



# Computational Physics (PHYS6350)

## *Lecture 16: Partial Differential Equations Part II*

- Initial value problems
  - Heat equation
  - Wave equation

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

**March 20, 2025**

**Instructor:** Volodymyr Vovchenko ([vvovchenko@uh.edu](mailto:vvovchenko@uh.edu))

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

# Initial value problem in PDEs

---

Many PDEs describe time evolution of fields  $u(t, x)$

For example *heat equation* describing the temperature profile

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2}$$

This equation describes the time evolution of  $u(t, x)$  given initial profile

$$u(t = 0, x) = u_0(x),$$

and boundary conditions

$$\begin{aligned} u(t, x = 0) &= u_{\text{left}}(t), \\ u(t, x = L) &= u_{\text{right}}(t). \end{aligned}$$

If boundary conditions are static, the solution will approach a stationary profile at large times

# Finite difference approach to heat equation

---

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2}$$

First, one discretizes the spatial coordinate into a grid with  $N+1$  points, i.e.

$$x_k = ak, \quad k = 0 \dots N, \quad a = L/N,$$

The spatial 2nd derivative is approximate with central difference

$$\frac{\partial^2 u(t, x)}{\partial x^2} \approx \frac{u(t, x + a) - 2u(t, x) + u(t, x - a)}{a^2}.$$

How to discretize time derivative?

Three common options:

- FTCS scheme
- Implicit scheme
- Crank-Nicolson method

# Finite difference approach to heat equation: FTCS scheme

---

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2},$$

FTCS (Forward Time Centered Space) scheme

Time derivative approximated by forward difference

$$\frac{\partial u(t, x)}{\partial t} \approx \frac{u(t + h, x) - u(t, x)}{h}$$

This gives the following discretized PDE

$$\frac{u(t + h, x) - u(t, x)}{h} = D \frac{u(t, x + a) - 2u(t, x) + u(t, x - a)}{a^2},$$

The method is explicit: to evaluate  $u(t+h, x)$  one only needs  $u(t, x)$  at the present time

Discretized form

$$u_k^{n+1} = u_k^n + r (u_{k+1}^n - 2u_k^n + u_{k-1}^n), \quad k = 1 \dots N - 1.$$

$$r \equiv \frac{Dh}{a^2}$$

# FTCS scheme for heat equation

---

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2},$$

```
# Single iteration of the FTCS scheme in the time direction
# r = Dh/a^2 is the dimensionless parameter
def heat_FTCS_iteration(u, r):
    N = len(u) - 1

    unew = np.empty_like(u)

    # Boundary conditions
    unew[0] = u[0]
    unew[N] = u[N]

    # FTCS scheme
    for i in range(1,N):
        unew[i] = u[i] + r * (u[i+1] - 2 * u[i] + u[i-1])

    return unew
```

```
# Perform nsteps FTCS time iterations for the heat equation
# u0: the initial profile
# h: the size of the time step
# nsteps: number of time steps
# a: the spatial cell size
# D: the diffusion constant
def heat_FTCS_solve(u0, h, nsteps, a, D = 1.):
    u = u0.copy()
    r = h * D / a**2
    for i in range(nsteps):
        u = heat_FTCS_iteration(u, r)

    return u
```

# Example

Let us consider Example 9.3 from M. Newman, *Computational Physics*:

We have a 1 cm long steel container, initially at a temperature 20° C. It is placed in a bath of cold water at 0° C and filled on top with hot water at 50° C. Our goal is to calculate the temperature profile as a function of time. The thermal diffusivity constant for stainless steel is  $D = 4.25 \cdot 10^{-6} \text{ m}^2 \text{ s}^{-1}$ .

We will calculate the profile at times  $t = 0.01 \text{ s}$ ,  $0.1 \text{ s}$ ,  $0.4 \text{ s}$ ,  $1 \text{ s}$ , and  $10 \text{ s}$ .

```
# Constants
L = 0.01      # Thickness of steel in meters
D = 4.25e-6   # Thermal diffusivity
N = 100       # Number of divisions in grid
a = L/N       # Grid spacing
h = 1e-4      # Time-step (in s)

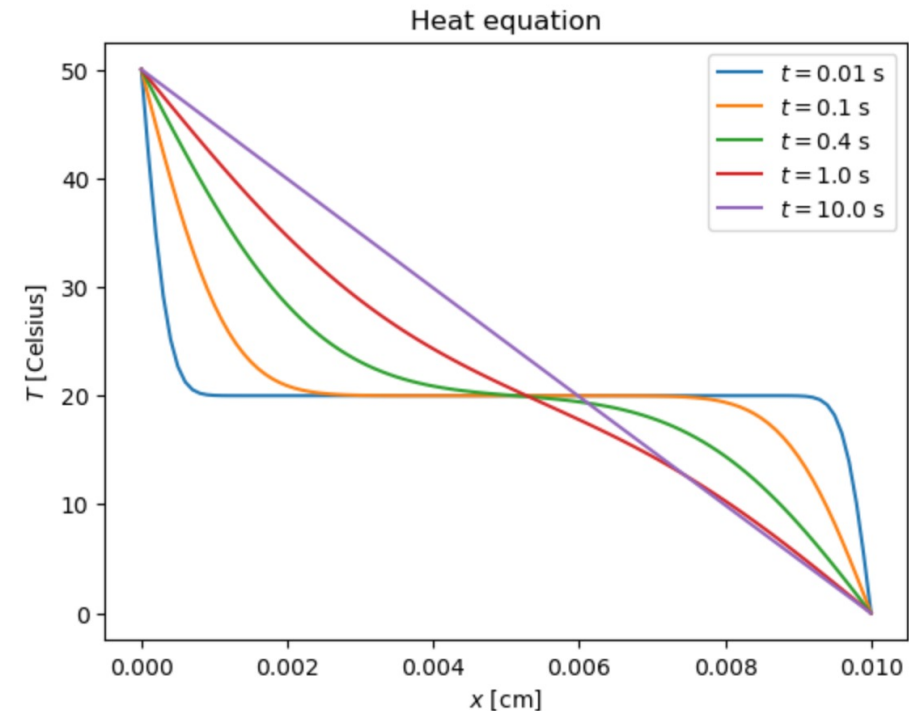
print("Solving the heat equation with FTCS scheme")
print("r = h*D/a^2 =", h*D/a**2)

Tlo = 0.0     # Low temperature in Celsius
Tmid = 20.0   # Intermediate temperature in Celsius
Thi = 50.0    # High temperature in Celsius

# Initialize
u = np.zeros([N+1], float)
# Initial temperature
u[1:N] = Tmid
# Boundary conditions
u[0] = Thi
u[N] = Tlo

current_time = 0.
for time in times:
    nsteps = round((time - current_time)/h)
    u = heat_FTCS_solve(u, h, nsteps, a, D)
    profiles.append(u.copy())
    current_time = time
```

