Partial differential equations

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np
from IPython.display import HTML
```

Voltage

Set up the data required

```
size = 50
voltages = np.zeros((size, size))
V0 = 50
cond = -0.5
voltages[10, 10] = V0
```

The partial differential equation we are solving is

$$\nabla V = -\frac{1}{\epsilon_0}\rho$$

the right hand side is `cond`. Then we can solve this with the boundaries of a 50x50 having a voltage of 0. To solve this we can use the following formula for $u_{i,j}$

$$u_{i,j} = \frac{1}{4}(u_{i,j+1} + u_{i+1,j} + u_{i,j-1} + u_{i-1,j} - cond)$$

```
for _ in range(500):
    for i in range(size):
        for j in range(size):
            u1 = voltages[i, j + 1] if j + 1 < size else 0
            u2 = voltages[i, j - 1] if j - 1 > -1 else 0
```

```
            u3 = voltages[i + 1, j] if i + 1 < size else 0
            u4 = voltages[i - 1, j] if i - 1 > -1 else 0
            voltages[i, j] = (u1 + u2 + u3 + u4 - cond) / 4
```

Then we get the gradient of this to find the electric field

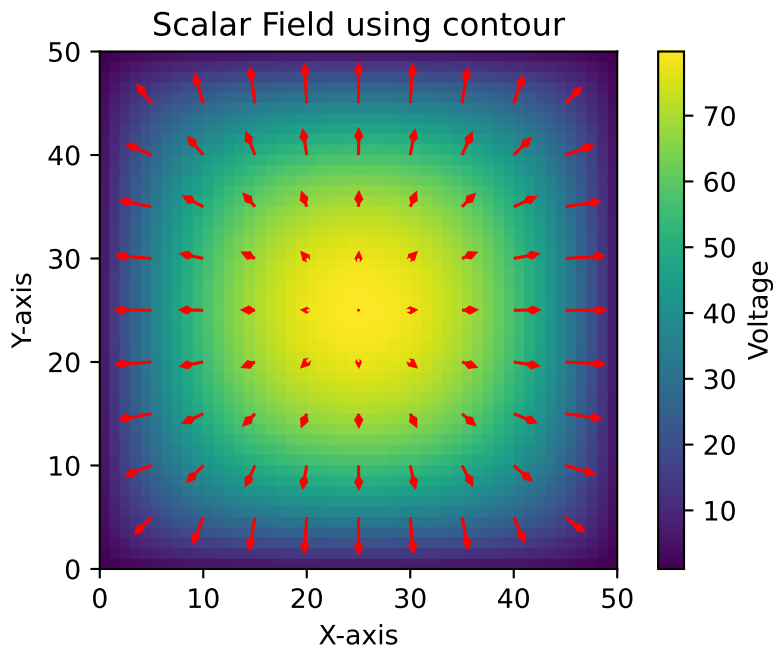
```
pos = np.array(range(size))

dy, dx = np.gradient(-voltages, pos, pos)
```

Then we graph it

```
skip = 5  # Number of points to skip
plt.figure()
plt.imshow(
    np.abs(voltages),
    extent=(0, size, 0, size),
    origin="lower",
    cmap="viridis",
)
plt.colorbar(label="Voltage")
plt.quiver(
    pos[::skip],
    pos[::skip],
    dx[::skip, ::skip],
    dy[::skip, ::skip],
    color="r",
    headlength=3,
)
plt.title("Scalar Field using contour")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

plt.show()
```



Wave equation

Set up the values for the wave equation

```
L = 2*np.pi # Length of the string
Nx = 100 # Number of spatial points
Nt = 100 # Number of time steps
dx = L/(Nx-1) # Spatial step size
v = 5 # Wave speed
dt = np.sqrt(1)*dx/v # Time step size

print(f"dt: {dt}s")
# Initialize the spring
x = np.linspace(0, L, Nx)
y = np.zeros(Nx)
y_old = np.zeros(Nx)

# initial disturbance
y_old = np.sin(x)
# Apply fixed boundary conditions
y_old[0] = y_old[Nx-1] =  0
```

```
y[0] = y[Nx-1] =  0

#  make sure the C calculated comes out to 1
C = (v*dt/dx)**2
print(C)
```

```
dt: 0.012693303650867852s
1.0
```

The partial differential equation we are solving is

$$\frac{\partial^2 y}{\partial^2 t} = \frac{T}{\mu}\frac{\partial^2 y}{\partial^2 x}$$

Then to solve the differential equation we can use the formula

$$y_{i+1,j} = 2y_{i,j} - y_{i-1,j} + C(y_{i,j+1} - 2y_{i,j} + y_{i,j-j})$$

where the i index represents time and the j index represents position. C is defined by

$$C = (T/\mu)(\Delta t)^2/(\Delta x)^2$$

for the first timestep, we dont have a previous time, so instead we use our initial velocity condition $\frac{\partial y}{\partial t} = 0$ at $t = 0$ which in turn we can use the forward difference formula and the previous formula for the differential equation to gain a new formula

$$y_{i,j} = y_{i-1,j} + \frac{1}{2}C(y_{i-1,j+1} - 2y_{i-1,j} + y_{i-1,j+1j})$$

One important thing to note is that for the numerical approximation to be stable, $C <= 1$ where the theoretical solution is when equivalence holds

```
# compute initial timestep assuming initial velocity is 0
y[1:-1] = y_old[1:-1] + 0.5 * C * (y_old[2:] - 2 * y_old[1:-1] + y_old[:-2])
def update_string(y_old, y):
  y_new = np.zeros(Nx)
  y_new[1:-1] = 2*y[1:-1] - y_old[1:-1] + C*(y[2:] - 2*y[1:-1] + y[:-2])
  # Update arrays for next iteration
  y_old[:] = y[:]
  y[:] = y_new[:]
```

Plot the data

```
# Set up the plot
fig, ax = plt.subplots()
line, = ax.plot(x, y)
ax.set_ylim(-1.5, 1.5)
ax.set_xlim(0, L)
ax.set_title('Vibrating String')
ax.set_xlabel('Position (m)')
ax.set_ylabel('Displacement (m)')
# Animation function
skip = 2
def animate(frame):
    global y_old, y, y_new, skip
    for _ in range(skip):
      update_string(y_old, y)
    line.set_ydata(y)
    return line,
# Create animation
ani = FuncAnimation(fig, animate, frames=Nt, blit=True, interval=dt*1000*skip)
ani.save('standing_wave.mp4', writer='ffmpeg', fps=1/(dt*skip))
HTML(ani.to_jshtml())
```

```
<IPython.core.display.HTML object>
```