Partial differential equations

Dr.ir. Ivo Roghair, Prof.dr.ir. Martin van Sint Annaland

Chemical Process Intensification group
Eindhoven University of Technology

Numerical Methods (6BER03), 2024-2025

Today's outline

Overview

Main question

How to solve parabolic PDEs like:

$$\frac{\partial c}{\partial t} = \mathcal{D}\frac{\partial^2 c}{\partial x^2} - u\frac{\partial c}{\partial x} + R$$

with

$$\begin{aligned} t = 0; 0 \leq x \leq \ell \quad & \Rightarrow c = c_0 \\ t > 0; x = 0 \quad & \Rightarrow -\mathcal{D}\frac{\partial c}{\partial x} + uc = u_{\text{in}}c_{\text{in}} \\ t > 0; x = \ell \quad & \Rightarrow \frac{\partial c}{\partial x} = 0 \end{aligned}$$

accurately and efficiently?

What is a PDE?

Partial differential equation

An equation containing a function and their derivatives to multiple independent variables.

Order of PDE

The highest derivative appearing in the PDE

General second order PDE:

$$A\frac{\partial^2 f}{\partial x^2}+B\frac{\partial^2 f}{\partial x\partial y}+C\frac{\partial^2 f}{\partial y^2}+D\frac{\partial f}{\partial x}+E\frac{\partial f}{\partial y}+Ff=G$$

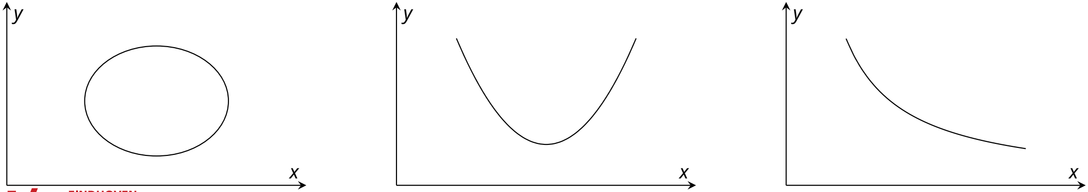
- Linear equation: Coefficients $A, B, \ldots, G$ do not depend on x and y.
- Non-linear equation: Coefficients $A, B, \ldots, G$ are a function of x and y.

Classification of PDE's

$$A\frac{\partial^2 f}{\partial x^2} + B\frac{\partial^2 f}{\partial x \partial y} + C\frac{\partial^2 f}{\partial y^2} + D\frac{\partial f}{\partial x} + E\frac{\partial f}{\partial y} + Ff = G$$

The discriminant Δ of a quadratic polynomial is computed as (note: only the higher order coefficients are important):
$\Delta = B^2 - 4AC$

- $\Delta < 0 \Rightarrow$ Elliptic equation
  (e.g. Laplace equation for stationary diffusion in 2D)
- $\Delta = 0 \Rightarrow$ Parabolic equation
  (e.g. instationary heat penetration in 1D)
- $\Delta > 0 \Rightarrow$ Hyperbolic equation
  (e.g. wave equation)

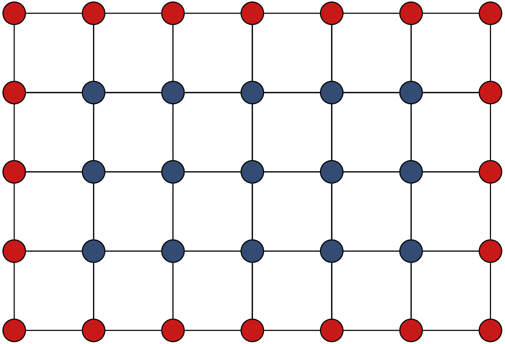


Importance of classification

Different PDE types require different solution techniques because of the difference in range of influence:

- *Characteristics*
  Curves in xy-domain along with signal propagation takes place
- *Domain of dependence of point P*
  points in xy-domain which influence the value of f in point P
- *Range of influence of point P*
  points in xy-domain which are influenced by the value of f in point P

Example elliptic PDE (boundary value problems: BVP)



Grid point at which dependent variable has to be computed

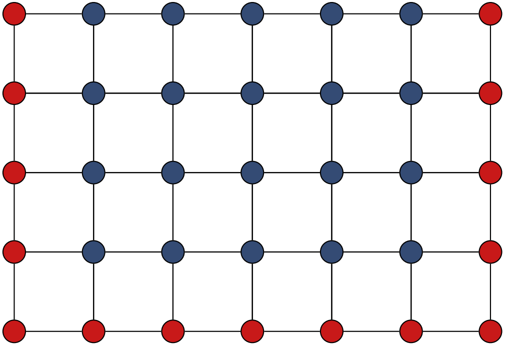
Grid point at which boundary condition is specified

Typical example: Poisson equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f(x,y)$$

Efficiency (memory requirements, CPU time) of the numerical method is of crucial importance.