Solving the heat equation with FTCS scheme  
 $r = h*D/a^2 = 0.0425$



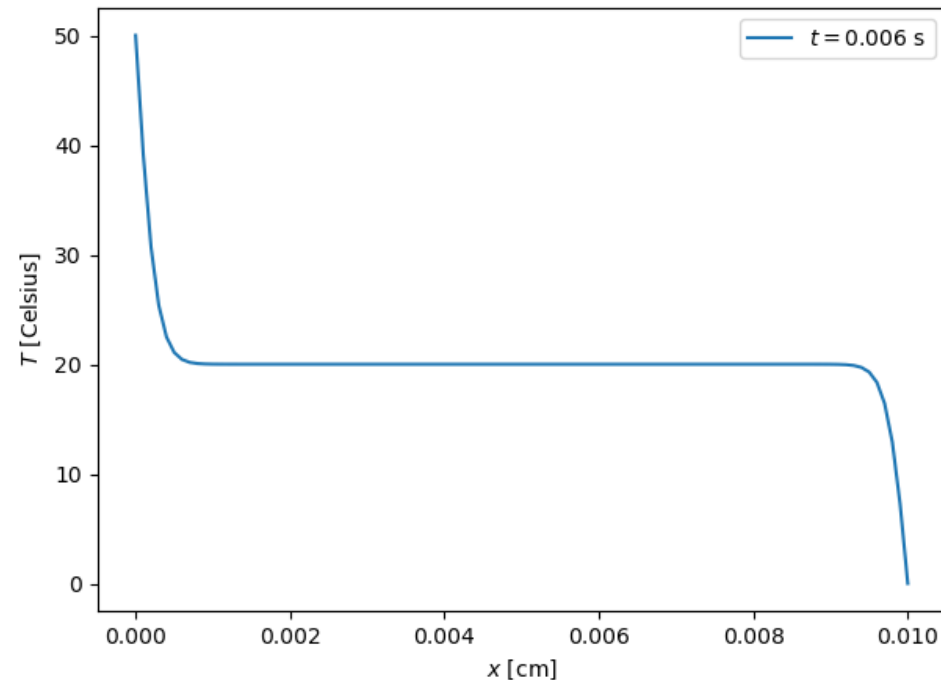
# Animation

Let us consider Example 9.3 from M. Newman, *Computational Physics*:

We have a 1 cm long steel container, initially at a temperature 20° C. It is placed in a bath of cold water at 0° C and filled on top with hot water at 50° C. Our goal is to calculate the temperature profile as a function of time. The thermal diffusivity constant for stainless steel is  $D = 4.25 \cdot 10^{-6} \text{ m}^2 \text{ s}^{-1}$ .

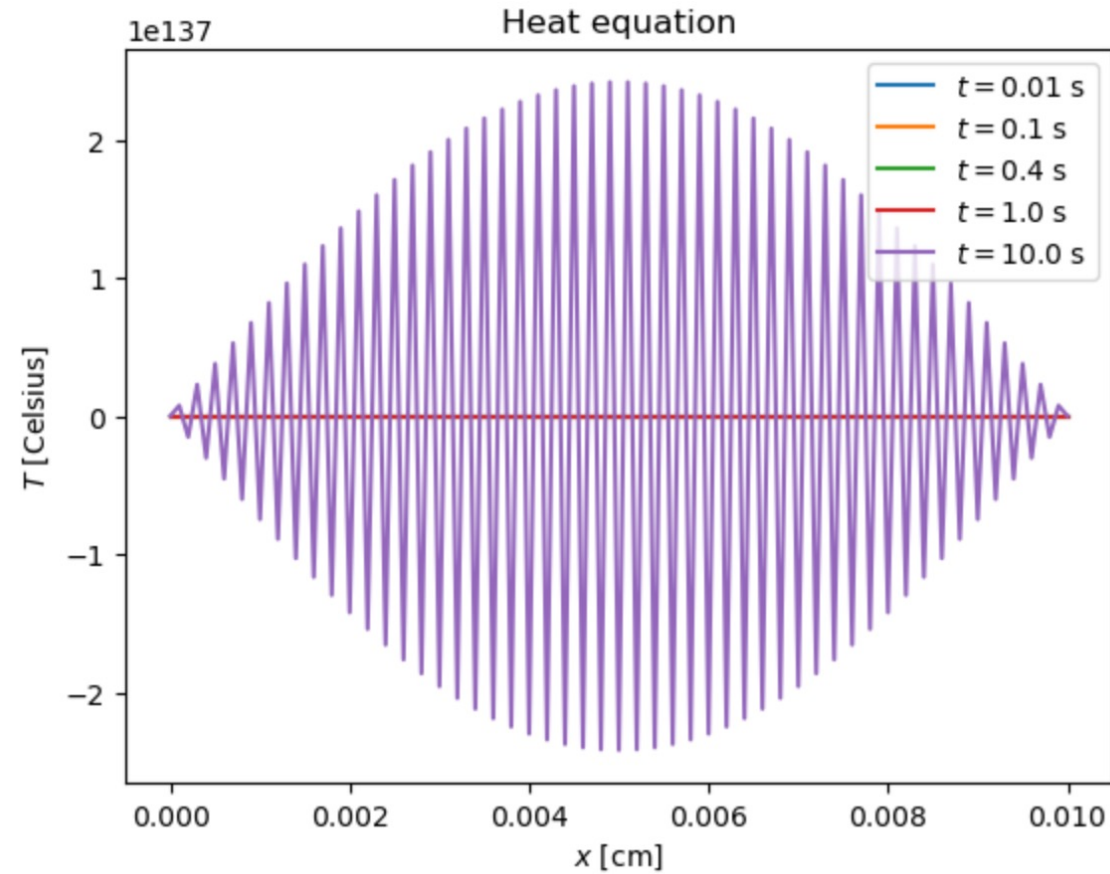
We will calculate the profile at times  $t = 0.01 \text{ s}$ ,  $0.1 \text{ s}$ ,  $0.4 \text{ s}$ ,  $1 \text{ s}$ , and  $10 \text{ s}$ .

Heat equation



# Try a larger time step

Solving the heat equation with FTCS scheme  
 $r = h \cdot D / a^2 = 0.5099999999999999$



The method is unstable if  $r > 0.5$



# Stability analysis: Neumann method

---

Recall Fourier transform  $u(t, x) = \sum_k c_k(t) e^{ikx}$

Analyze the following solutions:  $u(t, x) = c_k(t) e^{ikx}$

Plugging into

$$\frac{u(t+h, x) - u(t, x)}{h} = D \frac{u(t, x+a) - 2u(t, x) + u(t, x-a)}{a^2}.$$

one gets

$$u(t+h, x) = [1 - 4r \sin^2(ka/2)] c_k(t) e^{ikx}$$

i.e.

