# Scientific Computing: Partial Differential Equations

**Aleksandar Donev**
*Courant Institute, NYU*[1]
*donev@courant.nyu.edu*

[1]Course MATH-GA.2043 or CSCI-GA.2112, Fall 2020

Dec 3rd, 2020

# Outline

# Outline

# Partial Differential Equations

- **Partial differential equations** (PDEs) are differential equations that involve more then one independent variables.
- PDEs appear prominently in the modeling of **physical systems**, and so we will assume the independent variables are **time** $t$ and **spatial coordinate** $x$ in one dimension, or in higher dimensions $\mathbf{r}$, so our unknown is

$$u(x,t) \text{ or more generally } u(\mathbf{r},t)$$

- For **time-independent problems**, we will focus on one-dimensional or two-dimensional problems, $u(\mathbf{r}) \equiv u(x,y)$.
- Common short-hand notation for derivatives in PDE circles:

$$\frac{\partial u}{\partial t} = \partial_t u = u_t, \text{ and } \frac{\partial^2 u}{\partial x \partial y} = \partial_{xy} u = u_{xy}$$

- The **order of the PDE** is determined by the highest-order partial derivative appearing in the PDE. second-order PDE

# First Order Linear PDEs

- The simplest first-order linear PDE is the **advection equation**

Degenerate hyperbolic

$$u_t = -cu_x,$$

$u_t + c(x,t)\, u_x = f(x,t)$

$\to \cancel{+}\, g(u)$

where $c$ is a constant **speed of propagation**.

- If the domain of $x$-dependence is the whole real line, one needs an **initial condition** at time $t_0 = 0$,

$$u(0,x) = u_0(x) \text{ for } x \in \mathbb{R}.$$

- The solution of the equation can be constructed analytically for any initial condition:

$$u(x,t) = u_0(x - ct),$$

which means at time $t$ the solution is the same as at time $t_0$ but shifted by a distance $c(t - t_0)$.

- If $c > 0$ information propagates **upward**, and if $c < 0$ information propagates **downward**.

# Advection Equation

- Now consider the case when the domain of the PDE is a finite interval, $0 \leq x \leq 1$, with initial condition

$$u(0, x) = u_0(x) \text{ for } 0 \leq x \leq 1.$$

- If $c > 0$, then at a later time $t$ the solution would shift upward and we would not know what it is for $x < ct$.
- To specify the problem we thus also need **boundary conditions**, if $c > 0$ then

$$u(0, t) = u_L(t), \text{ where } u_L(t = 0) = u_0(x = 0),$$

or if $c < 0$ then

$$u(1, t) = u_R(t), \text{ where } u_R(t = 0) = u_0(x = 1).$$

# Second-Order PDEs

- Consider a second-order linear equation with constant coefficients:

$$\underbrace{au_{xx} + bu_{xy} + cu_{yy}} + du_x + eu_y + fu + g = 0.$$

- Depending on the values of the coefficients, this equation is classified as:
  - $b^2 > 4ac$: **hyperbolic**
  - $b^2 = 4ac$: **parabolic**
  - $b^2 < 4ac$: **elliptic**

most important for numerics

- The **type of the equation** makes a profound effect on how it is solved numerically.
- In real life, the coefficients depend on time or the spatial position instead of being constants, and one usually considers systems of SPDEs which may be of **mixed type**.