Example parabolic PDE (initial value problem: IVP)



Grid point at which dependent variable has to be computed

Grid point at which boundary condition is specified

Typical example: Poisson equation

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = \mathcal{D}\frac{\partial^2 c}{\partial x^2} + R$$

Stability (in numerical sense) of the numerical method is of crucial importance.

Boundary conditions

- Dirichlet or fixed condition: prescribed value of f at boundary

  $f = f_0$ $\quad$ f_0 is a known function

- Neumann condition: prescribed value of derivative of f at boundary

  $\frac{\partial f}{\partial n} = q$ $\quad$ q is a known function

- Mixed or Robin condition: relation between f and $\frac{\partial f}{\partial n}$ at boundary

  $a\frac{\partial f}{\partial n} + bf = c$ $\quad$ a, b and c are known functions

Numerical solution method

Finite differences (method of lines, MOL):

1. Discretize spatial domain in discrete grid points
2. Find suitable approximation for the spatial derivatives
3. Substitute approximations in PDE, which gives a system of ODE's, one for every grid points
4. Advance in time with a suitable ODE solver

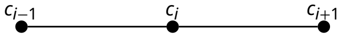
Alternative methods: collocation, Galerkin or Finite Element methods

Today's outline

Instationary diffusion equation (Fick's second law)

$$\frac{\partial c}{\partial t} = \mathcal{D}\frac{\partial^2 c}{\partial x^2}, \quad \text{with} \quad \begin{array}{l} t = 0; 0 \le x \le \ell \Rightarrow c = c_0 \\ t > 0; x = 0 \Rightarrow c = c_L \\ t > 0; x = \ell \Rightarrow c = c_R \end{array}$$

Second derivative $\dfrac{\partial^2 c}{\partial x^2}$



$$c_{i+1} = c_i + \left.\frac{\partial c}{\partial x}\right|_i \Delta x + \frac{1}{2}\left.\frac{\partial^2 c}{\partial x^2}\right|_i \Delta x^2 + \frac{1}{6}\left.\frac{\partial^3 c}{\partial x^3}\right|_i \Delta x^3 + \ldots$$

$$c_{i-1} = c_i - \left.\frac{\partial c}{\partial x}\right|_i \Delta x + \frac{1}{2}\left.\frac{\partial^2 c}{\partial x^2}\right|_i \Delta x^2 - \frac{1}{6}\left.\frac{\partial^3 c}{\partial x^3}\right|_i \Delta x^3 + \ldots$$

$$c_{i+1} + c_{i-1} = 2c_i + \left.\frac{\partial^2 c}{\partial x^2}\right|_i \Delta x^2 + \mathcal{O}(\Delta x^4)$$

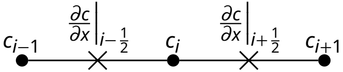
$$\Rightarrow \left.\frac{\partial^2 c}{\partial x^2}\right|_i = \frac{c_{i+1} - 2c_i + c_{i-1}}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

Due to symmetric discretization: second order (central discretization).

Instationary diffusion equation (Fick's second law)

An alternative discretization:

$$\left.\frac{\partial^2 c}{\partial x^2}\right|_i = \frac{\left.\frac{\partial c}{\partial x}\right|_{i+\frac{1}{2}} - \left.\frac{\partial c}{\partial x}\right|_{i-\frac{1}{2}}}{\Delta x} + \mathcal{O}(\Delta x^2)$$



$$= \frac{\frac{c_{i+1} - c_i}{\Delta x} - \frac{c_i - c_{i-1}}{\Delta x}}{\Delta x} = \frac{c_{i+1} - 2c_i + c_{i-1}}{\Delta x^2}$$