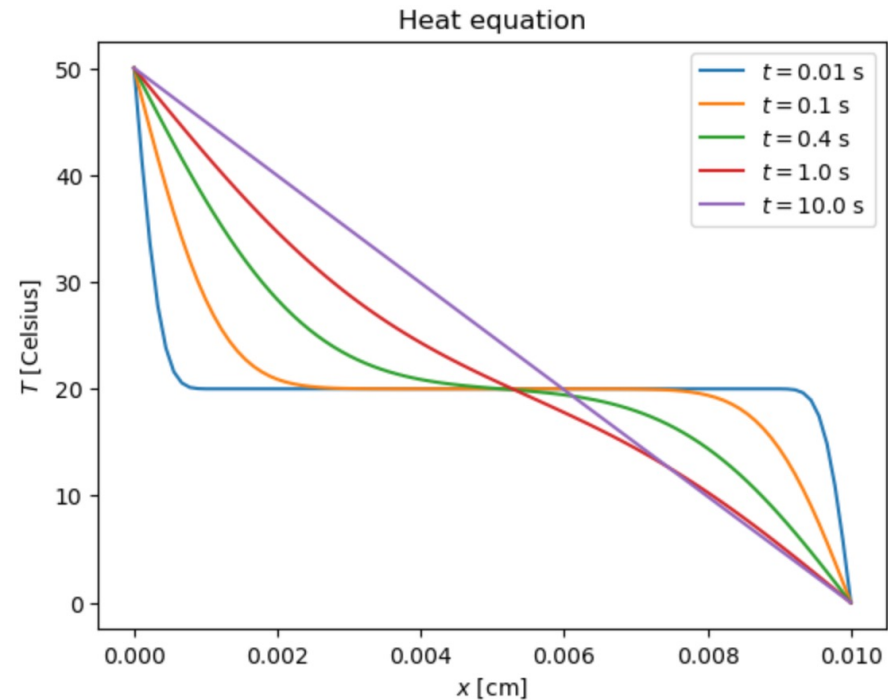
$$c_k(t+h) = [1 - 4r \sin^2(ka/2)] c_k(t), \quad c_k^{n+1} = [1 - 4r \sin^2(ka/2)]^n c_k^0$$

The method is stable if  $r < 1/2$

# Try a larger space step to compensate for large time step

```
# Constants
L = 0.01      # Thickness of steel in meters
D = 4.25e-6   # Thermal diffusivity
N = 90        # Number of divisions in grid
a = L/N       # Grid spacing
h = 1.2e-3    # Time-step (in s)
```

Solving the heat equation with FTCS scheme  
 $r = h \cdot D / a^2 = 0.41309999999999986$



# Implicit scheme

---

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2},$$

Time derivative approximated by backward difference

$$\frac{\partial u(t+h, x)}{\partial t} \approx \frac{u(t+h, x) - u(t, x)}{h}$$

This gives the following discretized PDE

$$\frac{u(t+h, x) - u(t, x)}{h} = D \frac{u(t+h, x+a) - 2u(t+h, x) + u(t+h, x-a)}{a^2}$$

Discretized form

$$u_k^{n+1} = u_k^n + r (u_{k+1}^{n+1} - 2u_k^{n+1} + u_{k-1}^{n+1}), \quad k = 1 \dots N-1$$

Tridiagonal system of linear equations at each step

$$-ru_{k-1}^{n+1} + (1 + 2r)u_k^{n+1} - ru_{k+1}^{n+1} = u_k^n, \quad k = 1 \dots N-1$$

Neumann stability analysis: method is stable for any  $r$

# Implicit scheme for heat equation

---

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2},$$

```
# Single iteration of the FTCS scheme in the time direction
# The new field is written into unew
# r = Dh/a^2 is the dimensionless parameter
def heat_implicit_iteration(u, r):
    N = len(u) - 1

    unew = np.empty_like(u)

    # Boundary conditions
    unew[0] = u[0]
    unew[N] = u[N]

    d = np.full(N-1, 1+2.*r)
    ud = np.full(N-1, -r)
    ld = np.full(N-1, -r)
    v = np.array(u[1:N])
    v[0] += r * u[0]
    v[N-2] += r * u[N]

    unew[1:N] = linsolve_tridiagonal(d,ld,ud,v)

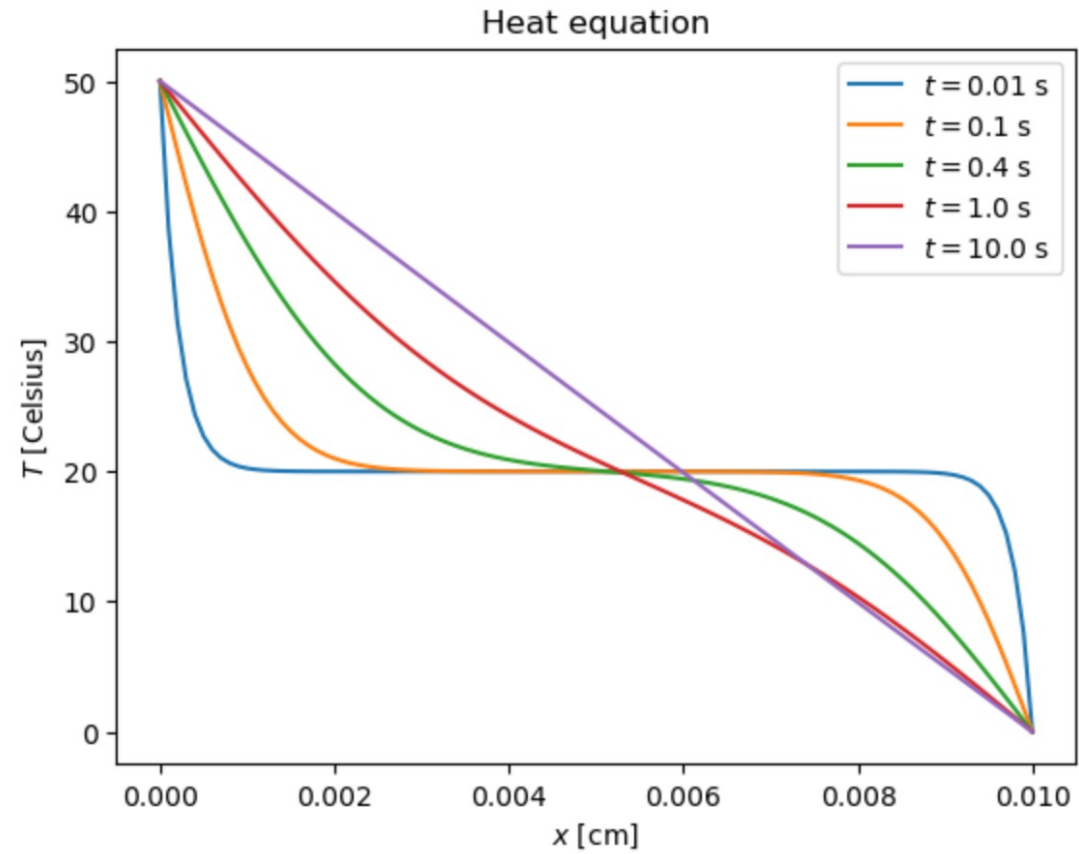
    return unew
```

```
# Perform nsteps FTCS time iterations for the heat equation
# u0: the initial profile
# h: the size of the time step
# nsteps: number of time steps
# a: the spatial cell size
# D: the diffusion constant
def heat_implicit_solve(u0, h, nsteps, a, D = 1.):
    u = u0.copy()
    r = h * D / a**2
    # print("Heat equation with r =", r)
    for i in range(nsteps):
        u = heat_implicit_iteration(u, r)

    return u
```

