Computational Physics (PHYS6350)

Lecture 15: Partial Differential Equations

- Boundary value problems

Reference: Chapter 9 of *Computational Physics* by Mark Newman

March 18, 2025

Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu)

Course materials: https://github.com/vlvovch/PHYS6350-ComputationalPhysics/tree/spring2025

Partial differential equations

Describe functions of more than one variable

In physics, this commonly corresponds to fields $\phi(x, y, z)$

Examples:

- Electrostatic potential $\phi(x, y, z)$ (Poisson's equation)

$$\Delta\phi(x, y, z) = -\frac{\rho(x, y, z)}{\epsilon_0}$$

- Density or temperature profiles (diffusion/heat equation)

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} = D\Delta u(\mathbf{x}, t)$$

- Displacement (amplitude) profile (wave equation)

$$\frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} = c^2\Delta u(\mathbf{x}, t)$$

- Fluid dynamical fields (flow velocity) -- e.g. Navier-Stokes equations

$$\frac{\partial}{\partial t}(\rho\,\mathbf{u}) + \nabla\cdot(\rho\,\mathbf{u}\otimes\mathbf{u}) = -\nabla p + \nabla\cdot\boldsymbol{\tau} + \rho\,\mathbf{g}$$

Common methods for PDEs

- **Finite difference method**
  - Approximate the derivatives by finite differences
  - Easier to implement than other methods
  - Works best for regular (rectangular) shapes

- Finite element method
  - Subdivide the system into smaller parts -- finite elements
  - Boundary value problems in 2/3 dimensions
  - Works well for irregular shapes

- Finite volume method
  - Convert surface integrals around each mesh point into volume integrals
  - Conserves the mass by design, good for solving fluid dynamical equations

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = \mathbf{0}.$$

Boundary value problem

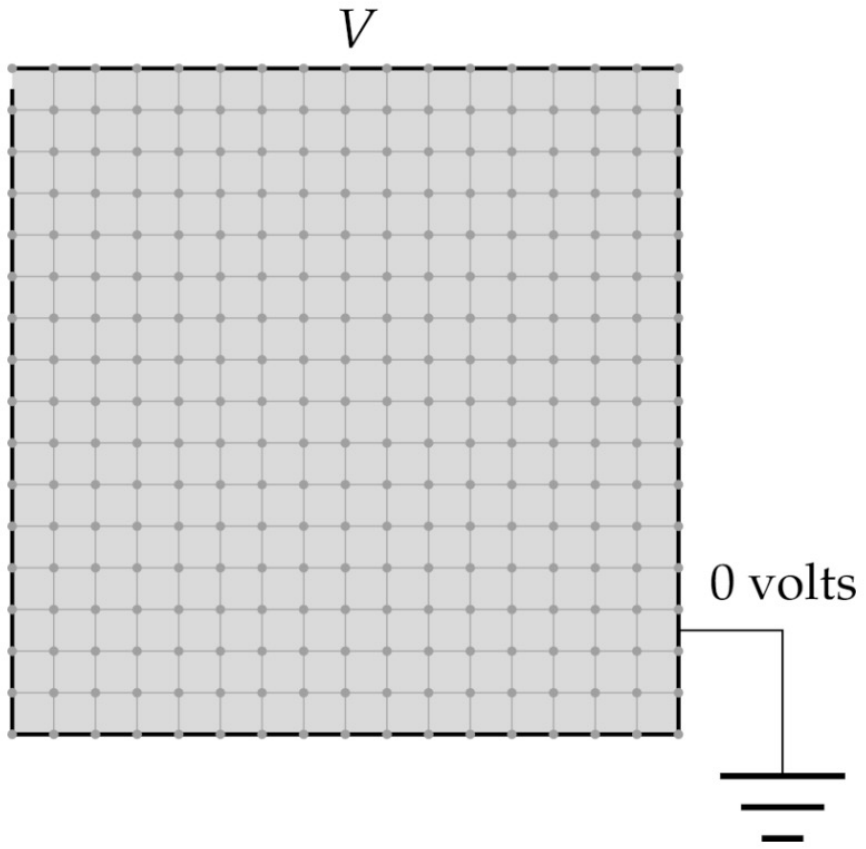
Consider Laplace's equation in two dimensions

$$\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} = 0.$$

Boundary conditions

$$\begin{aligned} \phi(x, L) &= V, \\ \phi(x, 0) &= 0, \\ \phi(0, y) &= 0, \\ \phi(L, y) &= 0. \end{aligned}$$

How to find the profile $\phi(x,y)$?

