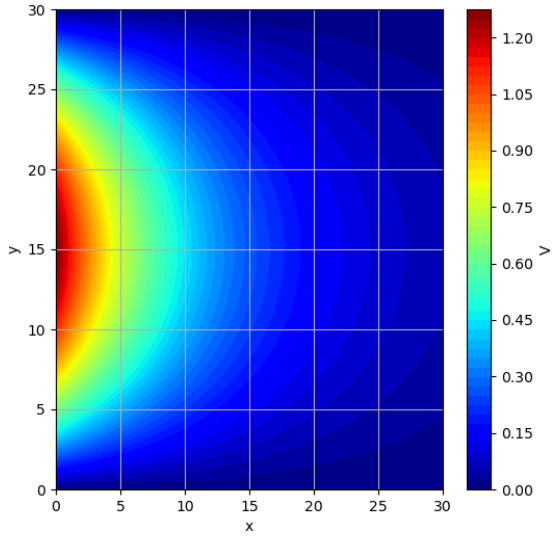

Figura 2: Comportamiento del potencial eléctrico en el contorno cuando $n = 1$.

En la entrada (10) del script implementamos para ver como se comporta el potencial eléctrico $V(x, y)$ en el contorno, como se muestra en la Figura 2.

Para ver el comportamiento del potencial eléctrico en 3D, importamos **Axes3D** de la biblioteca **mpl.toolkits.mplot3d** que nos permite visualizar gráficas en 3 dimensiones, como se muestra en la entrada (11) del script Python, teniendo como salida la Figura 3.

El potencial eléctrico depende de x, y y n (ecuación (6)). Para visualizar el comportamiento del potencial dentro de las placas paralelas cuando $n = 1, 2, 3, ..., 27$, hacemos uso de la herramienta **animation**, que esta dentro de la biblioteca **matplotlib** y la herramienta de visualización de funciones (**IPython.display**), como se muestra en la entrada (12) del script Python.

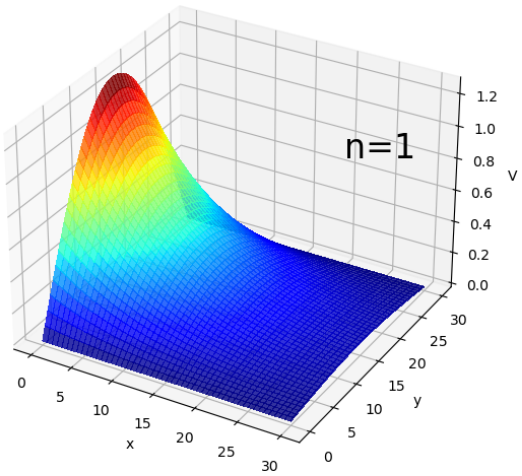

Figura 3: Comportamiento del potencial en las placas paralelas conductoras cuando $n = 1$.

Solución por el Método de Relajación

En esta sección se presenta la solución del problema planteado por el método de relajación y de manera que se llegue a la implementación computacional. El método de relajación es una técnica matemática de resolución numérica que se estudia en análisis numérico. En nuestro problema de dos dimensiones la técnica de relajación consiste en reemplazar la superficie continua de puntos en la que nos interesa calcular el potencial eléctrico, por una red de puntos, limitada por las condiciones de contorno del problema, pero a los puntos que no son de contorno tenemos que asignarles valores iniciales del potencial eléctrico. El método de relajación consiste en reemplazar el valor inicial del potencial en un punto libre determinado por el valor medio de sus vecinos como se tiene en la ecuación(7) [9]:

$$V_{i,j} = (V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1})/4 \qquad (7)$$

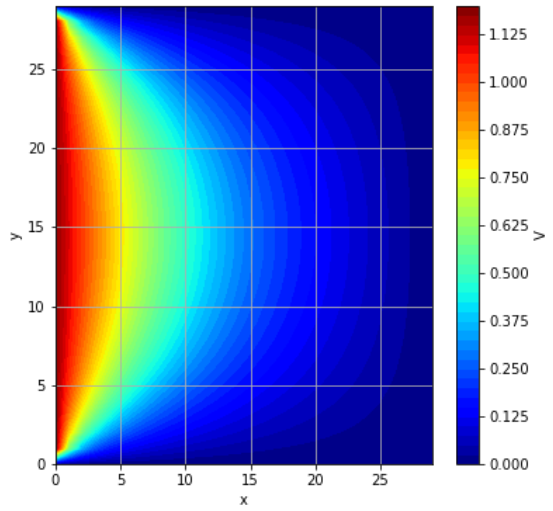

Figura 4: Comportamiento del potencial eléctrico en el contorno de las placas paralelas, por el método de relajación.

