

Computational Fluid Dynamics

Summer School on Fire Dynamics Modeling 2017

Lukas Arnold

Contents:

1. Computational Fluid Dynamics
2. Finite Difference Method
3. Scalar Transport

1. Computational Fluid Dynamics

1.1 Overview

1.2 Partial Differential Equations

1.3 Solution approaches

1.4 Finite Volume Method

1.5 Conclusions

1. Computational Fluid Dynamics

1.1 Overview

1.2 Partial Differential Equations

1.3 Solution approaches

1.4 Finite Volume Method

1.5 Conclusions

Lukas Arnold

Education:

Diploma (2005) and PhD (2008) in computational physics at the Ruhr-Universität Bochum, Germany

Employment:

Researcher at the Jülich Supercomputing Centre, Germany

- ▶ (2008 – 2009) Division “Application Support”
- ▶ (2009 – 2012) Division “Computational Science”
- ▶ (2012 – now) Division “Civil Safety and Traffic”

Lecturer at the Bergische Universität Wuppertal

- ▶ Master Civil Engineering
- ▶ Master Safety Engineering
- ▶ Master Computer Simulation in Science



Goals and contents

The goals of this lecture are to teach you the basics of:

- ▶ Numerical solution of partial differential equations, especially the Navier-Stokes equations, i.e. computational fluid dynamics (CFD)
- ▶ Discretisation techniques in space and time, especially the finite difference method, which is used by FDS
- ▶ Numerical schemes to solve the scalar transport equations

During the session, we will run a few Python scripts. The aim of those is to:

- ▶ 'Play' with parameters and methods – no programming skills are required
- ▶ Give you a starting point for further activities after the summer school

Optional tasks

The exercises contain optional tasks:

- ▶ If the default tasks are trivial for you, give it a try, or
- ▶ Address them during the week or after the school

In any case, just contact me if you need to discuss them.

1. Computational Fluid Dynamics

1.1 Overview

1.2 Partial Differential Equations

1.3 Solution approaches

1.4 Finite Volume Method

1.5 Conclusions

Nomenclature of differential operators

$$\phi = \phi(x, y, z, t)$$

$$\frac{\partial \phi(x, y, z, t)}{\partial t} = \partial_t \phi = \phi_t$$

$$\nabla = \begin{pmatrix} \partial_x \\ \partial_y \\ \partial_z \end{pmatrix} \quad \nabla^2 = \Delta$$

$$\nabla \phi = \begin{pmatrix} \partial_x \phi \\ \partial_y \phi \\ \partial_z \phi \end{pmatrix} \quad \nabla \cdot \vec{v} = \partial_x v_x + \partial_y v_y + \partial_z v_z \quad \nabla \times \vec{v} = \begin{pmatrix} \partial_y v_z - \partial_z v_y \\ \partial_z v_x - \partial_x v_z \\ \partial_x v_y - \partial_y v_x \end{pmatrix}$$

A word on PDEs

Partial differential equations (PDE) are the fundamental way to mathematically describe a huge range of processes. Many known formulas are special solutions of PDEs.

Examples for other major fields using PDEs:

- ▶ Electromagnetism, propagation of light
- ▶ Gravity, general relativity
- ▶ Quantum mechanics

In general it is not possible to solve PDE analytically and therefore approximation schemes are needed. In case of the Navier-Stokes equations, the schemes are referred to collectively as computational fluid dynamics (CFD).

Classes of PDEs

There exist three fundamental classes of PDEs, each with different challenges in numerical approximations.

The classification is based on the general form of a second order PDE:

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F = 0$$

It is important to note, that the coefficients depend on the variables, e.g. $A = A(x, y)$ and may therefore lead to inhomogeneous types.

- ▶ $B^2 - AC < 0$: elliptic
- ▶ $B^2 - AC = 0$: parabolic
- ▶ $B^2 - AC > 0$: hyperbolic

Classes of PDEs – Parabolic equations

A simple example is the heat equation:

$$\phi_t = k\phi_{xx}$$

These equations are often used to describe diffusion processes, where all disturbances are smoothed.