# PDE Classification

However, based on the most prominent character in the PDE, one still uses the classification loosely:

hardest

- **Hyperbolic** problems are **time-dependent** problems where there is no steady-state and **no dissipation** (diffusion), such as the **advection equation** or the **wave equation**: (also advection eq)

$$u_{tt} = c\, u_{xx}.$$

c – speed of wave

next simplest

- **Parabolic** problems are **time-dependent** problems evolving toward a steady-state because of **dissipation** (diffusion), such as the **heat equation**: diffusion equation

$$u_t = \mu u_{xx}, \quad \text{where } \mu > 0 \text{ is heat conductivity.}$$

simplest

- **Elliptic** problems are **time-independent** problems that describe the steady-state reached by parabolic PDEs, such as the **Laplace equation**: electrostatics or elasticity or flow

Poisson eq

$$\cancel{u_t =}\ u_{xx} + u_{yy} = \cancel{0}\, f(x,y).$$

# Heat Equation

- In one spatial dimension,

$$u_t = \mu u_{xx}, \text{ for } 0 \le x \le 1,$$

with **initial conditions**

$$u(0,x) = f(x) \text{ for } 0 \le x \le 1$$

- We also need one boundary condition for each of the end-points of the interval (in higher dimensions for each point along the boundary), e.g., **Dirichlet boundary conditions**

$$u(t,0) = u_L(t), \text{ and } u(t,1) = u_R(t)$$

or **Neumann boundary conditions**

$$\frac{\partial u}{\partial x}(t,0) = 0, \text{ and } \frac{\partial u}{\partial x}(t,1) = 0.$$

- The heat equation describes, for example, the temperature along the length of a rod where the ends are being held at specified temperatures (Dirichlet) or are insulated (Neumann).

# Outline

# Spatio-Temporal Discretization

- The first step in solving a PDE is **spatio-temporal discretization** of the solution, that is, representing the infinite-dimensional object $u(x,t)$ as a discrete collection $\mathbf{U}$ of values (a vector, matrix, or array) representing the solution over the spatial domain over some period of time $0 \leq t \leq T$.
- From the discrete solution $\mathbf{U}$, one should be able to obtain an approximation of $u(x,t)$ at any desired point in space and time inside the proper domain, for example, using interpolation.
- As a simple example, one could represent the solution on a **discrete spatio-temporal grid**,

time pointwise

$$U_i^{(k)} \approx u(i\Delta x,\ k\Delta t), \text{ for } i = 0, 1, \ldots, N, \text{ and } k = 0, 1, \ldots$$

space

- The same concepts of and relations between **consistency, stability and convergence** as for ODEs apply for (linear) PDEs.

# Spatial Discretization

$$u_t = f(u, u_x, u_{xx})$$

- Often, we construct the spatial discretization separately from the temporal one.

  or method of lines

  This **semidiscrete method** converts a PDE into **system of $N$ ODEs**

  second-order in space

$$\partial_t \mathbf{U}(t) = \mathbf{F}[\mathbf{U}(t), t],$$

  where $\mathbf{U}(t) \in \mathbb{R}^N$ is a **discrete approximation** to $u(x, t)$.
- $\mathbf{F}[\mathbf{U}(t), t]$ is a **consistent discretization** of $f(u, u_x, u_{xx})$, with error $O(h^2) = O(N^{-2})$ measured in an *appropriate* norm, where the **grid spacing** $h = \Delta x$ is a measure of the granularity of the spatial discretization.
- Then time is discretized, as for ODEs, with either a fixed or a variable **time step** $\Delta t$.

# Finite-Something Method

- Depending on how space is discretized, we distinguish the following classes of methods:

parabolic

  - **Finite-difference methods**, where the solution is represented **pointwise** on a discrete set of **nodes**, e.g., a regular grid:

$$U_i(t) \approx u(i\Delta x, t), \text{ for } i = 0, 1, \ldots, N$$

$U \equiv a$

elliptic

  - **Finite-element methods**, where the solution is directly represented through the interpolant, that is, $\mathbf{U}(t)$ actually stores the coefficients of the interpolating function $\tilde{u}(x;t)$, or more specifically, the coefficients of a discrete set of $N$ **basis functions**. Equations for the coefficients are obtained by integration of the PDE over the domain (**weak formulation**).

$$u(x,t) = \sum_{k=1}^{\infty} a_k(t)\, \varphi_k(x)$$

hyperbolic

  - **Finite-volume methods**, where the solution is represented by the **average values** over a set of cells (integral over the cell).