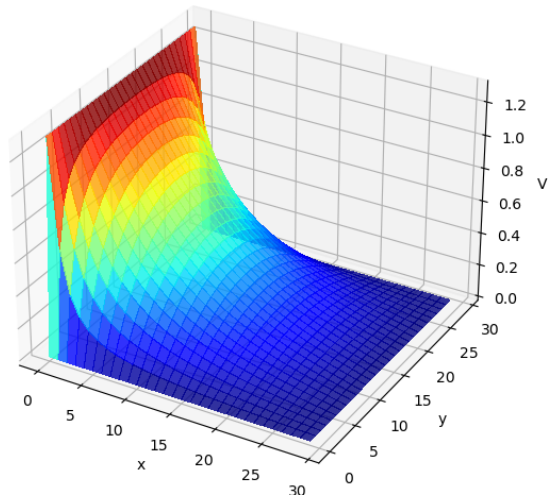

Figura 5: Comportamiento del potencial eléctrico en las placas paralelas, por el método de relajación.

Está es la ecuación principal que nos permite aproximar el potencial eléctrico mediante iteraciones. Para nuestro caso de las placas paralelas, Figura 1, con condiciones de contorno mencionadas. Implementamos el código, de la misma forma como para el método analítico, importamos las bibliotecas **matplotlib** y **numpy**. Así mismo, asignamos valores iniciales para $V_o = 1\ V$ (V0) en la cinta aislante, el potencial inicial (V_i), y las longitudes de x (lonX) y y (lonY), y las condiciones de contorno que exige el problema, como se muestra en el apendice del script Python.

Teniendo como salida, el comportamiento del potencial eléctrico en el contorno de las placas paralelas, como se muestra en la Figura 4.

De la misma forma para el potencial eléctrico de las placas paralelas en 3D, como se muestra en la Figura 5.

Resultados y Discusión

En la expresión para el potencial eléctrico (ecuación (6)) tenemos una dependencia del inverso de n acoplada con la exponencial que depende de n y x, además con la función seno que depende de n y y. El comportamiento del potencial cuando $x = 0$, es decir en V_0, será sinusoidal, como se muestra en la Figura 6.

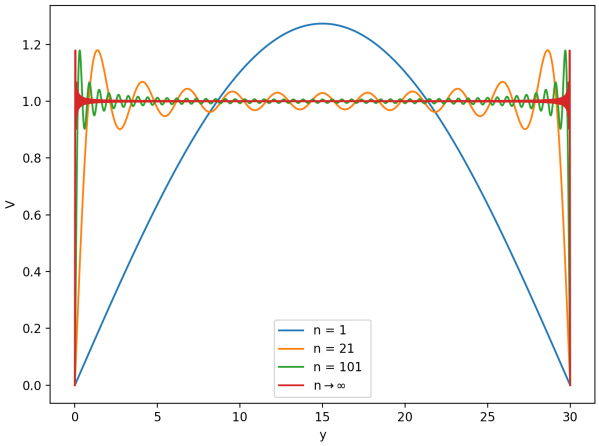

Figura 6: El potencial eléctrico cuando $x = 0$ y $n = 1,\ 21,\ 101$ y cuando $n \to \infty$.

Notemos que cuando $n \to \infty$ en la Figura 6, la función seno se hace más suave. Esto es característico del fenómeno de Gibbs observado como solución numérica convergente de la serie de Fourier con discontinuidades en los bordes. Este resultado se puede observar en la Figura 7 cuando n incrementa $n = 21,\ 101,\ n \to \infty$ y $x = 0$. Y conforme incrementa x la función dominante será la exponencial y el potencial eléctrico tendrá un decamiento exponencial. Tal como se muestra en la Figura 7.

Los resultados por el método de relajación para el comportamiento del potencial eléctrico en las placas pa-

ralelas se muestan en la Figura 4 y Figura 5. Este resultado es coherente, con los resultados obtenidos por la solución analítica, cuando $n \to \infty$ como se muestra en la Figura 7 y Figura 8. Un estudio similar se presenta en [5], que obtiene la misma coherencia con los resultados por estas dos formas de solución, que es de esperarse.

