Introduction:

The finite difference method (FDM) is a numerical method used to solve partial differential equations for which there is no analytical solution. It involves approximating the derivative of the unknown function using a finite difference equation in a discrete set of points on a grid. The points are usually divided in a uniform manner (or equally spaced points to ensure that the discretization is consistent and the approximation is accurate. With a uniform grid, the spacing between the points is constant throughout the domain which maintains the regularity of the problem.

To use the FDM, we first need to discretize the PDE into a set of algebraic equations. This is done by replacing the partial derivatives in the PDE with their corresponding finite difference approximations. The resulting system of equations can then be solved numerically using *matrix algebra*. The FDM is based on the Taylor series expansion of the unknown function around a point in space and time. By truncating the series after a finite number of terms, we obtain a finite difference equation that approximates the derivative at that point. Higher-order finite difference schemes provide more accurate approximations but may involve more neighboring grid points. This process is repeated for all points on the grid, resulting in a set of equations that can be solved simultaneously. The coefficients of these equations are based on the scheme used (forward difference, backward difference, or central difference).

The accuracy of the FDM depends on the step size of the grid, which determines the distance between grid points. A smaller step size leads to a more accurate solution, but also increases the computational cost of the method. Additionally, the FDM can only be used for regular-shaped domains and requires uniform grid spacing.

Application:

Now let's use the FTCS (Forward Time Centered Space) method to solve a PDE that we are familiar with: heat equation.

The key is the recurrence formula, a discretized form of the PDE that allows us to build the matrix. Luckily, we found a detailed deduction to the recurrence formula on *matlabgeeks:*

a. Approximate the 1st order time derivative with forward finite difference approximation of $\vartheta(\Delta t)$ at the grid point *(i,k)*:

$$\left.\frac{\partial U}{\partial t}\right|_{i,k} = \frac{U_i^{K+1} - U_i^K}{\Delta t} \quad [Eq\ 2]$$

The indices (*i,k*) at the LHS derivative operator means the discretization is at the node *(i,k).* As this is a time derivative, the space *i* index remain constant. The derivative approximation can be obtained directly from the 'lazy solver' tables.

b. Approximate the 2nd order space derivative with central difference approximation of $\vartheta(\Delta x^2)$ at the grid point *(i,k)* using the 'lazy solver' tables:

$$\left.\frac{\partial^2 U}{\partial x^2}\right|_{i,k} = \frac{U_{i+1}^K - 2U_i^K + U_{i-1}^K}{\Delta x^2} \quad [Eq\ 3]$$

The time *k* index remains constant as this is a space derivative.

c. Plug [Eq 2] and [Eq 3] in the original equation [Eq 1] to get the recurrence formula:

$$\frac{U_i^{K+1} - U_i^K}{\Delta t} = k\frac{U_{i+1}^K - 2U_i^K + U_{i-1}^K}{\Delta x^2} \quad [Eq\ 4]$$

which after simplification yields the recurrence formula:

$$U_i^{K+1} = U_i^K + \lambda\left(U_{i+1}^K - 2U_i^K + U_{i-1}^K\right) \quad [Eq\ 4.a]$$

where $\lambda = k\Delta t/\Delta x^2$ Clustering variables with same indices in ascending order offers another way to express the recurrence formula which is more suitable for a computer program:

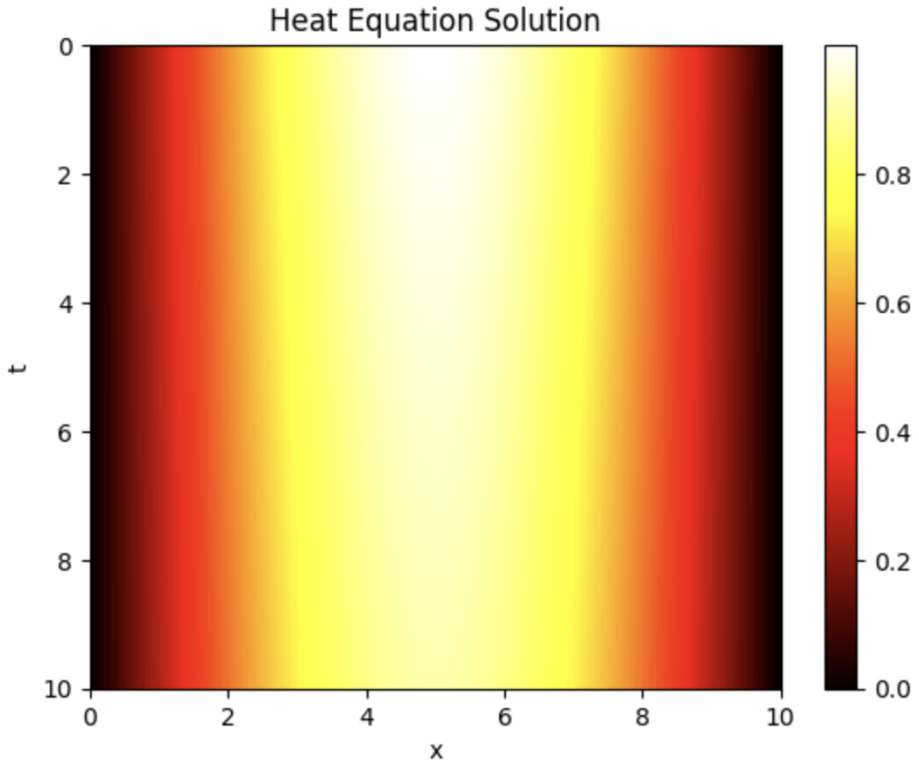
$$U_i^{K+1} = \lambda U_{i-1}^K + (1 - 2\lambda)U_i^K + \lambda U_{i+1}^K \quad [Eq\ 4.b]$$

#Code below

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

# Set up initial parameters
nx = 100  # number of spatial grid points
nt = 1000  # number of time grid points
dt = 0.01  # time step
dx = 0.1  # spatial step
alpha = 0.1  # thermal diffusivity
```

```
# Set up initial conditions: at time 0, the temperature at each x is set to be a sine wave
u = np.zeros((nt, nx))
u[0, :] = np.sin(np.pi * np.linspace(0, 1, nx))

# Apply the FTCS (Forward Time Centered Space) finite difference method and using the recurrence formula above Eq.4.a
for i in range(1, nt):
    for j in range(1, nx - 1):
        u[i, j] = u[i-1, j] + alpha * dt / dx**2 * (u[i-1, j+1] - 2 * u[i-1, j] + u[i-1, j-1])

# Plot the results
plt.imshow(u, cmap='hot', aspect='auto', extent=[0, 10, nt*dt, 0])
plt.xlabel('x')
plt.ylabel('t')
plt.title('Heat Equation Solution')
plt.colorbar()
plt.show()
```

Here is the result:



Reference:

https://matlabgeeks.weebly.com/uploads/8/0/4/8/8048228/chap_13_partial_differential_equations-v7.pdf