- If the solution is time-independent (steady-state, $u_t = 0$), then the problem simply becomes that of solving the system of $N$ equations,

$$\mathbf{F}(\mathbf{U}) = 0.$$

# Finite-Difference Method

- The idea behind finite-difference methods is simple: Use **finite-difference formulas** to approximate derivatives. For example, one can use the **second-order centered difference** (see Lectures 2 and 3 and Homework 1):

finite-difference

$$u_x(i\Delta x) \approx \frac{U_{i+1} - U_{i-1}}{2\Delta x} = u_x(i\Delta x) + O(\Delta x^2)$$

$$u_{xx}(i\Delta x) \approx \frac{U_{i+1} - 2U_i + U_{i-1}}{\Delta x^2}. \quad + O(\Delta x^2)$$

- For example, for the heat equation $u_t = \mu u_{xx}$ we get the system of ODEs:

method of lines

$$\partial_t U_i(t) = F_i(\mathbf{U}) = \mu \frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{\Delta x^2},$$

$\frac{d}{dt}$

where at the boundary points we use the boundary conditions, for example, for Dirichlet BCs we fix $u(0,t) = u_L(t)$, $u(1,t) = u_R(t)$

$$U_0(t) = u_L(t), \quad U_{N+1}(t) = u_R(t).$$

# Outline

# Poisson Equation

Elliptic PDE

- Finding the steady-state or equilibrium state of a system (the same as the limit $t \to \infty$ of the heat equation) is often modeled using the Poisson equation

$u(x,y)$

$$u_{xx} + u_{yy} = f(x,y) \text{ in a bounded } \textbf{domain } \Omega \subset \mathbb{R}^2$$

- To complete this equation we need **boundary conditions** but no initial conditions. A typical example is the **Dirichlet BC**

$$u\left(\partial\Omega\right) = 0,$$

where $\partial\Omega$ is the boundary of the domain $\Omega$. This is a model **elliptic PDE**.

- To illustrate things, let us consider a one-dimensional domain, $\Omega \equiv [a, b]$, and solve the **boundary-value problem**

$$u_{xx} = f(x) \quad \text{for } a < x < b$$

with the boundary condition

$$u(a) = 0, \text{ and } u(b) = 1$$

# Boundary Value Problems

$$u_{xx} = \cancel{0}\ f(x) \quad \text{for } a < x < b$$

- Observe that this is just a **second-order ODE**, and we have one initial condition $u(a) = 0$.
- What we are missing however is an initial condition for $u_x(a)$.
- One approach is to use a **shooting method**, which makes a guess for $u_x(a)$, then solves the ODE from $x = a$ to $x = b$, and sees if we get the correct value $u(b) = 1$.
- Denote with $u_b(z)$ the value $u(b)$ obtained by solving the ODE starting with initial guess $u_x(a) = z$.
- The shooting method basically requires solving the **nonlinear equation** for z

$$u_b(z) = 1,$$

which is not that easy.

# Finite-Difference BVP

- Instead, we can just use a finite-difference expression for the derivative to set

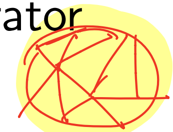
unknowns

$x_i = i\Delta x$

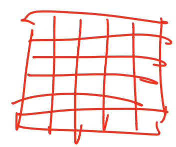
$$u_{xx}(i\Delta x) \approx \frac{U_{i+1} - 2U_i + U_{i-1}}{\Delta x^2} = f(x_i) \quad \text{for} \quad i = 1, \ldots, N-1$$

which together with $U_0 = u(a) = 0$ and $U_N = u(b) = 1$ gives us a linear system for $U_i$ with $N-1$ equations and $N-1$ unknowns.