# Implicit scheme: large time step

Solving the heat equation with implicit scheme  
 $r = h \cdot D / a^2 = 4.25$



# Crank-Nicolson scheme

---

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2},$$

“Average” between forward and backward difference

$$\frac{\partial u(t, x)}{\partial t} \approx \frac{1}{2} \left[ D \frac{\partial^2 u(t + h, x)}{\partial x^2} + D \frac{\partial^2 u(t, x)}{\partial x^2} \right]$$

Essentially a trapezoidal rule for the time integration (more accurate than forward/backward differences)

Discretized form

$$\frac{u(t + h, x) - u(t, x)}{h} = \frac{D}{2} \frac{u(t + h, x + a) - 2u(t + h, x) + u(t + h, x - a)}{a^2} + \frac{D}{2} \frac{u(t, x + a) - 2u(t, x) + u(t, x - a)}{a^2}.$$

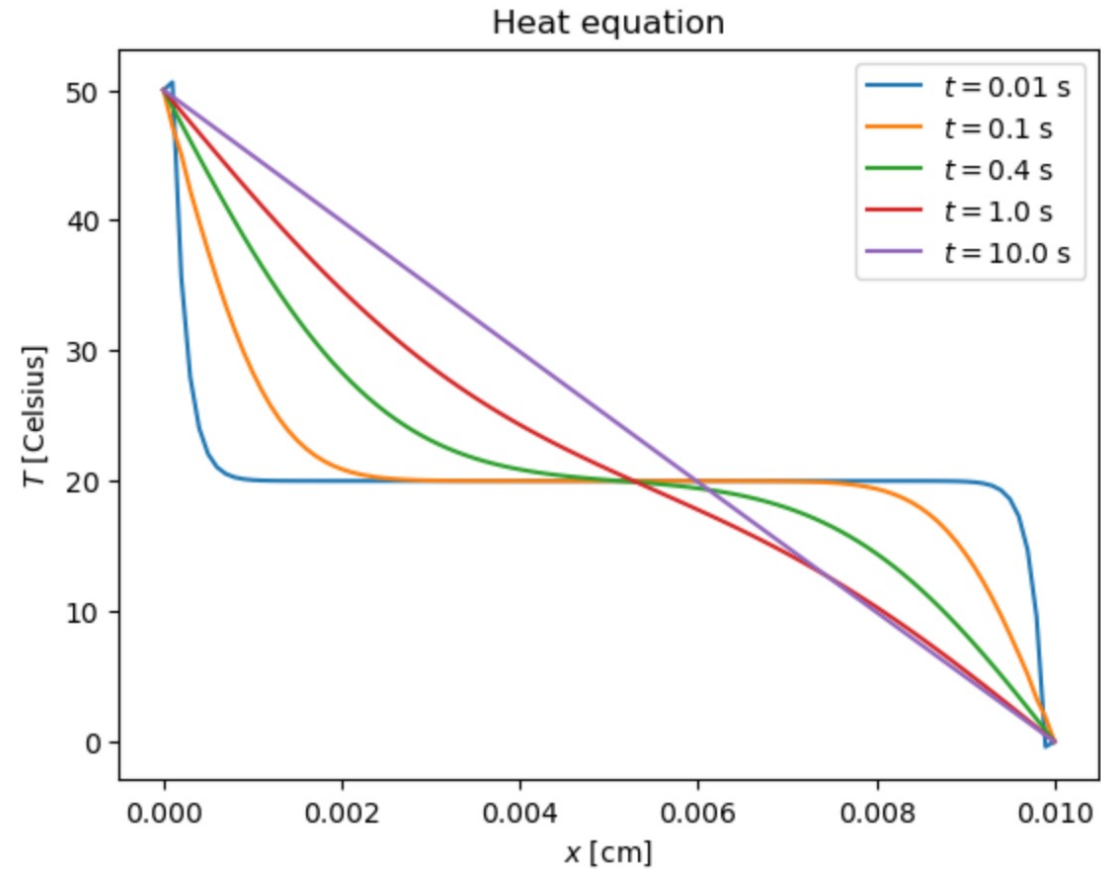
Tridiagonal system of linear equations at each step:

$$-ru_{k-1}^{n+1} + 2(1 + r)u_k^{n+1} - ru_{k+1}^{n+1} = ru_{k-1}^n + 2(1 - r)u_k^n + ru_{k+1}^n, \quad k = 1 \dots N - 1.$$

Neumann stability analysis: method is stable for any  $r$

# Crank-Nicolson scheme: large time step

Solving the heat equation with Crank-Nicolson scheme  
 $r = h \cdot D / a^2 = 4.25$



# Heat equation in two dimensions

---

In two dimensions, the heat equation reads:

$$\frac{\partial u}{\partial t} = D \left[ \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right].$$

This equation describes the time evolution of  $u(t, x, y)$  given the initial profile and boundary conditions:

$$\begin{aligned} u(t=0, x, y) &= u_0(x, y), \\ u(t, x=0, y) &= u_{\text{left}}(t; y), \\ u(t, x=L, y) &= u_{\text{right}}(t; y), \\ u(t, x=0, y) &= u_{\text{bottom}}(t; x), \\ u(t, x=L, y) &= u_{\text{top}}(t; x). \end{aligned}$$

Now, we perform discretization in both  $x$  and  $y$  directions. Taking the same step size  $a$  in both directions, we obtain the following discretized FTCS scheme:

$$u_{i,j}^{n+1} = u_{i,j}^n + r(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n) + r(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n), \quad i = 1 \dots N-1, \quad j = 1 \dots M-1.$$

