Modelowanie Komputerowe
Electrostatic Potentials

Gabriela Białoskórska, Mikołaj Knysak, Ignacy Tekieli
Fizyka Techniczna

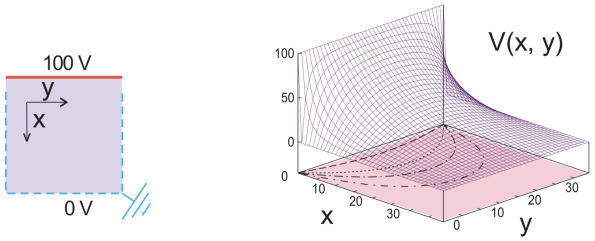
Grudzień 2020

Spis treści

1 Wprowadzenie

1.1 Cel Pracy

Celem niniejszego projektu było odnalezienie potencjału elektrostatycznego dla wszystkich pnktów, znajdujących się wewnątrz nienaładowanego kwadratu, przedstawionego na poniższej ilustracji.



Rysunek 1: *Po lewej: zacieniony obszar przestrzeni w kwadracie, w którym określamy potencjał elektryczny rozwiązując równanie Laplace'a. Na górze znajduje się przewód o stałym napięciu 100V i przewód uziemiający (przerywany) po bokach i na dole. Po prawej: obliczony potencjał elektryczny jako funkcja x i y. Rzuty na zacienioną płaszczyznę xy są liniami ekwipotencjalnymi (konturowymi).*

Dno i boki tego regionu składają się z przewodów, które są „uziemione". Górny przewód jest podłączony do akumulatora, utrzymującego go na stałym poziomie 100V.

1.2 Wstęp teoretyczny

1.2.1 Potencjał elektryczny

Energia potencjalna ładunku w polu elektrycznym zależy od wielkości tego ładunku, wobec tego do opisu pola elektrycznego lepiej posługiwać się energią potencjalną przypadającą na jednostkowy ładunek czyli potencjałem elektrycznym. **Potencjał elektryczny definiujemy jako energię potencjalną pola elektrycznego podzieloną przez jednostkowy ładunek.**

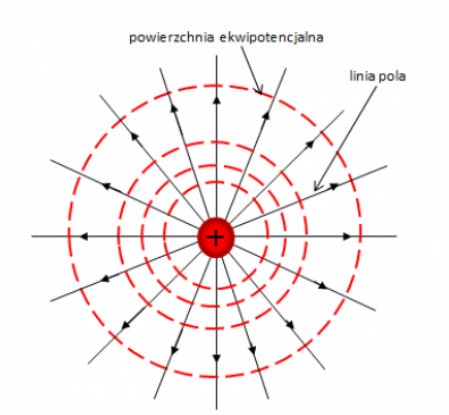
$$V(r) = \frac{E_p(r)}{q} = \frac{W_{\infty r}}{q}$$

W układzie SI jednostką potencjału elektrycznego jest wolt (V); 1V = 1J/C. **Potencjał pola ładunku punktowego Q wynosi:**

$$V(r) = k\frac{Q}{r}$$

Obliczony potencjał określa pracę potrzebną do przeniesienia jednostkowego ładunku z nieskończoności na odległość r od ładunku Q. Potencjał charakteryzuje pole elektryczne, jednak nie zależy od umieszczonego w nim ładunku.

Jednym ze sposobów graficznego przedstawienia pola elektrycznego jest wyrysowanie linii pola. Są to linie w każdym punkcie styczne do kierunku pola. Po nich poruszałby się nie zakłócający pola dodatni ładunek próbny.



Rysunek 2: Przykład: *Powierzchnie ekwipotencjalne dla pola centralnego*

Należy podkreślić, że liczba linii natężenia pola elektrycznego przypadających na jednostkę powierzchni informuje nas o wielkości natężenia pola elektrycznego.