- So we have converted the BVP into solving a **linear system of equations**, which we know how to solve. sparse, iterative
- The same works for the Poisson equation in two dimensions as wells, but we need to spend more time thinking about how to discretize the Laplacian operator



$$\nabla^2 u = u_{xx} + u_{yy}$$



on our domain of interest (finite differences for **regular grids**, finite elements for **irregular grids**).

# Finite-Element BVP

To discretize an **elliptic linear/nonlinear PDE** we first have to choose the following (all studied in this course!):

- A grid and a way to represent the function on the grid (related to interpolation, e.g. **piecewise polynomial interpolant**).
- A way to convert the PDE into a linear/nonlinear system of equations (strong or **weak form**),

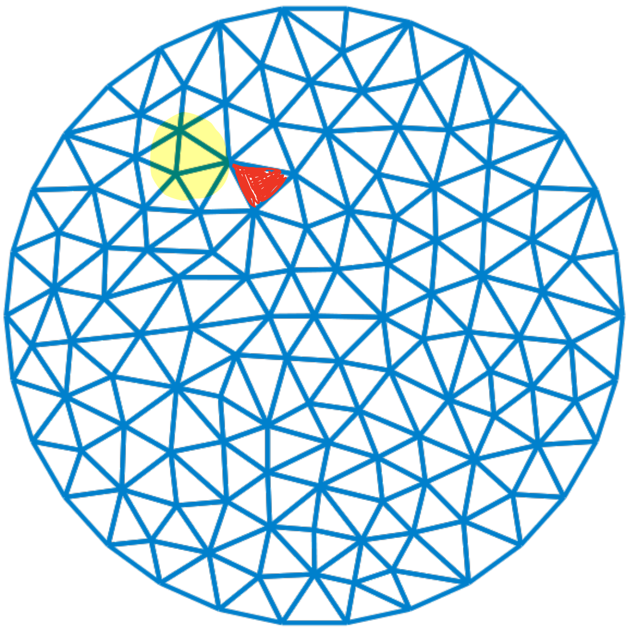
$$-\nabla^2 u = f \quad \text{on } \Omega, \quad u(\partial\Omega) = 0 \quad \Rightarrow$$

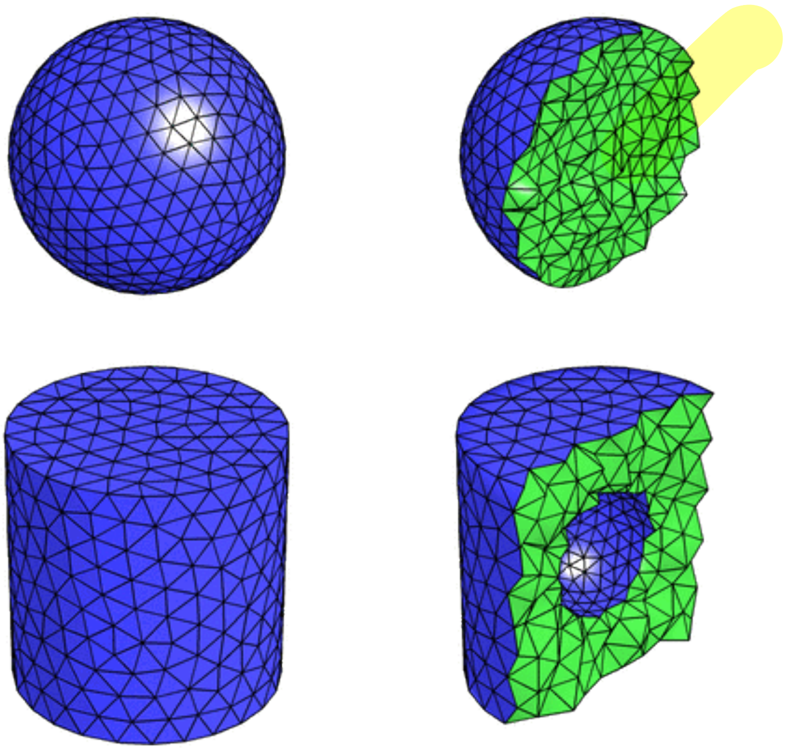
$$-\int_\Omega v\left(\nabla^2 u\right) dx = \int_\Omega fv\, dx = + \int_\Omega \nabla v \cdot \nabla u\, dx \quad \forall v(x)$$

- For weak (integral) form, a way to compute integrals (say **Gauss quadrature**)
- An efficient solver for the system of equations (**sparse iterative linear solvers**).