Here, as before:

$$r \equiv \frac{Dh}{a^2}, \quad N = L_x/a, \quad M = L_y/a,$$

and

$$u_{i,j}^n = u(t + hn, ai, aj).$$



# Heat equation in two dimensions: FTCS scheme

---

```
# Single iteration of the 2D FTCS scheme in the time direction
# r = Dh/a^2 is the dimensionless parameter
def heat_FTCS_iteration_2D(u, r):
    N, M = u.shape

    unew = np.empty_like(u)

    # Boundary conditions
    unew[ 0, :] = u[ 0, :]
    unew[N-1, :] = u[N-1, :]
    unew[:, 0] = u[:, 0]
    unew[:, M-1] = u[:, M-1]

    # FTCS scheme
    for i in range(1, M-1):
        for j in range(1, N-1):
            unew[i, j] = u[i, j] + r * (u[i+1, j] - 2 * u[i, j] + u[i-1, j]) + r * (u[i, j+1] - 2 * u[i, j] + u[i, j-1])

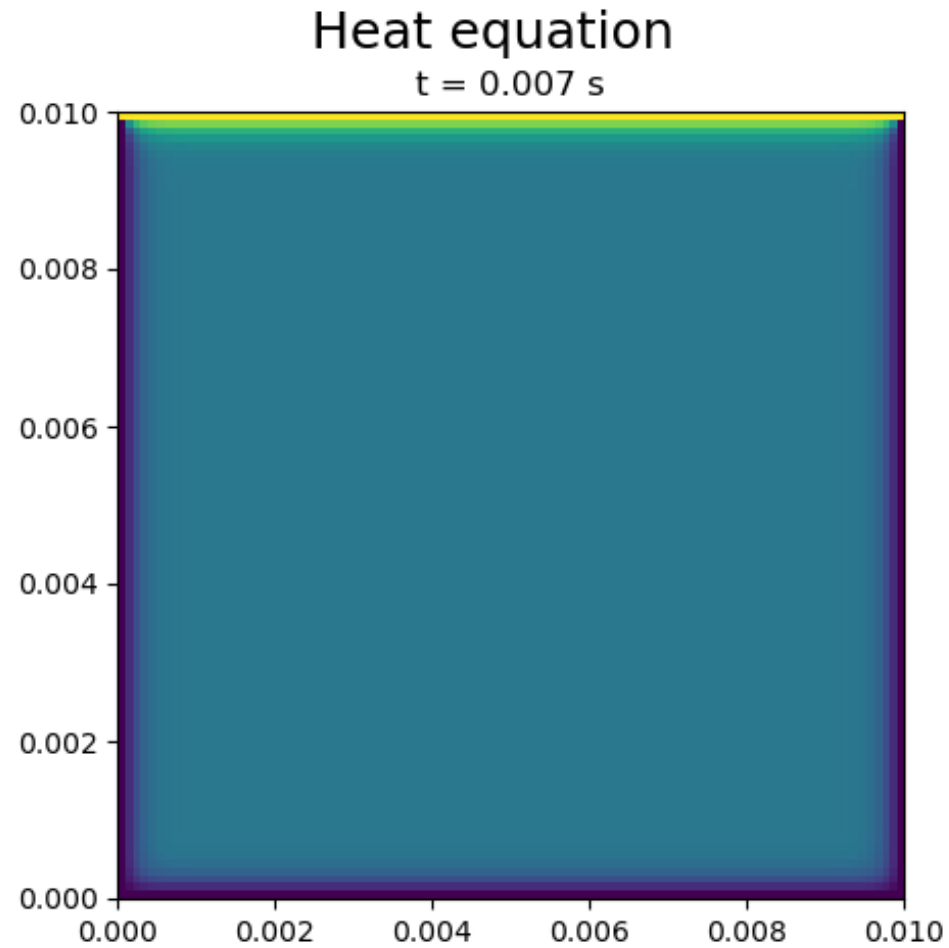
    return unew

# Perform nsteps 2D FTCS time iterations for the heat equation
# u0: the initial profile
# h: the size of the time step
# nsteps: number of time steps
# a: the spatial cell size
# D: the diffusion constant
def heat_FTCS_solve_2D(u0, h, nsteps, a, D = 1.):
    u = u0.copy()
    r = h * D / a**2
    for i in range(nsteps):
        u = heat_FTCS_iteration_2D(u, r)

    return u
```

# Heat equation in two dimensions: FTCS scheme

---



# Wave equation

---

The **wave equation** is an example of a second-order linear PDE describing waves and standing wave fields. In one dimension, it reads:

$$\frac{\partial^2 \phi}{\partial t^2} = v^2 \frac{\partial^2 \phi}{\partial x^2}. \quad \text{wave equation}$$

Since it is a second-order PDE, it is supplemented by initial conditions for both  $\phi(t = 0, x)$  and  $\phi'_t(t = 0, x)$ :

$$\begin{aligned} \phi(t = 0, x) &= \phi_0(x), \\ \phi'_t(t = 0, x) &= \phi'_0(x). \end{aligned} \quad \text{initial conditions}$$

The boundary conditions can be either Dirichlet,

$$\begin{aligned} \phi(t, x = 0) &= \phi_{\text{left}}(t), \\ \phi(t, x = L) &= \phi_{\text{right}}(t), \end{aligned} \quad \text{boundary conditions (Dirichlet)}$$

or Neumann,

$$\begin{aligned} \phi'_x(t, x = 0) &= \phi'_{\text{left}}(t), \\ \phi'_x(t, x = L) &= \phi'_{\text{right}}(t), \end{aligned} \quad \text{boundary conditions (Neumann)}$$

We shall focus on the Dirichlet form.

# Finite difference approach

---

## Finite Difference Approach

To deal with the second-order time derivative, we denote:

$$\psi(t, x) \equiv \frac{\partial \phi}{\partial t}.$$

This way, we are dealing with a [system of first-order \(in t\) PDEs](#):

$$\begin{aligned}\frac{\partial \phi}{\partial t} &= \psi(t, x), \\ \frac{\partial \psi}{\partial t} &= v^2 \frac{\partial^2 \phi}{\partial x^2}.\end{aligned}$$

To apply the finite difference method, we first approximate the derivative  $\partial^2 \phi / \partial x^2$  by the lowest-order central difference, just like for the heat equation:

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

To solve the PDEs numerically, we apply the same procedure as for the heat equation, but for  $\phi(t, x)$  and  $\psi(t, x)$  simultaneously.

**FTCS scheme:**

$$\begin{aligned}\phi_k^{n+1} &= \phi_k^n + h\psi_k^n, \\ \psi_k^{n+1} &= \psi_k^n + r(\phi_{k+1}^n - 2\phi_k^n + \phi_{k-1}^n), \quad k = 1 \dots N - 1.\end{aligned}$$

$$\begin{aligned}\phi(t = nh, x = ka) &= \phi_k^n \\ \psi(t = nh, x = ka) &= \psi_k^n\end{aligned}$$