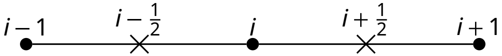
This is convenient for the derivation of $\frac{\partial}{\partial x}\left(\mathcal{D}\frac{\partial c}{\partial x}\right)$:

$$\frac{\partial}{\partial x}\left(\mathcal{D}\frac{\partial c}{\partial x}\right) = \frac{\mathcal{D}_{i+\frac{1}{2}}\left.\frac{\partial c}{\partial x}\right|_{i+\frac{1}{2}} - \mathcal{D}_{i-\frac{1}{2}}\left.\frac{\partial c}{\partial x}\right|_{i-\frac{1}{2}}}{\Delta x} = \frac{\mathcal{D}_{i+\frac{1}{2}}\frac{c_{i+1} - c_i}{\Delta x} - \mathcal{D}_{i-\frac{1}{2}}\frac{c_i - c_{i-1}}{\Delta x}}{\Delta x}$$

$$= \frac{\mathcal{D}_{i+\frac{1}{2}}c_{i+1} - \left(\mathcal{D}_{i+\frac{1}{2}} + \mathcal{D}_{i-\frac{1}{2}}\right)c_i + \mathcal{D}_{i-\frac{1}{2}}c_{i-1}}{(\Delta x)^2}$$

Instationary diffusion equation (Fick's second law)

$$\frac{\partial^2 f}{\partial x^2}$$



$$f_{i+\frac{1}{2}} = f_i + \frac{1}{2}\Delta x \left.\frac{\partial f}{\partial x}\right|_i \Delta x + \frac{1}{2}\left(\frac{1}{2}\Delta x\right)^2 \left.\frac{\partial^2 f}{\partial x^2}\right|_i + \mathcal{O}(\Delta x^3)$$

$$f_{i-\frac{1}{2}} = f_i - \frac{1}{2}\Delta x \left.\frac{\partial f}{\partial x}\right|_i \Delta x + \frac{1}{2}\left(\frac{1}{2}\Delta x\right)^2 \left.\frac{\partial^2 f}{\partial x^2}\right|_i + \mathcal{O}(\Delta x^3)$$

$$f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}} = \Delta x \frac{\partial f}{\partial x} + \mathcal{O}(\Delta x^3)$$

$$\Rightarrow \left.\frac{\partial f}{\partial x}\right|_i = \frac{f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}}{\Delta x} + \mathcal{O}(\Delta x^2)$$

Symmetric discretization yields second order!

Instationary diffusion equation: spatial discretization

Substitution of spatial derivatives yields:

$$\frac{dc_i}{dt} = \mathcal{D}\frac{c_{i-1} - 2c_i + c_{i+1}}{\Delta x^2} \quad \text{for } i = 0, \ldots, N$$

For example, using 6 (ridiculously low number!) grid points:

c_0 c_1 c_2 c_3 c_4 c_5

$$c_0 = c_L$$

$$\frac{dc_1}{dt} = \mathcal{D}\frac{c_0 - 2c_1 + c_2}{\Delta x^2}$$

$$\frac{dc_2}{dt} = \mathcal{D}\frac{c_1 - 2c_2 + c_3}{\Delta x^2}$$

$$\frac{dc_3}{dt} = \mathcal{D}\frac{c_2 - 2c_3 + c_4}{\Delta x^2}$$

$$\frac{dc_4}{dt} = \mathcal{D}\frac{c_3 - 2c_4 + c_5}{\Delta x^2}$$

$$c_5 = c_R$$

Instationary diffusion equation: boundary conditions

Two options:

1. Keep boundary conditions as additional equations:

$$c_0 = c_L, \frac{dc_1}{dt} = \mathcal{D}\frac{c_0 - 2c_1 + c_2}{\Delta x^2}, \frac{dc_2}{dt} = \mathcal{D}\frac{c_1 - 2c_2 + c_3}{\Delta x^2},$$
$$\frac{dc_3}{dt} = \mathcal{D}\frac{c_2 - 2c_3 + c_4}{\Delta x^2}, \frac{dc_4}{dt} = \mathcal{D}\frac{c_3 - 2c_4 + c_5}{\Delta x^2}, c_5 = c_R$$

2. Substitute boundary conditions to reduce number of equations:

$$\frac{dc_1}{dt} = \mathcal{D}\frac{c_L - 2c_1 + c_2}{\Delta x^2}, \frac{dc_2}{dt} = \mathcal{D}\frac{c_1 - 2c_2 + c_3}{\Delta x^2},$$
$$\frac{dc_3}{dt} = \mathcal{D}\frac{c_2 - 2c_3 + c_4}{\Delta x^2}, \frac{dc_4}{dt} = \mathcal{D}\frac{c_3 - 2c_4 + c_R}{\Delta x^2}$$