# Irregular (Simplicial) Meshes

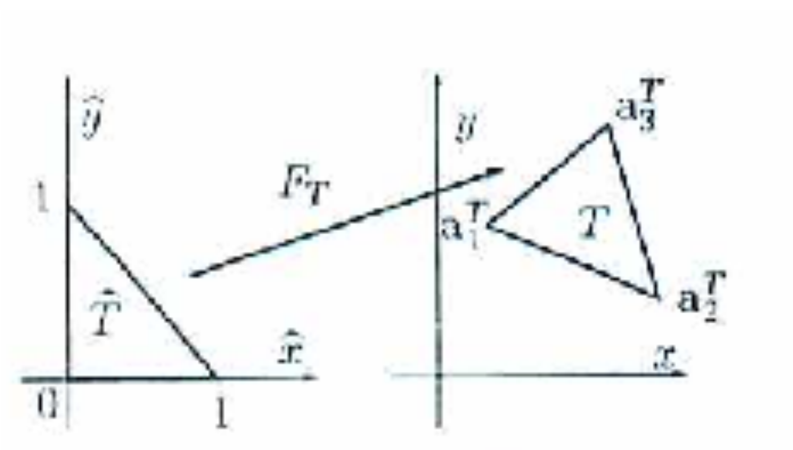
Any polygon can be triangulated into arbitrarily many **disjoint triangles**. Similarly **tetrahedral meshes** in 3D.





# Basis functions on triangles

- For irregular grids the $x$ and $y$ directions are no longer separable.
- But the idea of using basis functions $\phi_{i,j}$, a **reference triangle**, and **piecewise polynomial interpolants** still applies.
- For a piecewise constant function we need one coefficient per triangle, for a linear function we need 3 coefficients $(x, y, \text{const})$, for quadratic 6 $(x, y, x^2, y^2, xy, \text{const})$, so we choose the **reference nodes**:



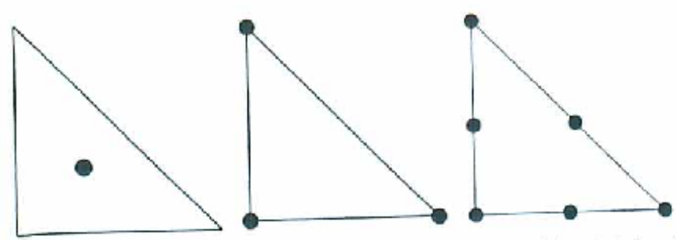
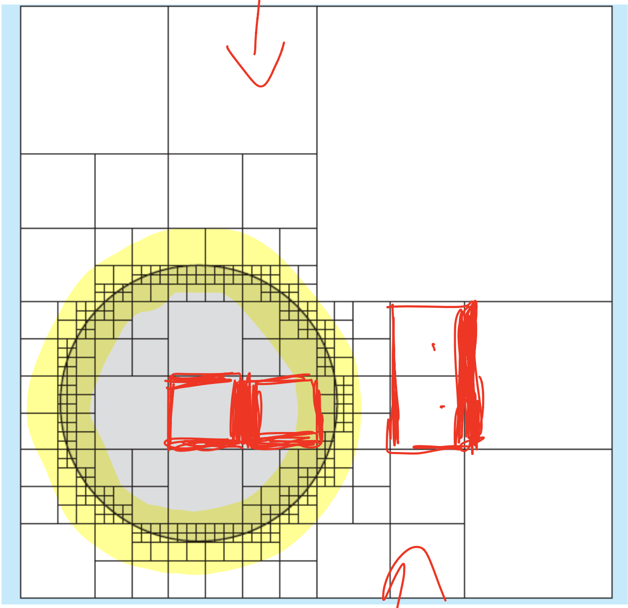