1.2.2 Równanie Laplace'a

Traktujemy cały kwadrat na Rysunku 1. jako granicę z określonymi napięciami. Jeśli wyobrazimy sobie nieskończenie małe izolatory umieszczone w górnych rogach "pudełka", wówczas uzyskamy zamkniętą granicę, w ramach której rozwiążemy nasz problem. Ponieważ wartości potencjału są podane ze wszystkich stron, na granicy mamy warunki Neumanna i, zgodnie z poniższą tabelą, unikalne i stabilne rozwiązanie:

Boundary Condition	*Elliptic* (*Poisson Equation*)	*Hyperbolic* (*Wave Equation*)	*Parabolic* (*Heat Equation*)
Dirichlet open surface	Underspecified	Underspecified	*Unique & stable (1-D)*
Dirichlet closed surface	*Unique & stable*	Overspecified	Overspecified
Neumann open surface	Underspecified	Underspecified	*Unique & Stable (1-D)*
Neumann closed surface	*Unique & stable*	Overspecified	Overspecified
Cauchy open surface	Nonphysical	*Unique & stable*	Overspecified
Cauchy closed surface	Overspecified	Overspecified	Overspecified

Rysunek 3: *Relacja między warunkami brzegowymi a niepowtarzalnością dla równań różniczkowych cząstkowych (RRC)*

Z klasycznej elektrodynamiki wiadomo, że potencjał elektryczny *U(x)* powstający z ładunków statycznych spełnia RRC Poissona:

$$\nabla^2 U(x) = -4\pi\rho(x)$$

gdzie $\rho(x)$ jest gęstością ładunku. W wolnych od ładunków obszarach przestrzeni, to znaczy w regionach, w których $\rho(x) = 0$, potencjał spełnia równanie Laplace'a:

$$\nabla^2 U(x) = 0$$

Oba te równania są eliptycznymi RRC o postaci mającej wiele zastosowań. Rozwiązujemy je we współrzędnych prostokątnych 2D:

$$\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = \begin{cases} 0, & \text{Laplace's equation,} \\ -4\pi\rho(\mathbf{x}), & \text{Poisson's equation.} \end{cases}$$

W obu przypadkach widzimy, że potencjał zależy jednocześnie od x i y. W przypadku równania Laplace'a ładunki, które są źródłem pola, wchodzą pośrednio, określając potencjalne wartości w pewnym obszarze przestrzeni; w przypadku równania Poissona wchodzą bezpośrednio.

2 Materiały i metody

2.1 Wiadomości wstępne

2.2 Lattice PDE Implementation

Najbardziej ogólna forma RRC (*ang. PDE*) dla dwóch niezależnych zmiennych wygląda następująco:

$$A\frac{\partial^2 U}{\partial x^2} + 2B\frac{\partial^2 U}{\partial x \partial y} + C\frac{\partial^2 U}{\partial y^2} + D\frac{\partial U}{\partial x} + E\frac{\partial U}{\partial y} = F$$

gdzie A, B, C i F są funkcjami zmiennych x i y.
Rozwiązuje ona problem kwadratu z Rysunku 1. (lewa strona).

Napiszmy teraz kod ustalając, że długość pudełka $L = N_{max}\triangle = 100$, dla $\triangle = 1$:

