EE320 - Electromagnetic Theory

Assignment on Computational Methods

W.M.B.S.K.Wijenayake (E/19/445)

25/11/2023

a. Analytical Solution

Using Poisson's Equation,

$$\nabla^2\Phi = -\frac{\rho}{\epsilon}$$

Using given data,

$$\begin{aligned}
\frac{\partial^2\Phi}{\partial z^2} &= -\frac{\rho_0 z}{a\epsilon} \\
\frac{\partial\Phi}{\partial z} &= -\frac{\rho_0 z^2}{2a\epsilon} + C \\
\Phi &= -\frac{\rho_0 z^3}{6a\epsilon} + Cz + D \qquad (1)
\end{aligned}$$

By substituting given conditions at z=0,

$$D = 0$$

at z=10,

$$\begin{aligned}
100 &= -\frac{10^{-10} \times 10^3}{6 \times 5 \times \frac{4}{36\pi} \times 10^{-9}} + 10C \\
10C &= 100 + 30\pi \\
C &= 10 + 3\pi
\end{aligned}$$

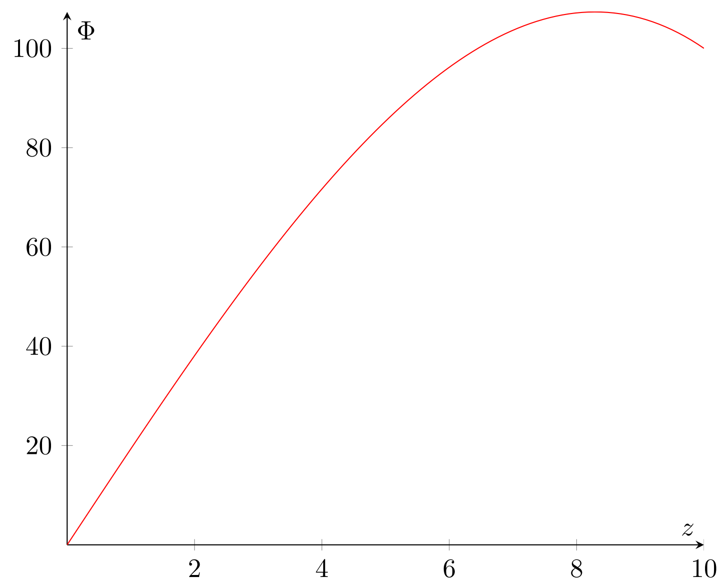
Thus this gives,

$$\begin{aligned}
\Phi &= -\frac{10^{-10} z^3}{30\epsilon_r\epsilon} + (10 + 3\pi)z \\
&= -\frac{3\pi z^3}{100} + (10 + 3\pi)z
\end{aligned}$$

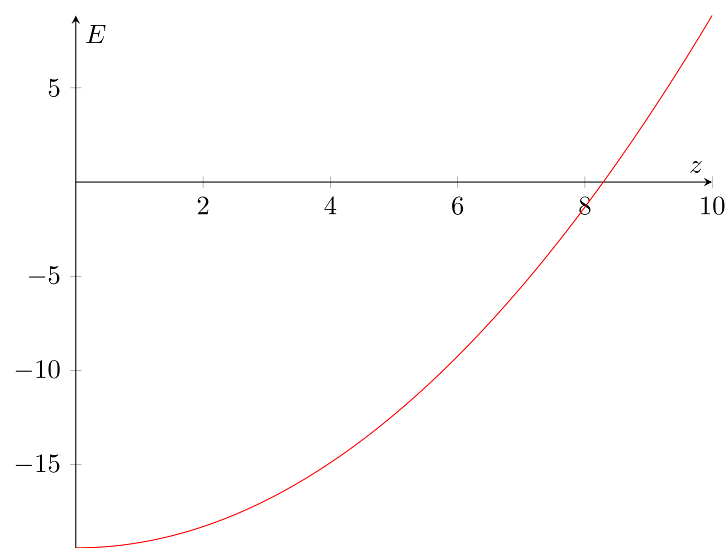
Thus Electric field E,

$$\begin{aligned}
E &= -\nabla\Phi \\
&= \frac{9\pi z^2}{100} - 10 - 3\pi
\end{aligned}$$

Variation of the Potential



Variation of the Electric Field



b. Finite Difference Method

Consider a 1D grid of height H, divided into nodes each of height h,

h
h
. . .
A
B

Let the potential at upper boundary of node A be Φ_1, potential at upper boundary of node B be Φ_0 and the potential of lower boundary of node B be Φ_2.