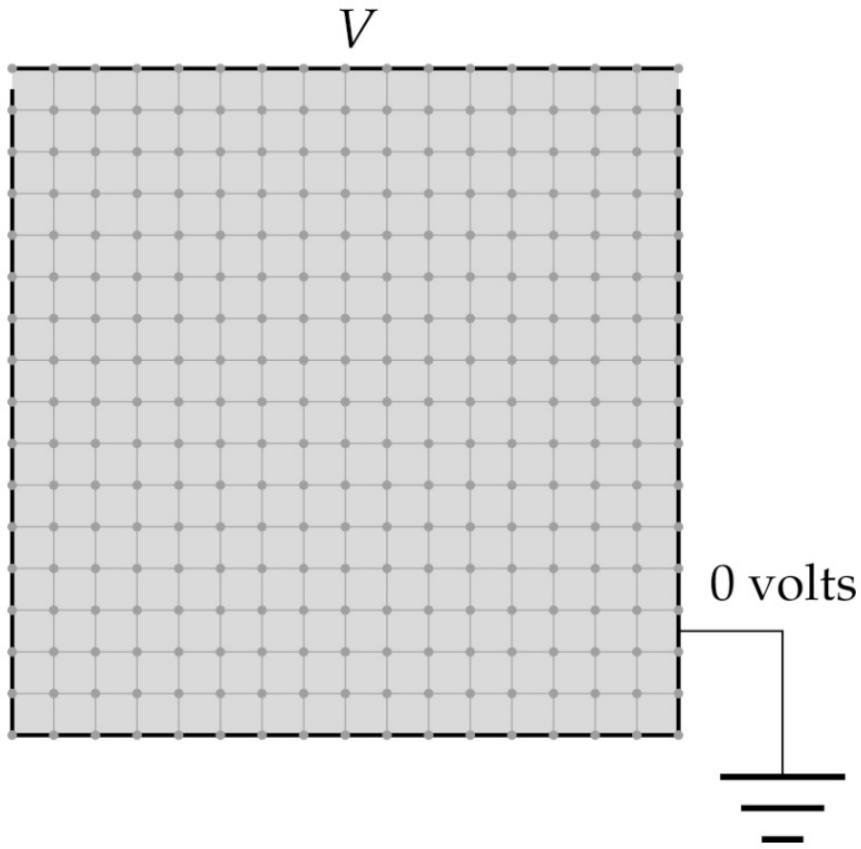


Source: M. Newman, Computational Physics

Boundary value problem: Finite difference method

$$\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} = 0.$$

$$\begin{aligned}
\phi(x, L) &= V, \\
\phi(x, 0) &= 0, \\
\phi(0, y) &= 0, \\
\phi(L, y) &= 0.
\end{aligned}$$



Source: M. Newman, Computational Physics

Discretize the space onto a grid $M \times M$ of length $a = L/M$

Apply central difference to the derivatives

$$\frac{\partial^2 \phi(x,y)}{\partial x^2} = \frac{\phi(x+a,y) - 2\phi(x,y) + \phi(x-a,y)}{a^2},$$

$$\frac{\partial^2 \phi(x,y)}{\partial y^2} = \frac{\phi(x,y+a) - 2\phi(x,y) + \phi(x,y-a)}{a^2}.$$