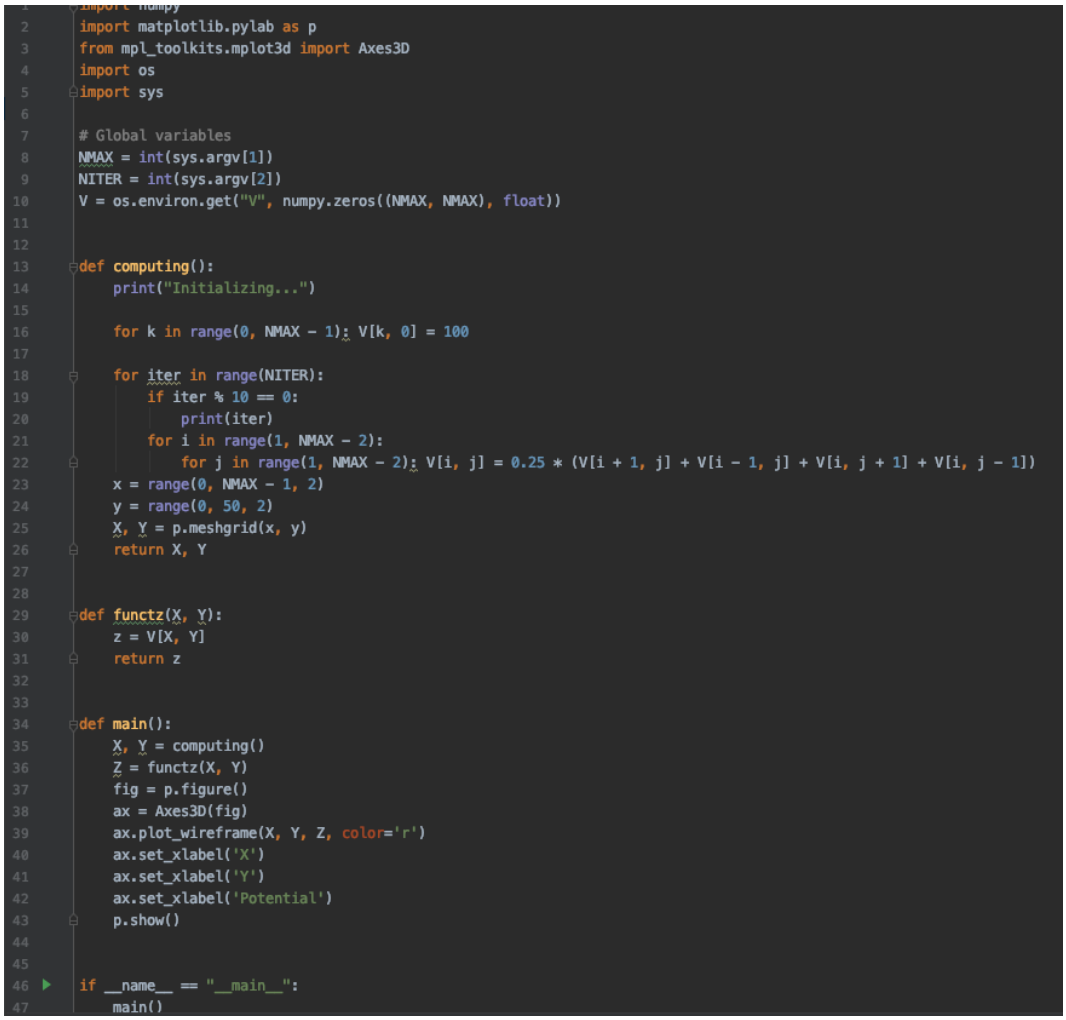
$$\begin{aligned} U(i, N_{\max}) &= 99 \quad \text{(top)}, & U(1, j) &= 0 \quad \text{(left)}, \\ U(N_{\max}, j) &= \ \ 0 \quad \text{(right)}, & U(i, 1) &= 0 \quad \text{(bottom)}, \end{aligned}$$

```
# LaplaceLine.py:  Solve Laplace's eqtn, 3D matplot, close shell to quit

from numpy import *
import matplotlib.pylab as p;
from mpl_toolkits.mplot3d import Axes3D

print("Initializing")
Nmax = 100; Niter = 70; V = zeros((Nmax, Nmax), float)            # float maybe Float

print( "Working hard, wait for the figure while I count to 60")
for k in range(0, Nmax-1):  V[k,0] = 100.0                              # line at 100V

for iter in range(Niter):                                     # iterations over algorithm
    if iter%10 == 0: print( iter)
    for i in range(1, Nmax-2):
        for j in range(1,Nmax-2): V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])
x = range(0, Nmax-1, 2);  y = range(0, 50, 2)                 # plot every other point
X, Y = p.meshgrid(x,y)

def functz(V):                                                 # Function returns V(x, y)
    z = V[X,Y]
    return z

Z = functz(V)
fig = p.figure()                                                          # Create figure
ax = Axes3D(fig)                                                           # plot axes
ax.plot_wireframe(X, Y, Z, color = 'r')                                   # red wireframe
ax.set_xlabel('X')                                                           # label axes
ax.set_ylabel('Y')
ax.set_zlabel('Potential')
p.show()                                              # display fig, close shell to quit
```

Rysunek 4: *Powyższy screen przedstawia sugerowane rozwiązanie, które pochodzi z pdf-a załączonego do zajęć.*