Instationary diffusion equation: temporal discretization

$$\frac{dc_i}{dt} = \mathcal{D}\frac{c_{i-1} - 2c_i + c_{i+1}}{\Delta x^2}$$

Time discretization: forward Euler (explicit)

$$\frac{c_i^{n+1} - c_i^n}{\Delta t} = \mathcal{D}\frac{c_{i-1}^n - 2c_i^n + c_{i+1}^n}{\Delta x^2}$$

$$\Rightarrow c_i^{n+1} = \mathrm{Fo}c_{i-1}^n + (1 - 2\mathrm{Fo})c_i^n + \mathrm{Fo}c_{i+1}^n \quad \text{with } \mathrm{Fo} = \frac{\mathcal{D}\Delta t}{\Delta x^2}$$

Straightforward updating (explicit equation), simple to implement in a program but stability constraint $\mathrm{Fo} = \frac{\mathcal{D}\Delta t}{\Delta x^2} < \frac{1}{2}$!

Small $\Delta x \Rightarrow$ small $\Delta t \Rightarrow$ patience required ☹

Instationary diffusion equation: temporal discretization

$$\frac{dc_i}{dt} = \mathcal{D}\frac{c_{i-1} - 2c_i + c_{i+1}}{\Delta x^2}$$

Time discretization: backward Euler (implicit)

$$\frac{c_i^{n+1} - c_i^{n}}{\Delta t} = \mathcal{D}\frac{c_{i-1}^{n+1} - 2c_i^{n+1} + c_{i+1}^{n+1}}{\Delta x^2}$$

$$\Rightarrow -\mathrm{Fo}c_{i-1}^{n+1} + (1 + 2\mathrm{Fo})c_i^{n+1} - \mathrm{Fo}c_{i+1}^{n+1} = c_i^{n} \quad \text{with}\ \mathrm{Fo} = \frac{\mathcal{D}\Delta t}{\Delta x^2}$$

Requires the solution of a system of linear equations, but no stability constraints!

Note: extension to higher order schemes (with time step adaptation) straightforward. Often second or third order optimal, because for each Euler-like step in the additional order an often large system needs to be solved (not treated in this course).

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Solving the instationary diffusion equation: example

Solve the diffusion problem using explicit discretization:

$$\frac{\partial c_i}{\partial t} = \mathcal{D}\frac{\partial^2 c}{\partial x^2} \quad \text{with} \quad \begin{aligned} &0 \le x \le \delta,\ \delta = 5\cdot 10^{-3}\ \text{m} \\ &\delta/\Delta x = 100 \text{ grid cells} \\ &\mathcal{D} = 1\cdot 10^{-8}\ \text{m}^2\,\text{s}^{-1} \\ &t_{\text{end}} = 5000\ \text{s} \\ &c_{\text{L}} = 1\ \text{mol}\,\text{m}^{-3}\ c_{\text{R}} = 0\ \text{mol}\,\text{m}^{-3} \end{aligned}$$

$$c_i^{n+1} = \text{Fo}c_{i-1}^n + (1-2\text{Fo})c_i^n + \text{Fo}c_{i+1}^n \quad \text{with}\,\text{Fo} = \frac{\mathcal{D}\Delta t}{\Delta x^2}$$

1. Initialise variables
2. Compute time step so that $\text{Fo} \le \frac{1}{2} \Rightarrow \Delta t = 0.125\text{s}!$
3. Compute 40000 time steps times 100 grid nodes!
4. Store solution

Solving the instationary diffusion equation: example

Initialise the variables and matrices:

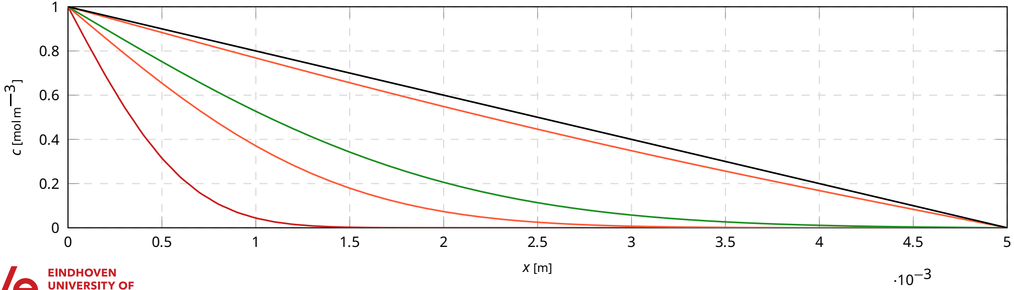
```python
import numpy as np

Nx = 100 # Nx grid points
Nt = 40000 # Nt time steps
D = 1e-8 # m/s
c_L = 1.0; c_R = 0 # mol/m3
t_end = 5000.0 # s
x_end = 5e-3 # m

# Time step and grid size
dt = t_end / Nt
dx = x_end / Nx

# Fourier number
Fo = D * dt / dx / dx

# Initial matrices for solutions (Nx times Nt)
c = np.zeros((Nt + 1, Nx + 1)) # All concentrations are zero
c[:, 0] = c_L # Concentration at the left side
c[:, Nx] = c_R # Concentration at the right side

# Grid node and time step positions
x = np.linspace(0, x_end, Nx + 1)
```

Solving the instationary diffusion equation: example

Compute the solution (nested time-and-grid loop):

- Create a time-loop
- Create a loop over *internal* grid points
- Update each node using $c_i^{n+1} = \text{Fo} c_{i-1}^n + (1 - 2\text{Fo}) c_i^n + \text{Fo} c_{i+1}^n$
- Plot the solution for selected time steps

Plotting the solution at $t = \{12.5, 62.5, 125, 625, 5000\}$ s.



Solving the instationary diffusion equation: example

A double-loop can impose serious computation times if the number of grid points increases:

```
for n in range(Nt - 1): # time loop
    for i in range(1, Nx): # Nested loop for grid nodes
        c[n+1, i] = Fo * c[n, i-1] + (1 - 2*Fo) * c[n, i] + Fo * c[n, i+1]
```

Remedy: vectorization. Construct a 3-point stencil Laplacian matrix first, then use the matrix product to evolve the simulation:

```
from scipy.sparse import diags

# Construct sparse matrix
e = np.ones(Nx-1)
md = np.concatenate(([1], (1 - 2 * Fo) * e, [1]))
ld = np.concatenate((Fo * e, [0]))
ud = np.concatenate(([0], Fo * e))
A = diags([ld, md, ud], offsets=[-1, 0, 1])

# Time evolution
for n in range(Nt - 1): # time loop
    c[n+1, :] = A.dot(c[n,:])
```

Solving the diffusion equation implicitly

Linear system $\boldsymbol{Ax} = \boldsymbol{b}$ from $-\mathrm{Fo}c_{i-1}^{n+1} + (1+2\mathrm{Fo})c_i^{n+1} - \mathrm{Fo}c_{i+1}^{n+1} = c_i^n$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ -\mathrm{Fo} & (1+2\mathrm{Fo}) & -\mathrm{Fo} & 0 & \cdots & 0 \\ 0 & -\mathrm{Fo} & (1+2\mathrm{Fo}) & -\mathrm{Fo} & \cdots & 0 \\ 0 & 0 & -\mathrm{Fo} & (1+2\mathrm{Fo}) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} c_0^{n+1} \\ c_1^{n+1} \\ c_2^{n+1} \\ c_3^{n+1} \\ \vdots \\ c_m^{n+1} \end{pmatrix} = \begin{pmatrix} c_0^n \\ c_1^n \\ c_2^n \\ c_3^n \\ \vdots \\ c_m^n \end{pmatrix}$$

$1 \times c_0^{n+1} = c_0^n$ (boundary condition)

$-\mathrm{Fo}c_0^{n+1} + (1+2\mathrm{Fo})c_1^{n+1} - \mathrm{Fo}c_2^{n+1} = c_1^n$

$-\mathrm{Fo}c_1^{n+1} + (1+2\mathrm{Fo})c_2^{n+1} - \mathrm{Fo}c_3^{n+1} = c_2^n$

$-\mathrm{Fo}c_2^{n+1} + (1+2\mathrm{Fo})c_3^{n+1} - \mathrm{Fo}c_4^{n+1} = c_3^n$

$1 \times c_m^{n+1} = c_m^n$ (boundary condition)