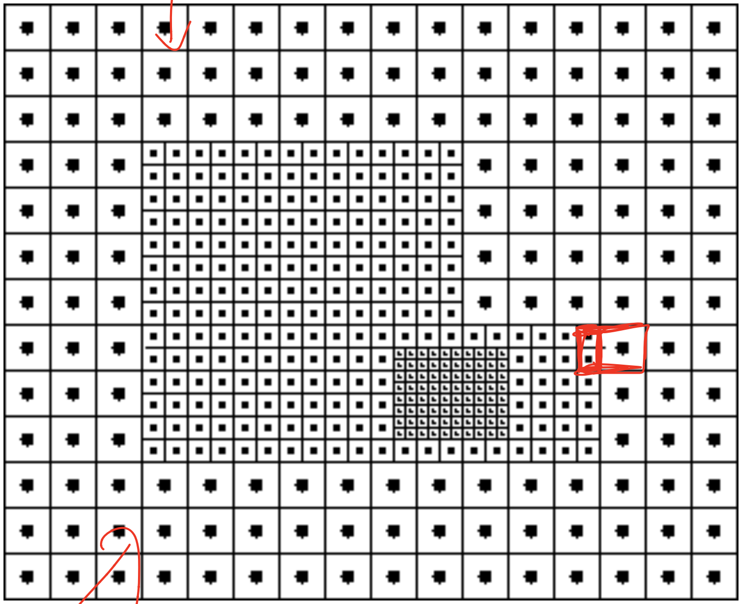
Fig. 8.8. Local interpolation nodes on $\hat{T}$ for $k = 0$ (*left*), $k = 1$ (*center*), $k = 2$ (*right*)