2.3 Tworzenie projektu

Nasz projekt zawiera w sobie pewne istotne różnice, w porównaniu do przedstawionego powyżej przykładu, które omówimy bazując na poszczególnych fragmentach i wierszach. Najpierw jednak załączamy poglądowo screena całości kodu:



```
import numpy
import matplotlib.pylab as p
from mpl_toolkits.mplot3d import Axes3D
import os
import sys

# Global variables
NMAX = int(sys.argv[1])
NITER = int(sys.argv[2])
V = os.environ.get("V", numpy.zeros((NMAX, NMAX), float))


def computing():
    print("Initializing...")

    for k in range(0, NMAX - 1): V[k, 0] = 100

    for iter in range(NITER):
        if iter % 10 == 0:
            print(iter)
        for i in range(1, NMAX - 2):
            for j in range(1, NMAX - 2): V[i, j] = 0.25 * (V[i + 1, j] + V[i - 1, j] + V[i, j + 1] + V[i, j - 1])
    x = range(0, NMAX - 1, 2)
    y = range(0, 50, 2)
    X, Y = p.meshgrid(x, y)
    return X, Y


def functz(X, Y):
    z = V[X, Y]
    return z


def main():
    X, Y = computing()
    Z = functz(X, Y)
    fig = p.figure()
    ax = Axes3D(fig)
    ax.plot_wireframe(X, Y, Z, color='r')
    ax.set_xlabel('X')
    ax.set_xlabel('Y')
    ax.set_xlabel('Potential')
    p.show()


if __name__ == "__main__":
    main()
```

Rysunek 5: Całość kodu

Na początku kodu importujemy konieczne biblioteki. Bibliotekę `matplotlib.pylab` oznaczamy jako *p*, natomiast z biblioteki `mpl-toolkits.mplot3d` importujemy wyłącznie funkcję *Axes3D*.

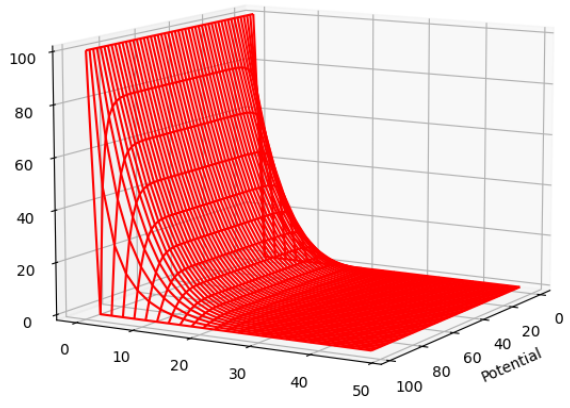
Następnie definiujemy zmienne globalne (zgodnie z komentarzem, który zamieściliśmy powyżej): NMAX, NITER (lioczba iteracji pętli funkcji computing) oraz V. Zaznaczamy także, że w momencie wykonywania obliczeń wyświetla się słowo "Initializing".

Funckja `computing` wylicza zmienne x oraz y z równania Laplace'a, opisanego we wstępie. Następnie funckja `functz` oblicza wartość zmiennej z na podstawie wyliczonych wcześniej x i y (tak, aby wykres wyświetlał się w trzech wymiarach). Funkcja `main` rysuje wykres na podstawie pozostałych, opisanych funkcji.

Warto zaznaczyć, że nasz kod umożliwia użytkownikowi samodzielne ustalenie danych wejściowych (tj. NMAX oraz NITER). Opcja ta jest dostępna dzięki dwóm dodatkowym bibliotekom - `sys` i `os`. W kolejnej sekcji przedstawiamy w jaki sposób przebiega rysowanie funkcji.

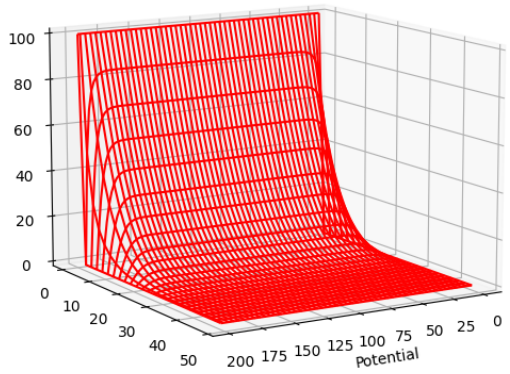
3 Wyniki

Rozpocznijmy od wizualizacji potencjału dla danych domyślnych. Wykres wygląda następująco:



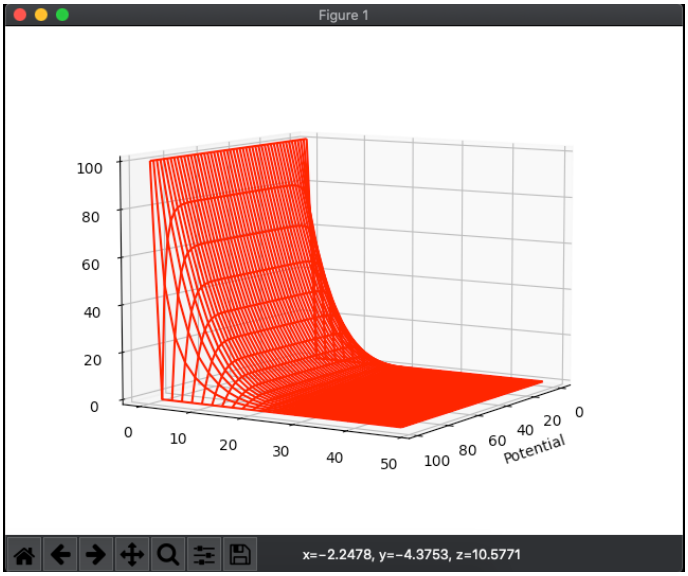
Rysunek 6: Wizualizacja dla danych domyślnych