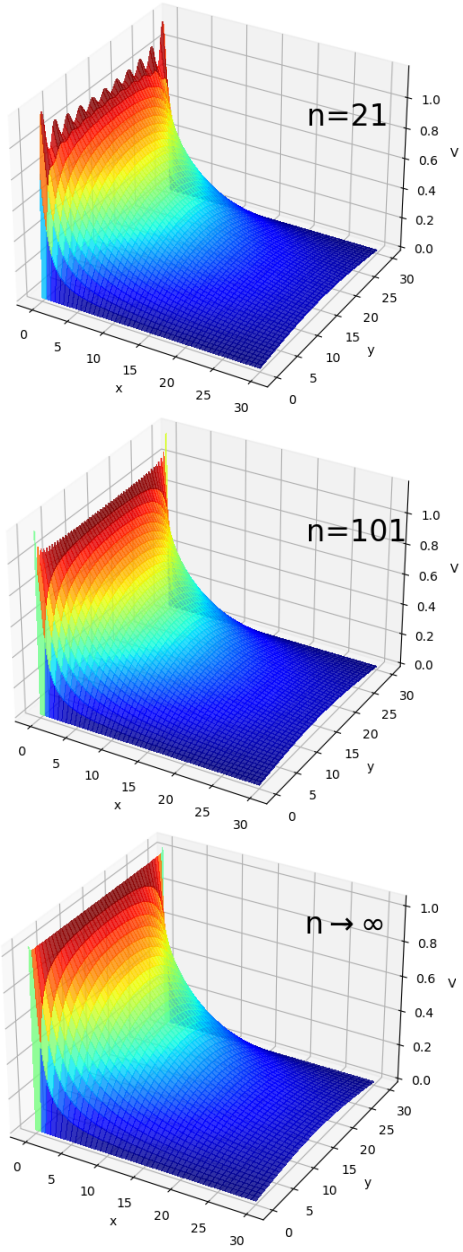

Figura 7: Comportamiento del potencial en las placas paralelas cuando $n = 21, 101$ y $n \to \infty$, solución analítica.

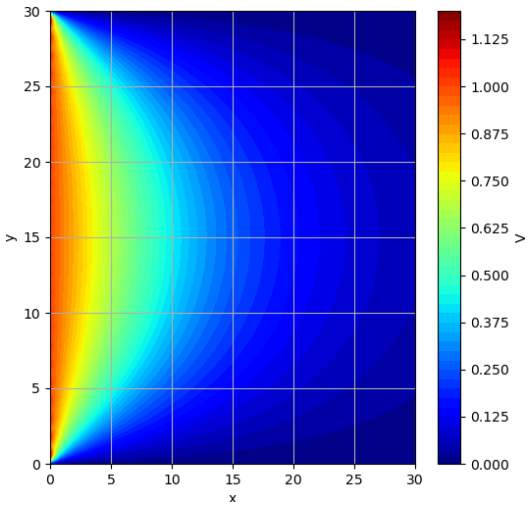

Figura 8: Comportamiento del potencial eléctrico en el contorno, cuando $n \to \infty$, solución analítica.

Conclusiones

En la presente investigación se demostró que el método de relajación entrega resultados satisfactorios en comparación con la solución analítica. Además, de forma general el método de relajación, implementada en el lenguaje Python, puede resolver problemas de gran complejidad sin incurrir en procesos de solución demasiado elaborados.

El método es sencillo de desarrollar e implementar computacionalmente, sobre todo en lenguajes de programación orientados a objetos (POO). El costo computacional es bajo debido a que la convergencia es relativamente rápida. Este tipo de desarrollos e implementaciones computacionales son el punto de partida para problemas más complejos, relacionados a casos electrodinámicos.

Una propuesta a futuro es de aplicar técnicas que permitan aproximar las soluciones a problemas con funciones o parámetros variables en el tiempo. Consiguiendo de esta manera reducir las incertidumbres propios de los modelos numéricos y así aproximar los resultados a la realidad del problema estudiado.

Agradecimientos

Los autores agradecen, a la Unidad de Posgrado de la Facultad de Ciencias Físicas de la UNMSM por la transmisión del conocimiento realizada en sus aulas, y en este tiempo de pandemia de manera remota pero manteniendo la misma calidad de siempre.

Apéndice (Anexos)

Script para la solución analítica y malla bidimensional, entrada (9):

[7]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[8]:
```
a = 1
x = np.arange(0,1.01,0.05)
y = np.arange(0,a+0.01,0.05)
X,Y = np.meshgrid(x,y)
```

[9]:
```
%matplotlib notebook
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(5,5))
ax = fig.gca(projection='3d')

colortuple = ('w', 'b')
colors = np.empty(X.shape, dtype=str)
for i in range(len(x)):
    for j in range(len(x)):
        colors[i, j] = colortuple[(i + j) %␣
↪len(colortuple)]
surf = ax.plot_surface(X, Y, np.zeros(np.
↪shape(X)), rstride=1, cstride=1,␣
↪facecolors=colors)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('V')
```

[10]:
```
n=1
a=1
V0 = 1
x = np.arange(0,1.01,0.01)
y = np.arange(0,a+0.01,0.01)
X,Y = np.meshgrid(x,y)
V=(4*V0/np.pi)*(1/n*np.exp(-n*np.pi*X/a)*np.
↪sin(n*np.pi*Y/a))

fig = plt.figure(figsize=(5.,5.))
ax = fig.gca()
cf = ax.contourf(X,Y,V,64,cmap='Blues')
ax.grid()
plt.colorbar(cf)
ax.set_xlabel('x')
ax.set_ylabel('y')
```