# Adaptive Meshes: Quadtrees and Block-Structured

finite-element
elliptic

finite-volume
hyperbolic



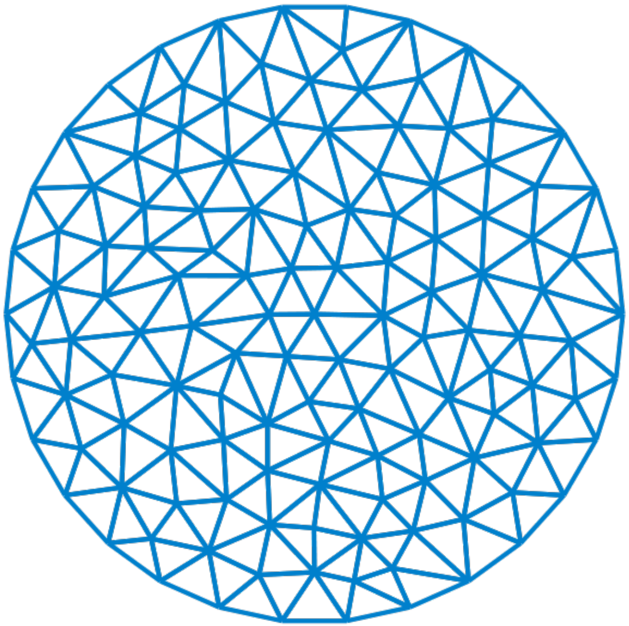


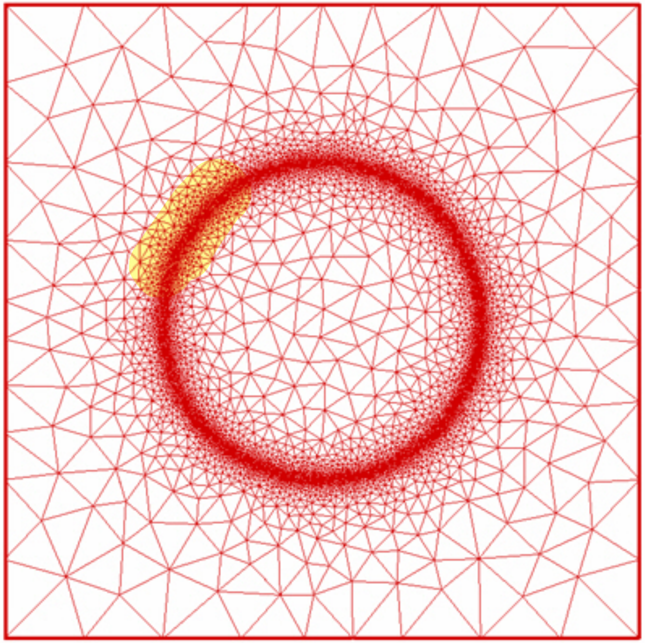
Quad Tree

Regular grid

# Irregular (Simplicial) Meshes

Any polygon can be triangulated into arbitrarily many **disjoint triangles**. Similarly **tetrahedral meshes** in 3D.





# Outline

# Temporal Integrators

$u_t = \mu u_{xx}$ linear ODEs

$$\partial_t U_i(t) = \mu \frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{\Delta x^2} \Rightarrow \boxed{\partial_t \mathbf{U} = \frac{\mu}{\Delta x^2} \mathbf{A} \mathbf{U}}$$

- Recall that the stiffness of this system of ODEs is measured by the eigenvalues of $\mu \mathbf{A} / \Delta x^2$.
  Here $\mathbf{A}$ is a tri-diagonal matrix with $-2$ on the diagonal, and 1 on the off-diagonal.
- In one spatial dimension, the non-zero eigenvalues are in the interval

$$\lambda_i \in \left[ -\frac{4\mu}{\Delta x^2}, -\frac{\pi^2 \mu}{(N \Delta x)^2} \right],$$

$$A = \begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -2 \end{bmatrix}$$

  which means that the ratio of the largest to the smallest eigenvalue (in magnitude) is $r \sim N^2$. stiff equation
- The system of ODEs becomes **very stiff** as the spatial discretization is refined (not good!). implicit method

# Explicit Scheme

- Consider using **forward Euler** method with a fixed time step $\Delta t$, and denote $U_i^{(k)} \approx U_i(k\Delta t)$:

impractical

$$U_i(t+\Delta t) = U_i^{(k+1)} = U_i^{(k)} + \frac{\mu \Delta t}{\Delta x^2}\left(U_{i+1}^{(k)} - 2U_i^{(k)} + U_{i-1}^{(k)}\right).$$

- Euler's method will be stable if

$$\Delta t < \frac{2}{\max_i |\mathrm{Re}(\lambda_i)|} = \frac{\Delta x^2}{2\mu},$$

which is a manifestation of the so-called **Courant-Friedrichs-Lewy (CFL) stability condition**

$$\frac{\mu \Delta t}{\Delta x^2} < \frac{1}{2}.$$

# Implicit Schemes

Parabolic

$$\partial_t \mathbf{U} = \frac{\mu}{\Delta x^2}\mathbf{A}\mathbf{U}$$

Implicit midpoint

- If one uses an **implicit method** such as ~~Crank-Nicolson~~ the time step can be increased, but a **linear system** must be solved at each time step:

No CFL condition

$$\frac{\mathbf{U}^{(k+1)} - \mathbf{U}^{(k)}}{\Delta t} = \frac{\mu}{\Delta x^2}\mathbf{A}\left[\frac{\mathbf{U}^{(k+1)} + \mathbf{U}^{(k)}}{2}\right].$$

practical

- For time-independent problems, e.g., elliptic PDEs, one may need to solve a non-linear system of equations but Newton's method will ultimately require solving a similar linear system!
- The linear systems that appear when solving PDEs have **large but sparse and structured matrices**. Often **preconditioned iterative methods** are used.