# Wave equation

---

```
# Single iteration of the FTCS scheme in the time direction
# h is the time step
# r = Dh/a^2 is the dimensionless parameter
def wave_FTCS_iteration(phi, psi, h, r):
    N = len(phi) - 1

    phinew = np.empty_like(phi)
    psinew = np.empty_like(psi)

    # Boundary conditions (here static Dirichlet)
    phinew[0] = phi[0]
    phinew[N] = phi[N]
    psinew[0] = 0.
    psinew[N] = 0.

    # FTCS scheme
    for i in range(1,N):
        phinew[i] = phi[i] + h * psi[i]
        psinew[i] = psi[i] + r * (phi[i+1] - 2 * phi[i] + phi[i-1])

    return phinew, psinew
```

```
# Perform nsteps FTCS time iterations for the heat equation
# u0: the initial profile
# h: the size of the time step
# nsteps: number of time steps
# a: the spatial cell size
# D: the diffusion constant
def wave_FTCS_solve(phi0, psi0, h, nsteps, a, v = 1.):
    phi = phi0.copy()
    psi = psi0.copy()
    r = h * v**2 / a**2
    for i in range(nsteps):
        phi, psi = wave_FTCS_iteration(phi, psi, h, r)

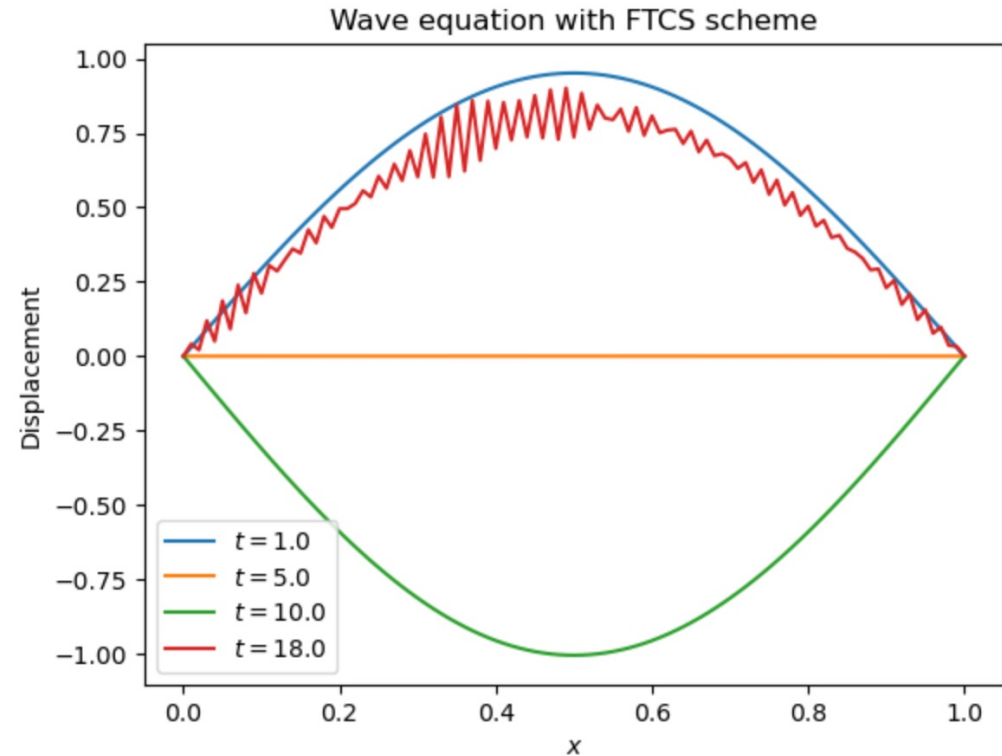
    return phi, psi
```

# Wave equation

```
# Constants
L = 1          # Length
v = 0.1        # Wave propagation speed
N = 100        # Number of divisions in grid
a = L/N        # Grid spacing
h = 1e-2       # Time-step

print("Solving the wave equation with FTCS scheme")
print("r = h*v^2/a^2 =", h*v**2/a**2)

# Initialize
phi = np.array([np.sin(k*np.pi/N) for k in range(N+1)])
psi = np.zeros([N+1], float)
```



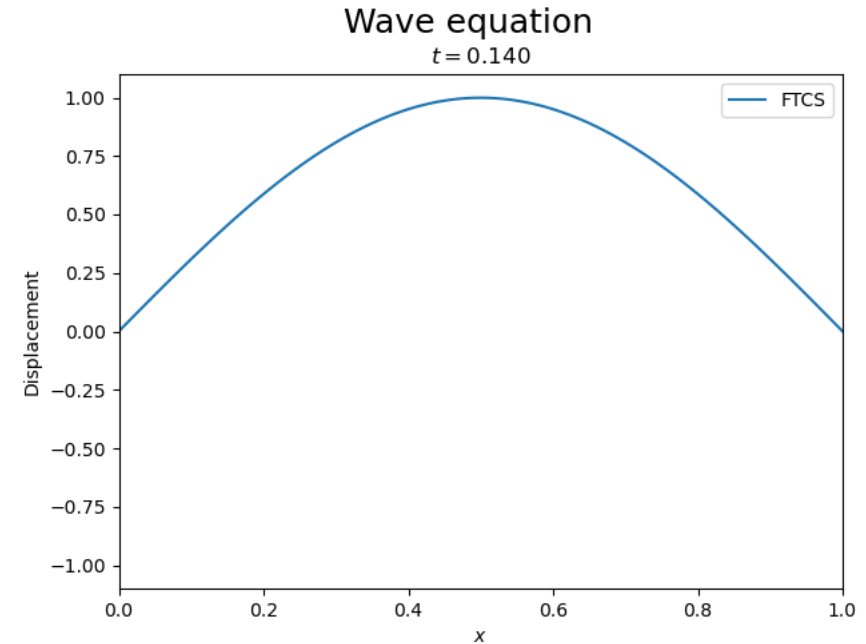
FTCS scheme is unstable

# Wave equation

```
# Constants
L = 1          # Length
v = 0.1        # Wave propagation speed
N = 100        # Number of divisions in grid
a = L/N        # Grid spacing
h = 1e-2       # Time-step

print("Solving the wave equation with FTCS scheme")
print("r = h*v^2/a^2 =", h*v**2/a**2)

# Initialize
phi = np.array([np.sin(k*np.pi/N) for k in range(N+1)])
psi = np.zeros([N+1], float)
```



FTCS scheme is unstable

# Wave equation: other schemes

---

## Implicit Scheme

$$\begin{aligned}\phi_k^{n+1} &= \phi_k^n + h\psi_k^{n+1}, \\ \psi_k^{n+1} &= \psi_k^n + r(\phi_{k+1}^{n+1} - 2\phi_k^{n+1} + \phi_{k-1}^{n+1}), \quad k = 1 \dots N-1.\end{aligned}$$

Substituting the first equation into the second one gives the tridiagonal system of linear equations for  $\psi_k^{n+1}$ :

$$-rh\psi_{k+1}^{n+1} + (1 + 2rh)\psi_k^{n+1} - rh\psi_{k-1}^{n+1} = \psi_k^n + r(\phi_{k+1}^n - 2\phi_k^n + \phi_{k-1}^n), \quad k = 1 \dots N-1$$

Stable, but has exponential decay (waves don't propagate forever)

## Crank-Nicolson Scheme

$$\begin{aligned}\phi_k^{n+1} &= \phi_k^n + \frac{h}{2}[\psi_k^{n+1} + \psi_k^n], \\ \psi_k^{n+1} &= \psi_k^n + \frac{r}{2}(\phi_{k+1}^{n+1} - 2\phi_k^{n+1} + \phi_{k-1}^{n+1}) + \frac{r}{2}(\phi_{k+1}^n - 2\phi_k^n + \phi_{k-1}^n), \quad k = 1 \dots N-1.\end{aligned}$$