[11]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(5.,5.))
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X, Y, V, cmap='Blues',␣
↪alpha=0.8,linewidth=0, antialiased=False)
cset = ax.contour(X, Y, V, zdir='y',␣
↪offset=1, cmap='Blues',levels=1)
ann = ax.text(.7,.8,1, "n={}".format(n),␣
↪color='k',fontsize=24)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('V')
```

[12]:
```
from matplotlib import animation
from IPython.display import HTML
import matplotlib.animation as animation

def update_plot(frame_number, zarray, cf):
    cf[0].remove()
    cf[1].remove()
    cf[0] = ax.plot_surface(X, Y, zarray[:,:
↪,frame_number], cmap="Blues")
    cf[1] = ax.text(.7,.8,1, "n={}".
↪format(n_range[frame_number]),␣
↪color='k',fontsize=24)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

n_range = np.arange(1,28,2)
zarray = np.zeros((np.shape(X)[0], np.
↪shape(Y)[1], len(n_range)))

V_n = lambda X,Y,n : (4*V0/np.pi)*(1/n*np.
↪exp(-n*np.pi*X/a)*np.sin(n*np.pi*Y/a))

for i, n in enumerate(n_range):
    zarray[:,:,i] = V_n(X,Y,n)  + zarray[:,:
↪,i-1]

cf = []
cf.append(ax.plot_surface(X, Y, zarray[:,:
↪,0], color='0.75'))
cf.append(ax.text(.7,.8,1, "n={}".
↪format('1'), color='k',fontsize=24))

ax.set_zlim(0,1.5)
anim = animation.FuncAnimation(fig,␣
↪update_plot, len(n_range), fargs=(zarray,␣
↪cf),interval=400)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('V')

plt.close(anim._fig)

HTML(anim.to_html5_video())
```

Script para la solución numérica (Método de Relajación)

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

[2]:
```
maxIter = 500

lonX = lonY = 30
delta = 1
V0 = 1
Vi = 1

Vtop = 0
Vbottom = 0
Vleft = V0
Vright = 0

colourMap = plt.cm.jet

X, Y = np.meshgrid(np.arange(0, lonX), np.
 ↪arange(0, lonY))

V = np.empty((lonX, lonY))
V.fill(Vi)

V[:, :1] = Vleft
V[(lonY-1):, :] = Vtop
V[:1, :] = Vbottom
V[:, (lonX-1):] = Vright

for iteration in range(0, maxIter):
    for i in range(1, lonX-1, delta):
        for j in range(1, lonY-1, delta):
            V[i, j] = 0.25 * (V[i+1][j] +␣
 ↪V[i-1][j] + V[i][j+1] + V[i][j-1])

fig = plt.figure(figsize=(6.,6.))
ax = fig.gca()
cf = ax.contourf(X, Y, V, 50, cmap=colourMap)
ax.grid()

plt.colorbar(cf,label="V")

ax.set_xlabel('x')
ax.set_ylabel('y')
```

[3]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(8.,8.))
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X, Y, V,␣
 ↪cmap=colourMap, alpha=0.8, linewidth=0,␣
 ↪antialiased=False)
```

```
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('V')
fig.savefig('3d-rm.png')
```

Referencias

[1] S. Nakamura *Applied Numerical Methods with Software*. Editorial Prentice Hall, Upper Saddle River, NJ, United States (1991).

[2] R.P. Feynman and R.B. Leighto and M.L. Sands. *The Feynman Lectures on Physics*, V. 2. Addicon-Wesley Publishing Company, Reading, Massachusetts, U.S.A (1998).

[3] G. Segundo. *Solución numérica de la ecuación de Laplace*. Anales del Instituto de Profesores "Artigas", ANEP CFE Instituto de Profesores "Artigas" (2010).

[4] F. Mulligan. *An illustration of method of finite differences in the solution of Laplace's equation*. European journal of physics, **13**(2) 57 (1992).

[5] A. Mitra. *Finite difference method for the solution of Laplace equation*. Department of aerospace engineering Iowa state University (2010).

[6] E. Kreyszig. *Matemáticas Avanzadas para Ingeniería*, Vol. II, Métodos Numéricos para Ecuaciones Diferenciales Parciales, 551,427 (2000)

[7] S. Nakamura *Applied Numerical Methods with Software*. Editorial Prentice Hall, Upper Saddle River, NJ, United States (1991).

[8] V. D. Correa, A. Molina y A. Mejía. *Método iterativo para el cálculo de potenciales electrostáticos*. Scientia et technica, **32**(3) 1018-1040 (2006).

[9] D. Robertson. *Relaxation Methods for Partial Differential Equations: Applications to Electrostatics*. Lecture Module, Otterbein University, Westerville (2010).