Solving the diffusion equation implicitly in Python

To solve the linear system, we need to define matrix A. It is clear that storing many zeros is not efficient in terms of memory. We use a *sparse matrix* format. Two alternative ways to set up the matrix:

Set individual elements of the matrix:

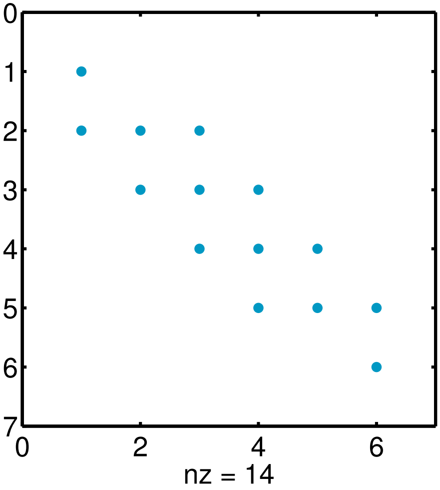
- Loop over the internal cells
- Set the coefficients in matrix `A` (main diagonal + elements left/right to it)
- Then set the coefficients for the boundary cells

Set matrix using bands:

- Consider the matrix structure (previous slide) and create vectors containing the values in each band
- Recall the `sp.sparse.diags` function to set entire bands to a sparse matrix

Solving the diffusion equation implicitly in Python

The command `plt.spy(A)` shows a figure with the non-zero positions.



Solving the diffusion equation implicitly in Python

The concentration matrix is initialised and the boundary conditions are set as follows:

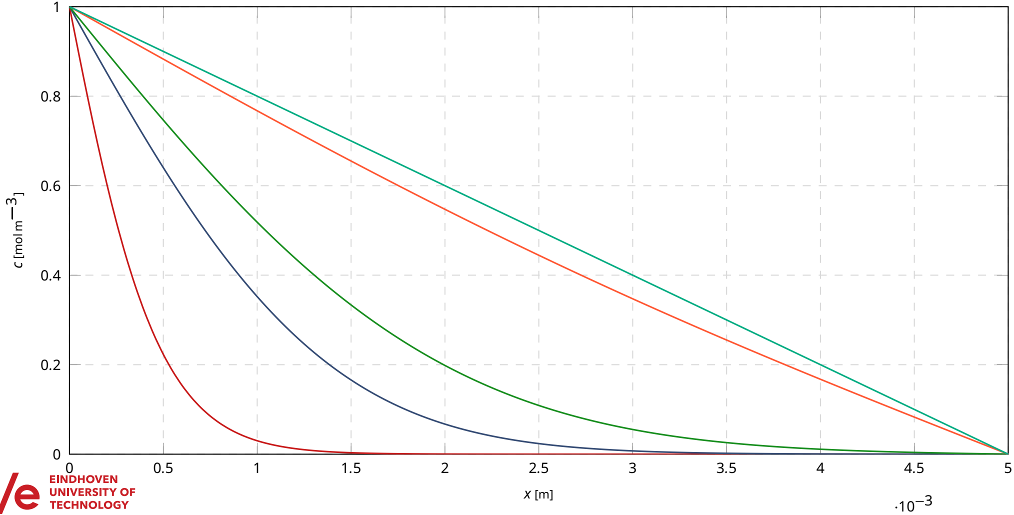
```python
# Initial matrices for solutions (Nx times Nt)
c = np.zeros((Nt+1, Nx+1)) # All concentrations are zero
c[:, 0] = c_L # Concentration at left side
c[:, Nx] = c_R # Concentration at right side
```

The right hand side vector (***b***) can now be set during the time-loop:

```python
from scipy.sparse.linalg import spsolve

for n in range(Nt-1): # time loop
   b = c[n, :] # Set right hand side
   solX = spsolve(A, b) # Solve linear system
   c[n+1, :] = solX # Store solution each time step
```

Solving the diffusion equation implicitly in Matlab

Plotting the solution at $t = \{12.5, 62.5, 125, 625, 5000\}$ s.



About explicit vs. implicit solutions

- Explicit solution:
    - Easy to implement
    - Very small time steps required.
    - This problem took about 0.5 s.
- Implicit solution:
    - Harder to implement, needs sparse matrix solver
    - No stability constraint
    - This problem took about 0.05 s
- The difference will become much larger for systems with e.g. more grid nodes!