Substituting the first equation into the second one gets the tridiagonal system of linear equations for  $\psi_k^{n+1}$ :

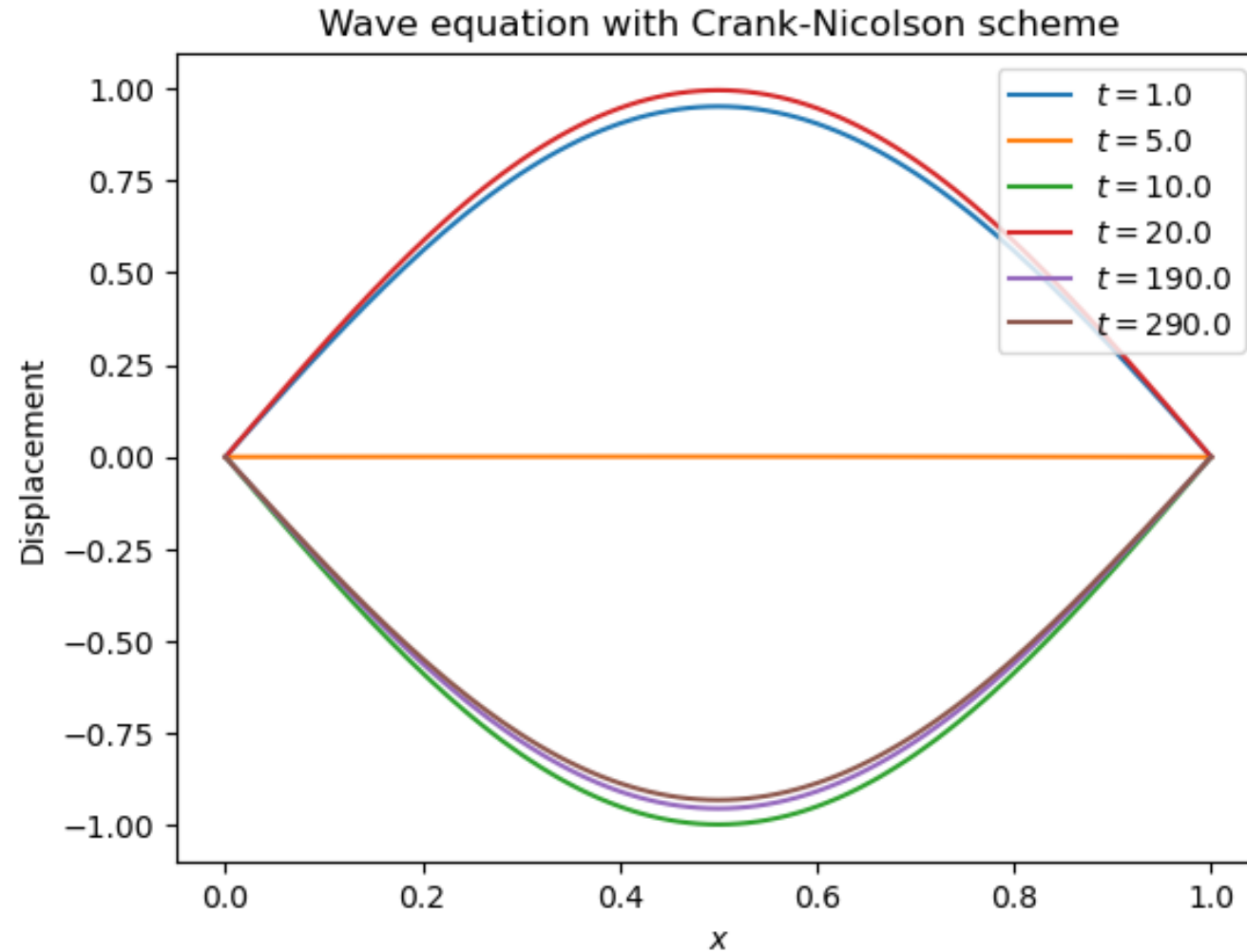
$$-rh\psi_{k+1}^{n+1} + 2(1 + rh)\psi_k^{n+1} - rh\psi_{k-1}^{n+1} = 2\psi_k^n + 2r(\phi_{k+1}^n - 2\phi_k^n + \phi_{k-1}^n) + rh(\psi_{k+1}^n - 2\psi_k^n + \psi_{k-1}^n), \quad k = 1 \dots N-1.$$

Stable, no growth or decay



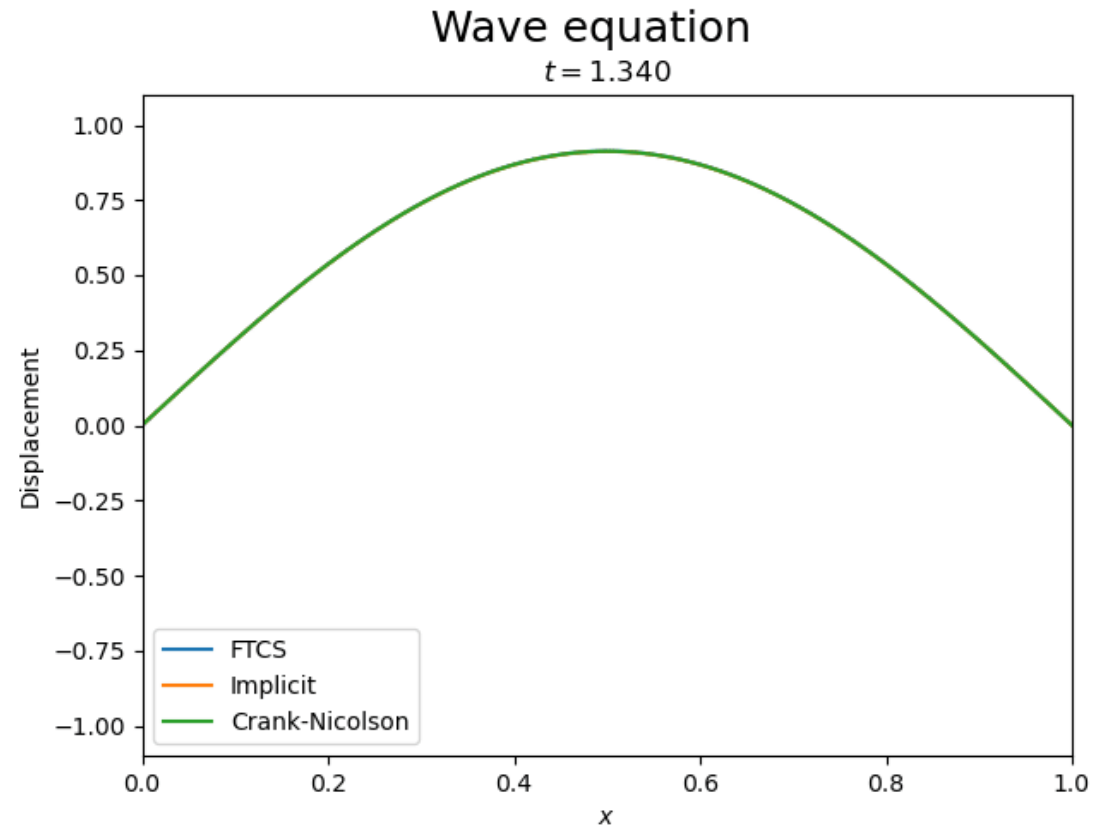
# Wave equation with Crank-Nicolson scheme