Considering Boundaries of A and B ,

$$\begin{aligned}\tfrac{\partial \Phi}{\partial x}\Big|_A &= \frac{\Phi_1 - \Phi_0}{h}\\ \tfrac{\partial \Phi}{\partial x}\Big|_B &= \frac{\Phi_0 - \Phi_2}{h}\end{aligned}$$

Thus at the lower boundary of A,

$$\begin{aligned}\tfrac{\partial^2 \Phi}{\partial x^2}\Big|_0 &= \frac{\tfrac{\partial \Phi}{\partial x}\Big|_A - \tfrac{\partial \Phi}{\partial x}\Big|_B}{h}\\ &= \frac{\Phi_1 + \Phi_2 - 2\Phi_0}{h^2} \qquad (1)\end{aligned}$$

Using Poissen's Equation,

$$\begin{aligned}\boldsymbol{\nabla}^2 \Phi &= -\frac{\rho}{\epsilon}\\ \frac{\partial^2 \Phi}{\partial x^2} &= -\frac{\rho}{\epsilon}\\ -\frac{\rho}{\epsilon} &= \frac{\Phi_1 + \Phi_2 - 2\Phi_0}{h^2}\end{aligned}$$

This gives the potential of the lower boundary of A,

$$\begin{aligned}\Phi_0 &= \frac{\Phi_1 + \Phi_2}{2} + \frac{h^2 \rho}{2\epsilon}\\ &= \frac{\Phi_1 + \Phi_2}{2} + \frac{h^2 \rho_0 z}{2a\epsilon}\end{aligned}$$

Approach 1

Rather than selecting 0 for all unknown potentials, I tried with a different approach that worked on this specific example.

I calculated the potential at the mid point using the known potentials at $z = 0$ and $z = 10$
Here the 1D grid is only divided to two parts hence, $h = 5$

$$\begin{aligned}\Phi_5 &= \frac{0+100}{2} + \frac{5^2\rho_0 \times 5}{2a\epsilon} \\ &= 85.3429V\end{aligned}$$

Now, using the potential at $z = 5$ and $z = 0$, we can calculate the potential at $z = 2.5$ by selecting h as 2.5.
Similarly, using potentials at $z = 5$ and $z = 10$, we can can calculate the potential at $z = 7.5$ by selecting h as 2.5

$$\begin{aligned}\Phi_{2.5} &= \frac{\Phi_0 + \Phi_5}{2} + \frac{2.5^2\rho_0 \times 2.5}{2a\epsilon} \\ \Phi_{7.5} &= \frac{\Phi_5 + \Phi_{10}}{2} + \frac{2.5^2\rho_0 \times 7.5}{2a\epsilon}\end{aligned}$$

And I continued this process until $h \leq 0.01$, which will calculate the potential of n(≥ 10000) points in the range of $z \in [0.10]$
Even though this worked totally fine for this example, when we apply this to other scenarios there might be an error. Thus I have included the iterative method as well.

c. Code

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

RESULTS = dict()

EPSILON = 4 * 10 ** (-9) / (36 * np.pi)
P_0 = 10 ** (-10)

ITERATIONS = 10000


def evaluate(v1, v2, z, h):
    return (v1 + v2) / 2 + (z * P_0 * h**2) / (10 * EPSILON)


def analytical(z):
    return -3 * np.pi * z**3 / 100 + (10 + 3 * np.pi) * z


class bisect_method:
    def __init__(self) -> None:
        self.initial_conditions()

```

```
    def initial_conditions(self):
        RESULTS[0] = 0
        RESULTS[10] = 100

    def initial_guess(self, L, U):
        z = (L + U) / 2
        h = (U - L) / 2
        if h < 0.01:
            return
        v1, v2 = RESULTS[L], RESULTS[U]
        RESULTS[z] = evaluate(v1, v2, z, h)
        self.initial_guess(z, U)
        self.initial_guess(L, z)
        return


class iterative_method:
    def __init__(self) -> None:
        self.without_guess()

    def without_guess(self):
        RESULTS[0] = 0
        i = 1
        while i < 100:
            RESULTS[i] = 0.1
            i += 1
        RESULTS[100] = 100

    def iteration_without_guess(self):
        i = 99
        temp = RESULTS
        while i > 0:
            temp[i] = evaluate(RESULTS[i - 1], RESULTS[i + 1], i /
    10, 0.1)
            i -= 1

        for key in RESULTS:
            RESULTS[key] = temp[key]


class Demonstration:
    def __init__(self):
        self.fig = plt.figure()
        self.temp = self.fig.add_subplot(1, 1, 1)

    def animate(self, i):
        if i > len(RESULTS):
            plt.close()
            return
        X = list(RESULTS.keys())[:i]
        Y = list(RESULTS.values())[:i]