Extension with non-linear source terms

$$\frac{\partial c}{\partial t} = \mathcal{D}\frac{\partial^2 c}{\partial x^2} + R(c) \quad \text{with} \quad \begin{array}{l} t = 0; 0 \leq x \leq \ell \Rightarrow c = c_0 \\ t > 0; x = 0 \Rightarrow c = c_L \\ t > 0; x = \ell \Rightarrow c = c_R \end{array}$$

- Forward Euler (explicit): simply add to right-hand side

$$\frac{c_i^{n+1} - c_i^n}{\Delta t} = \mathcal{D}\frac{c_{i-1}^n - 2c_i^n + c_{i+1}^n}{\Delta x^2} + R(c_i^n)$$

$$\Rightarrow c_i^{n+1} = \mathrm{Fo}c_{i-1}^n + (1 - 2\mathrm{Fo})c_i^n + \mathrm{Fo}c_{i+1}^n + R_i^n\Delta t$$

- Backward Euler (implicit): linearization required

$$R(c_i^{n+1}) = R(c_i^n) + \left.\frac{dR}{dc}\right|_i^n (c_i^{n+1} - c_i^n)$$

$$\frac{c_i^{n+1} - c_i^n}{\Delta t} = \mathcal{D}\frac{c_{i-1}^{n+1} - 2c_i^{n+1} + c_{i+1}^{n+1}}{\Delta x^2} + R(c_i^{n+1})$$

$$\Rightarrow -\mathrm{Fo}c_{i-1}^{n+1} + (1 + 2\mathrm{Fo} - \left.\frac{dR}{dc}\right|_i^n \Delta t)c_i^{n+1} - \mathrm{Fo}c_{i+1}^{n+1} = c_i^n + \left(R_i^n - \left.\frac{dR}{dc}\right|_i^n c_i^n\right)\Delta t$$

Today's outline

Extension with convection terms

$$\frac{\partial c}{\partial t} = \mathcal{D}\frac{\partial^2 c}{\partial x^2} - u\frac{\partial c}{\partial x} + R$$

Discretization of first derivative $\frac{dc}{dx}$,
looks simple but is numerical headache!

Central discretization:

$$\frac{dc}{dx} = \frac{c_{i+1} - c_{i-1}}{2\Delta x}$$

$\Rightarrow$ simple and easy, too bad it doesn't work: yields unstable solutions if convection dominated.

Central difference scheme of 1st derivative

Unsteady convection:

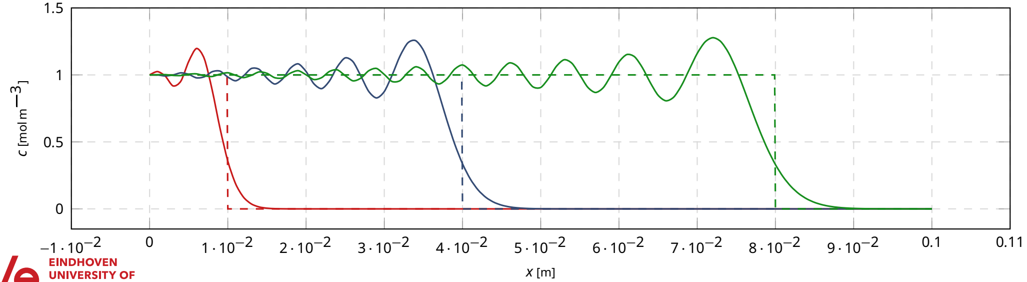
$$\frac{\partial c}{\partial t} = -u\frac{\partial c}{\partial x}$$

Central difference for first derivative:

$$\frac{dc}{dx} = \frac{c_{i+1} - c_{i-1}}{2\Delta x}$$

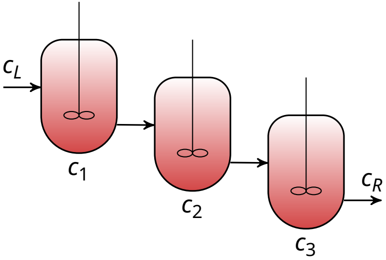
Forward Euler discretization of temporal and spatial domain:

$$\frac{c_i^{n+1} - c_i^n}{\Delta t} = -u\frac{c_{i+1} - c_{i-1}}{2\Delta x} \Rightarrow c_i^{n+1} = c_i^n - u\frac{c_{i+1}^n - c_{i-1}^n}{2\Delta x}\Delta t$$



Extension with convection terms

Solution: upwind discretization, like CSTR's in series:



First order upwind: $-u\left.\frac{dc}{dx}\right|_i = \begin{cases} -u\frac{c_i - c_{i-1}}{\Delta x} & \text{if } u \geq 0 \\ -u\frac{c_{i+1} - c_i}{\Delta x} & \text{if } u < 0 \end{cases}$ Stable if $\text{Co} = \frac{u\Delta t}{\Delta x} < 1$ (with Co the Courant number). However, only 1$^{\text{st}}$ order accurate (large smearing of concentration fronts). Higher order upwind requires TVD schemes (trick of the trade)...