Laplace's equation becomes

$$\phi(x+a,y) + \phi(x-a,y) + \phi(x,y+a) + \phi(x,y-a) - 4\phi(x,y) = 0.$$

Boundary value problem: Jacobi method

$$\phi(x+a,y)+\phi(x-a,y)+\phi(x,y+a)+\phi(x,y-a)-4\phi(x,y)=0.$$

is a system of M^2 linear equations

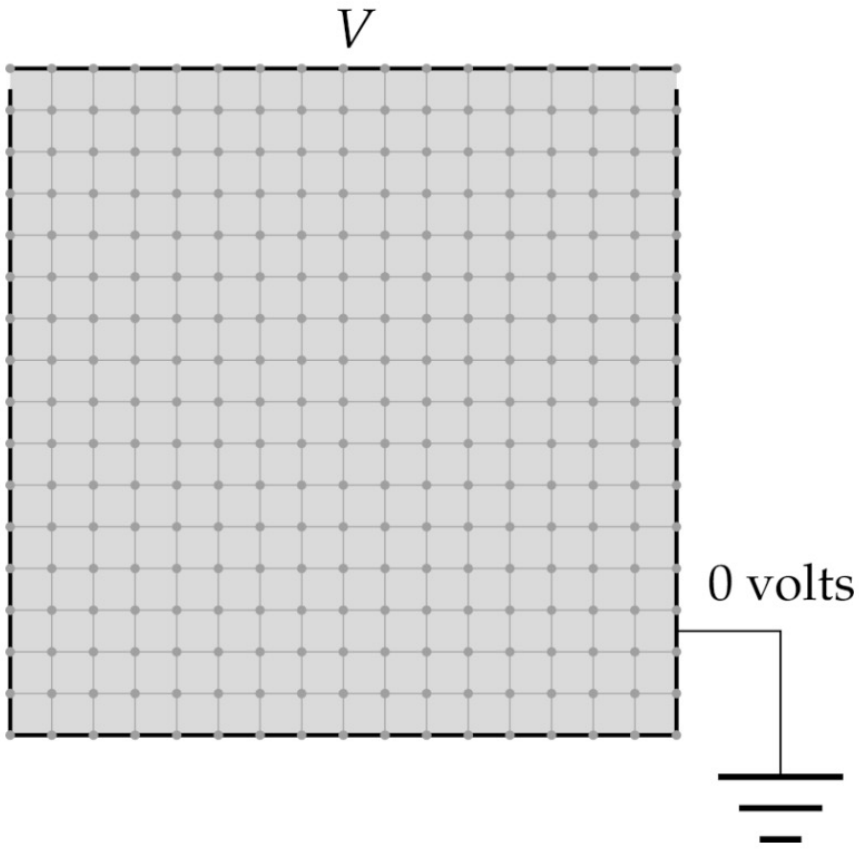
The general solution is $O(M^6)$

Jacobi method:

Perform the solution iteratively

$$\phi_{n+1}(x,y)=\frac{\phi_n(x+a,y)+\phi_n(x-a,y)+\phi_n(x,y+a)+\phi_n(x,y-a)}{4}$$

until the desired accuracy is met



Source: M. Newman, Computational Physics

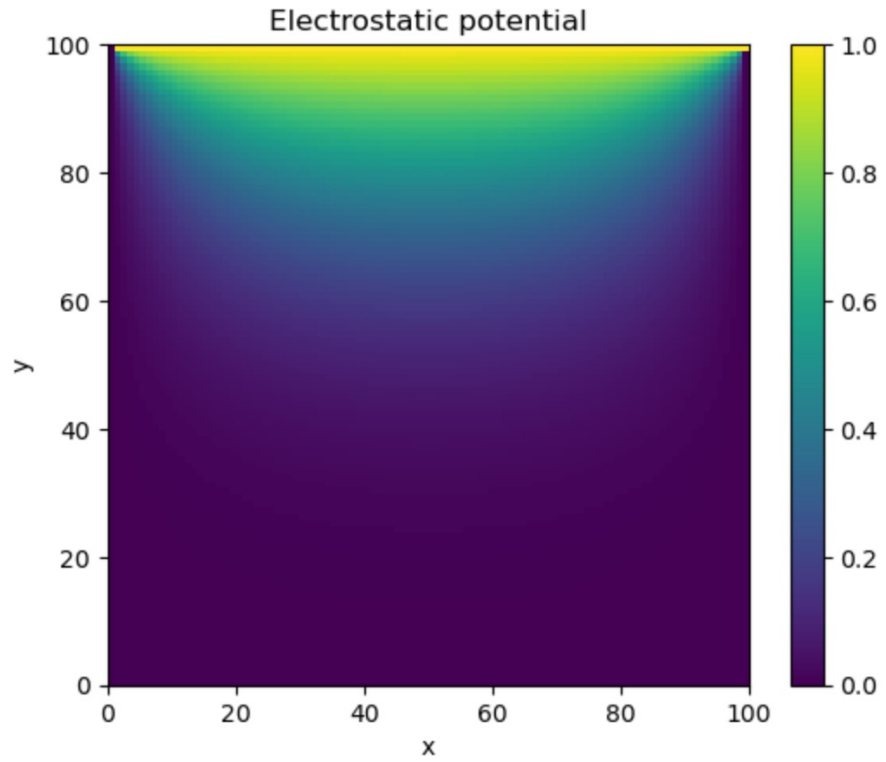
Boundary value problem: Jacobi method

```python
import numpy as np

# Single iteration of the Jacobi method
# The new field is written into phinew
def iteration_jacobi(phinew, phi):
    M = len(phi) - 1

    # Boundary conditions
    phinew[0,:] = phi[0,:]
    phinew[M,:] = phi[M,:]
    phinew[:,0] = phi[:,0]
    phinew[:,M] = phi[:,M]

    for i in range(1,M):
        for j in range(1,M):
            phinew[i,j] = (phi[i+1,j] + phi[i-1,j] + phi[i,j+1] + phi[i,j-1])/4

    delta = np.max(abs(phi-phinew))

    return delta

def jacobi_solve(phi0, target_accuracy = 1e-6, max_iterations = 100):
    delta = target_accuracy + 1.
    phi = phi0.copy()
    for i in range(max_iterations):
        delta = iteration_jacobi(phi, phi0)
        phi0, phi = phi, phi0

        if (delta <= target_accuracy):
            print("Jacobi method converged in " + str(i+1) + " iterations")
            return phi0

    print("Jacobi method failed to converge to a required precision in " + str(max_iterations) + " iterations")
    print("The error estimate is ", delta)

    return phi
```