        X2 = list(sorted(RESULTS.keys()))
        Y2 = [analytical(x) for x in X2]

        self.temp.clear()
        self.temp.plot(X2, Y2, c="black")
```

```
        plt.title("Bisect Method Animation")
        self.temp.scatter(X, Y, linewidths=0.001)
        self.temp.scatter(X[-2:], Y[-2:], c="r", linewidths=0.001)

    def animate_2(self, i):
        if i > len(RESULTS):
            plt.close()
            return
        X = list(RESULTS.keys())[:i]
        Y = list(RESULTS.values())[:i]

        X2 = list(sorted(RESULTS.keys()))
        Y2 = [analytical(x / 10) for x in X2]

        self.temp.clear()
        plt.title("Iterative Method Animation (Skipped)")
        self.temp.plot(X2, Y2, c="black")
        self.temp.plot(X, Y)

    def show_animation(self, flag):
        fn = self.animate
        interval = 30
        if flag:
            fn = self.animate_2
            interval = 5
        self.animation_ = animation.FuncAnimation(
            self.fig, fn, interval=interval, repeat=False,
            cache_frame_data=False  # type: ignore
        )
        plt.show()


def main_bisect():
    method = bisect_method()
    method.initial_guess(0, 10)
    Demo = Demonstration()
    Demo.show_animation(flag=False)
    RESULTS.clear()


def main_iterative_method():
    method = iterative_method()
    for iter in range(ITERATIONS):
        method.iteration_without_guess()
        if iter % 800 == 0 and iter // 800 < 8:
            Demo = Demonstration()
            Demo.show_animation(flag=True)


if __name__ == "__main__":
    main_bisect()
    main_iterative_method()
```

d. Results

It is very clear to mention that using both of the above implemented methods the potential curve is converging over the iterations.

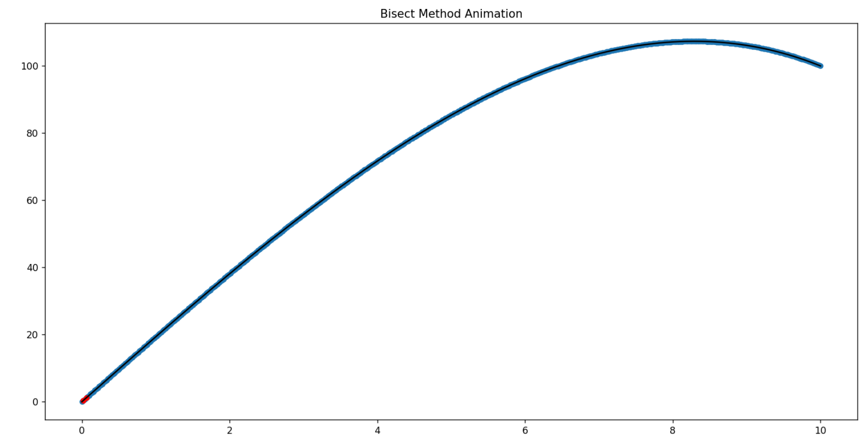


Figure 1: Bisect Method Results

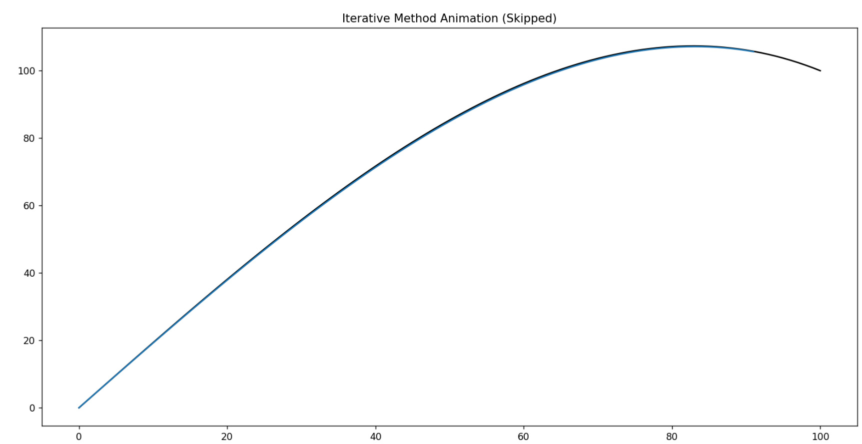


Figure 2: Iterative Method Results