# Advection Equation

$u(x, t=0) = \varphi(x)$ Hyperbolic

$u(x,t) = \varphi(x-ct)$

$$u_t = -cu_x$$

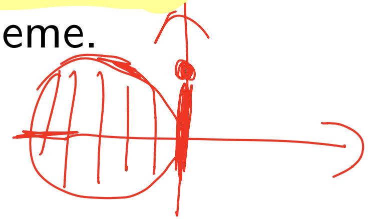
- Consider first a finite-difference explicit method that uses a centered difference approximation to $u_x$:

Euler's method

$$\frac{U_i^{(k+1)} - U_i^{(k)}}{\Delta t} = -c\,\frac{U_{i+1}^{(k)} - U_{i-1}^{(k)}}{2\Delta x} + O(\Delta x^2)$$

- While this seems reasonable, this scheme is **unconditionally unstable** for any time step $\Delta t$.
- If one uses at least a third-order Runge-Kutta scheme one can get a conditionally stable scheme.

$U_t = AU$



$$\begin{bmatrix} 0 & +1 & & \\ -1 & 0 & +1 & \\ & -1 & 0 & +1 \\ & & -1 & 0 \end{bmatrix}$$

# Upwinding

- Instead, we need to use the physics of the equation (direction of information propagation), to come up with a **upwind discretization** that uses one-sided derivatives: $O(\Delta x, \Delta t)$

$$\frac{U_i^{(k+1)} - U_i^{(k)}}{\Delta t} = \begin{cases} -c\,\frac{U_i^{(k)} - U_{i-1}^{(k)}}{\Delta x} & \text{if } c > 0 \\ -c\,\frac{U_{i+1}^{(k)} - U_i^{(k)}}{\Delta x} & \text{if } c \leq 0 \end{cases} \quad + O(\Delta x)$$

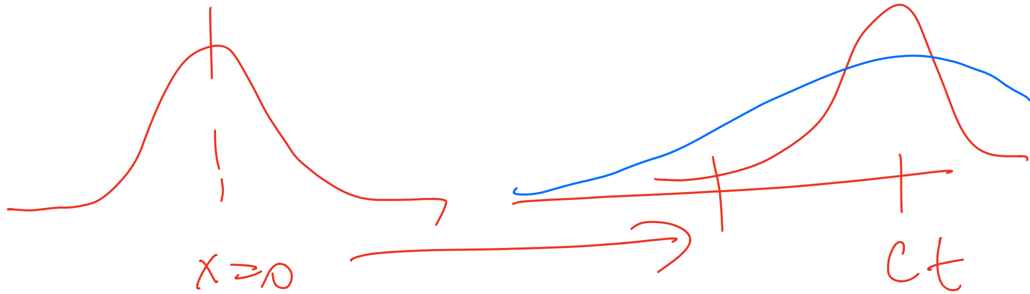
not used in practice

- The upwind method is stable if the **CFL stability condition** is satisfied:

$$\boxed{\Delta t < \frac{\Delta x}{|c|}}$$

- Constructing schemes that are stable and have good order of accuracy and are also efficient is often **an art form** and relies heavily on past experience and lessons learned over the years. There is little systematic guidance...

# Outline



# Conclusions/Summary

- The appropriate numerical method for solving a PDE depends heavily on its type: **hyperbolic** (advection, wave), **parabolic** (heat) or **elliptic** (Poisson or Laplace), or mixed, e.g., **advection/convection-diffusion** equation.
- The first step in solving a PDE is the construction of a spatial discretization of the solution: **finite-difference**, **finite-element** or **finite-volume**.
- This leads to a large system of ODEs that can in principle be solved with any of the methods we already discussed.
- Using an explicit method leads to a severe **CFL time-step restriction** due to increasing stiffness as the discretization is refined.
- One can use implicit methods but this requires **solving a large sparse linear system** at every time step.