First order upwind scheme of 1st derivative

Unsteady convection:

$$\frac{\partial c}{\partial t} = -u\frac{\partial c}{\partial x}$$

Upwind scheme for first derivative:

$$-u\left.\frac{dc}{dx}\right|_i = \begin{cases} -u\dfrac{c_i - c_{i-1}}{\Delta x} & \text{if } u \geq 0 \\[2ex] -u\dfrac{c_{i+1} - c_i}{\Delta x} & \text{if } u < 0 \end{cases}$$

Forward Euler discretization of temporal and spatial domain:

$$\frac{c_i^{n+1} - c_i^n}{\Delta t} = -u\frac{c_{i+1} - c_{i-1}}{2\Delta x}$$

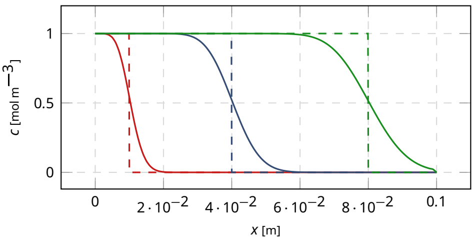
$$\Rightarrow c_i^{n+1} = \begin{cases} c_i^n - u\Delta t\dfrac{c_i - c_{i-1}}{\Delta x} & \text{if } u \geq 0 \\[2ex] c_i^n - u\Delta t\dfrac{c_{i+1} - c_i}{\Delta x} & \text{if } u < 0 \end{cases}$$

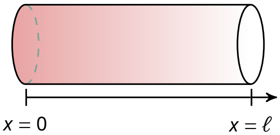
Upwind scheme: example

Unsteady convection through a pipe:

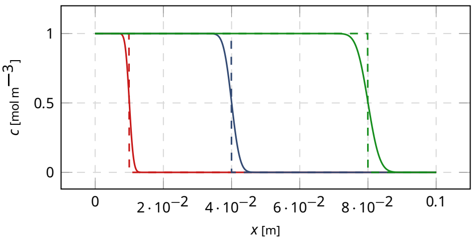
$$\frac{\partial c}{\partial t} = -u\frac{\partial c}{\partial x} \quad \text{with} \quad u = 0.1\,\mathrm{m\,s^{-1}} \Rightarrow c_i^{n+1} = c_i^n - u\frac{c_i - c_{i-1}}{\Delta x}\Delta t$$

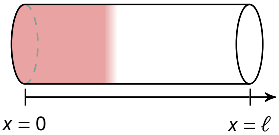
Using 100 grid cells





Using 1000 grid cells





Today's outline

Extension to systems of PDE's

- Explicit methods: straightforward extension
- Implicit methods: yields block-tridiagonal matrix (note ordering of equations: all variables per grid cell)

Extension to 2D or 3D systems

Spatial discretization in 2 directions — different methods available:

- Explicit
- Fully implicit
  - 1D gives tri-diagonal matrix
  - 2D gives penta-diagonal matrix
  - 3D gives hepta-diagonal matrix

  Use of dedicated matrix solvers (e.g. ICCG, multigrid, ...)
- Alternating direction implicit (ADI)
  - Per direction implicit, but still overall unconditionally stable

Further extensions for parabolic PDEs

- Higher order temporal discretization (multi-step) with time step adaptation
- Non-uniform grids with automatic grid adaptation
- Higher-order discretization methods, especially higher order TVD (flux delimited) schemes for convective fluxes (e.g. WENO schemes)
- Higher-order finite volume schemes (Riemann solvers)

Summary

- Several classes of PDEs were introduced
  - Elliptic, Parabolic, Hyperbolic PDEs
- Diffusion equation: discretization of temporal and spatial domain was discussed
  - Solutions of the diffusion equation using explicit and implicit methods
  - How to add non-linear source terms
- Convection: upwind vs. central difference schemes