---



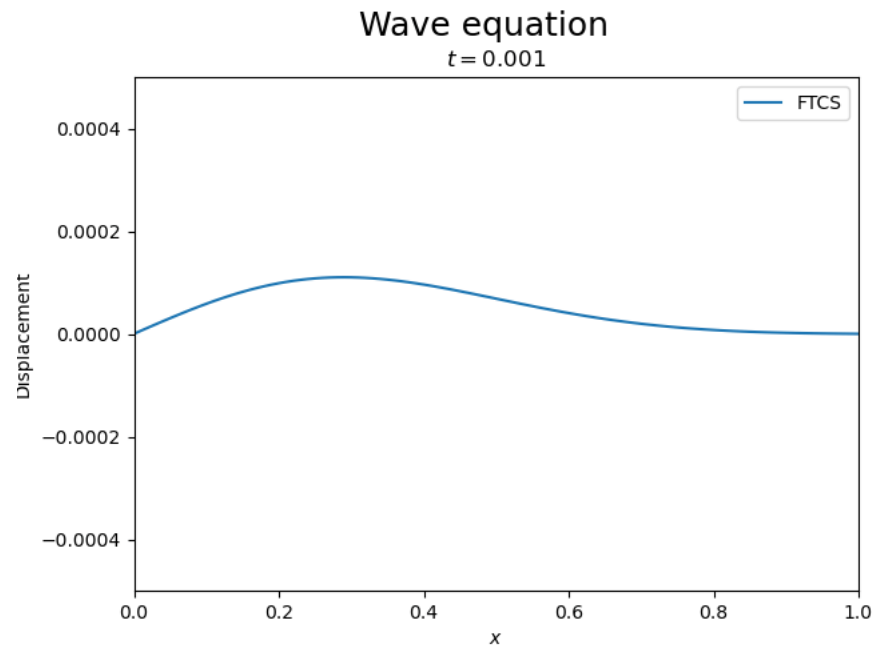
# Wave equation: Comparison

---



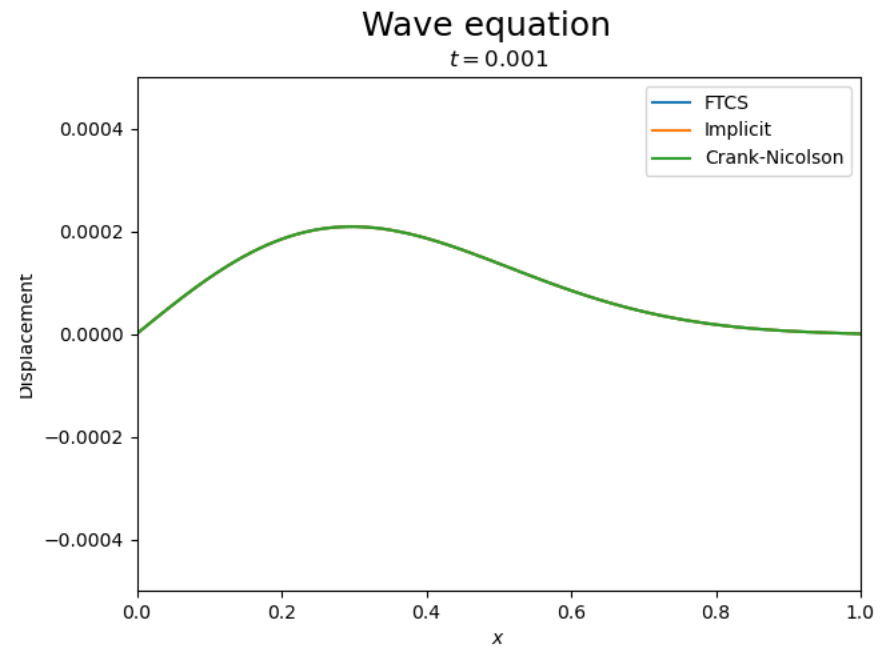
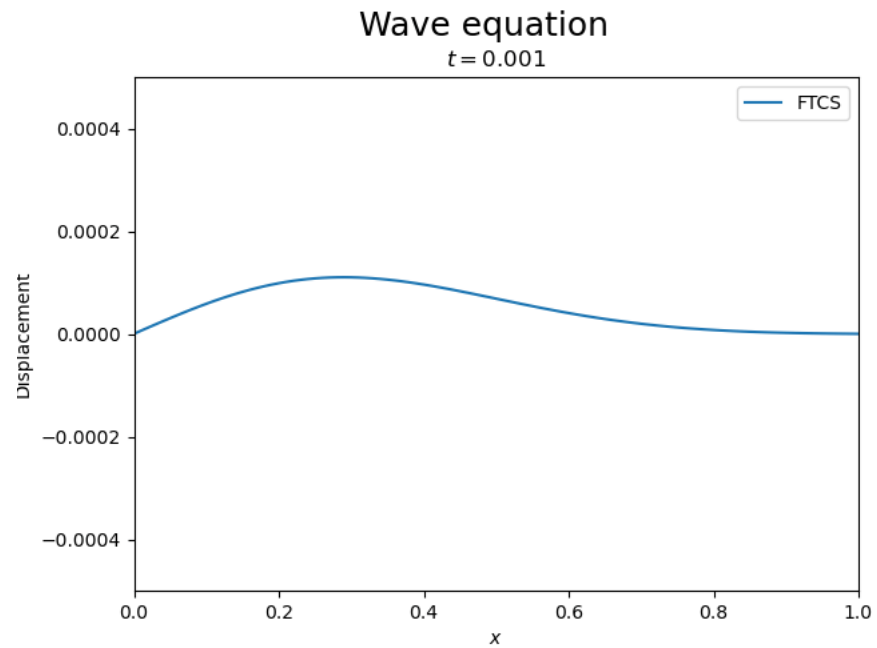
# Wave equation: Pulses

---



# Wave equation: Pulses

---



# Summary

---

Scheme	Complexity	Stability	Accuracy
FTCS	Explicit	Heat eq.: Conditionally stable Wave Eq.: Unstable	Exponential explosion
Implicit	Tridiagonal SLE	Stable	Exponential decay
Crank-Nicholson	Tridiagonal SLE	Stable	Conserves wave amplitude