Dla danych ustalonych przez użytkownika wpisujemy w terminalu `python3 custom-potential.py 200 100`, gdzie 200 i 100 to przykładowe, ustalone przez nas dane:

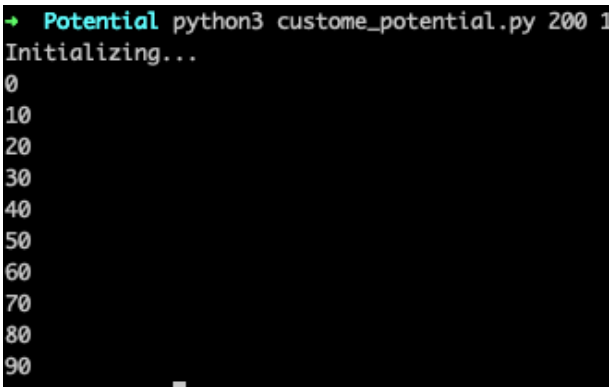


Rysunek 7: Wizualizacja dla ustalonych przez użytkownika danych

Całościowo output wygląda następująco:



Wpisywanie komendy w terminalu:



4 Podsumowanie

Zadanie zostało wykonane prawidłowo, przy jednoczesnym wprowadzeniu drobnych zmian. W naszym kodzie dajemy użytkownikowi możliwość wyboru danych wejściowych, jednak kod wciąż działa prawidłowo i na tej samej zasadzie co załączony przykład. Pozostałe zmiany obejmują technikę programowania. W naszym kodzie wywołujemy tylko konieczne funkcje z danej biblioteki, nie stosujemy *. Ponadto wprowadziliśmy dla wygody pracy zmienne globalne.

4.1 Optymalizacja

Dodatkową zmianą, wprowadzaną już po powstaniu powyższego raportu, była zwiększona optymalizacja kodu. Załączamy zmodyfikowany kod poniżej, dla osób zainteresowanych.

```
import argparse
import numpy
import matplotlib.pylab as plt
from mpl_toolkits.mplot3d import Axes3D
import os


def func_z(X, Y):
    return V[X, Y]


def compute():
    print("Initializing...")
```

```
    for k in range(0, N_MAX - 1):
        V[k, 0] = 100

    for i in range(N_I):
        if i % 10 == 0:
            print(i)
        for y in range(1, N_MAX - 2):
            for x in range(1, N_MAX - 2):
                V[y, x] = 0.25 * (V[y + 1, x] + V[y - 1, x]
                + V[y, x + 1] + V[y, x - 1])
    x = range(0, N_MAX - 1, 2)
    y = range(0, 50, 2)
    X, Y = plt.meshgrid(x, y)
    return X, Y


def main():
    X, Y = compute()
    Z = func_z(X, Y)

    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_wireframe(X, Y, Z, color='r')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Potential')
    plt.show()


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('--n_max', type=int, required=False, default=100)
    parser.add_argument('--n_i', type=int, required=False, default=70)
    args = parser.parse_args()

    N_MAX, N_I = args.n_max, args.n_i
    V = os.environ.get("V", numpy.zeros((N_MAX, N_MAX), float))

    main()
```

Jest on jeszcze bardziej przyjazny użytkownikowi dzięki zastosowaniu biblioteki `argparse`. Działa szybciej, a zmienne które użytkownik wprowadza samodzielnie mogą być wpisane w dowolnej kolejności. W poprzedniej wersji musiała zostać ona zachowana (najpierw NMAX, następnie NITER). Jeśli dowolnie wybrane zmienne nie zostaną podane - program skompiluje się dla zmiennych domyślnych, podanych w zadaniu.

5 Bibliografia

Kod oraz rozdział 1.2.2 powstały na podstawie prezentowanego na zajęciach pliku pdf.