Boundary value problem: Jacobi method

```
# Constants
M = 100          # Grid squares on a side
V = 1.0          # Voltage at top wall
target = 1e-4    # Target accuracy

# Initialize with zeros
phi = np.zeros([M+1,M+1],float)
# Boundary condition
phi[0,:] = V
phi[:,0] = 0

phi = jacobi_solve(phi, target, 10000)

# Plot
import matplotlib.pyplot as plt

plt.title("Electrostatic potential")
plt.xlabel("x")
plt.ylabel("y")
CS = plt.imshow(phi, vmax=1., vmin=0.,origin="upper",extent=[0,M,0,M])
plt.colorbar(CS)
plt.show()
```

```
Jacobi method converged in 1909 iterations
```



Boundary value problem: Gauss-Seidel with overrelaxation

Gauss-Seidel method: Use newly computed ϕ_{n+1} whenever available

$$\phi_{n+1}(x,y) = \frac{\phi_n(x+a,y) + \phi_{n+1}(x-a,y) + \phi_n(x,y+a) + \phi_{n+1}(x,y-a)}{4}$$

Overrelaxation:

$$\phi_{n+1}(x,y) = \phi_n(x,y) + (1+\omega)\Delta_n\phi(x,y),$$

Combined:

$$\phi_{n+1}(x,y) = (1+\omega)\frac{\phi_n(x+a,y) + \phi_{n+1}(x-a,y) + \phi_n(x,y+a) + \phi_{n+1}(x,y-a)}{4} - \omega\,\phi_n(x,y),$$

Stable for w $<$ 1, much faster

Boundary value problem: Gauss-Seidel method

```python
import numpy as np

# Single iteration of the Jacobi method
# The new field is written into phinew
# omega >= 0 is the overelaxation parameter
def gaussseidel_iteration(phi, omega = 0):
    M = len(phi) - 1

    delta = 0.

    # New iteration
    for i in range(1,M):
        for j in range(1,M):
            phiold = phi[i,j]
            phi[i,j] = (1. + omega) * (phi[i+1,j] + phi[i-1,j] + phi[i,j+1] + phi[i,j-1])/4 - omega * phi[i,j]
            delta = np.maximum(delta, abs(phiold - phi[i,j]))

    return delta

def gaussseidel_solve(phi0, omega = 0, target_accuracy = 1e-6, max_iterations = 100):
    delta = target_accuracy + 1.
    phi = phi0.copy()
    for i in range(max_iterations):
        delta = gaussseidel_iteration(phi, omega)

        if (delta <= target_accuracy):
            print("Jacobi method converged in " + str(i+1) + " iterations")
            return phi

    print("Jacobi method failed to converge to a required precision in " + str(max_iterations) + " iterations")
    print("The error estimate is ", delta)

    return phi
```

Boundary value problem: Gauss-Seidel method

```
# Constants
M = 100          # Grid squares on a side
V = 1.0          # Voltage at top wall
target = 1e-4    # Target accuracy

# Initialize with zeros
phi = np.zeros([M+1,M+1],float)
# Boundary condition
phi[0,:] = V
phi[:,0] = 0

omega = 0.9
phi = gaussseidel_solve(phi, omega, target, 1000)

# Plot
import matplotlib.pyplot as plt

plt.title("Electrostatic potential")
plt.xlabel("x")
plt.ylabel("y")
CS = plt.imshow(phi, vmax=1., vmin=0.,origin="upper",extent=[0,M,0,M])
plt.colorbar(CS)
plt.show()
```

```
Jacobi method converged in 202 iterations
```