Classes of PDEs – Hyperbolic equations

The model equation here is the wave equation:

$$\phi_{tt} = c^2 \phi_{xx}$$

Solutions are 'wave-like', disturbances travel with finite propagation speed.

A conservative representation of hyperbolic equation systems is prescribed with a flux F :

$$\phi_t + \nabla \cdot F(\phi) = 0$$

Classes of PDEs – Elliptic equations

A simple elliptic equation is the Laplace equation:

$$\nabla^2 \phi = 0$$

This type of equations is often used to describe static processes, where all disturbances have already been relaxed.

For example, the steady state solution ($\partial_t \phi = 0$) of the heat equation is a Laplace equation.

Boundary conditions

The solution of PDE depends on initial and boundary conditions. In the case of elliptic equations, only boundary conditions are needed.

Two main kinds of fundamental boundary conditions are:

- ▶ Dirichlet: The solution at the boundary $\partial\Omega$ is prescribed or fixed in time, e.g.:

$$\phi(x, t) = \phi_0(x) \quad \text{at} \quad x = \partial\Omega$$

- ▶ Neumann: The derivative in the boundary normal direction is prescribed or constant in time:

$$\partial_n \phi(x, t) = f(x) \quad \text{at} \quad x = \partial\Omega$$

Examples:

- ▶ Adiabatic wall: $\partial_n T = 0$
- ▶ No-slip boundary: $v = 0$

1. Computational Fluid Dynamics

1.1 Overview

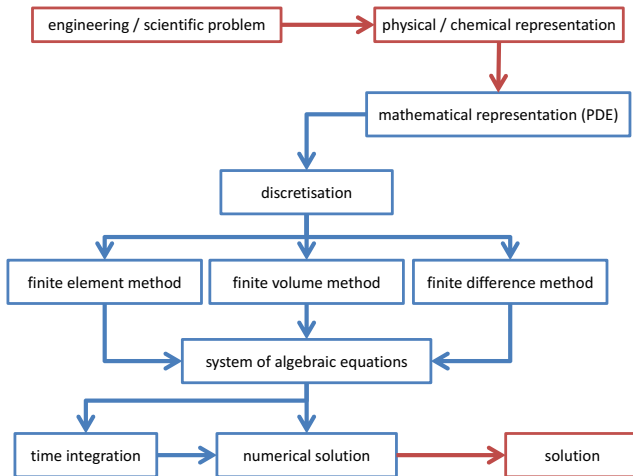
1.2 Partial Differential Equations

1.3 Solution approaches

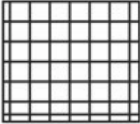
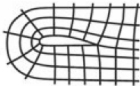

1.4 Finite Volume Method

1.5 Conclusions

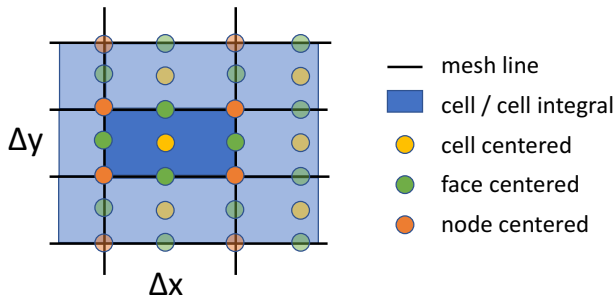
Modelling approach



Discretization methods

Method	Pros	Cons
 Finite difference	Fast evaluation Easy High order	Simple geometry No local mesh refinement
 Finite volume	Conservative Easy Complex geometry Local mesh refinement	Low order Slow evaluation
 Finite element	(Conservative) Complex geometry Local mesh refinement High order	Slow evaluation Complex scheme

Nodes and cells



- ▶ Subscripts for positioning: $\phi_i = \phi(i \cdot \Delta x)$
- ▶ Mesh spacing Δx , Δy and Δz may be inhomogeneous
- ▶ If the mesh lines are orthogonal, the mesh is called Cartesian (like in FDS)
- ▶ All above degrees of freedom (dof) may be used for discretization, i.e. numerical approximation

Exercise 1 – Discretization

Input file: 01_discretize_function.py

Goal: Visualize the discretization of a given analytical 1D function.

Tasks:

1. Execute the Python script and observe the discretization of the function

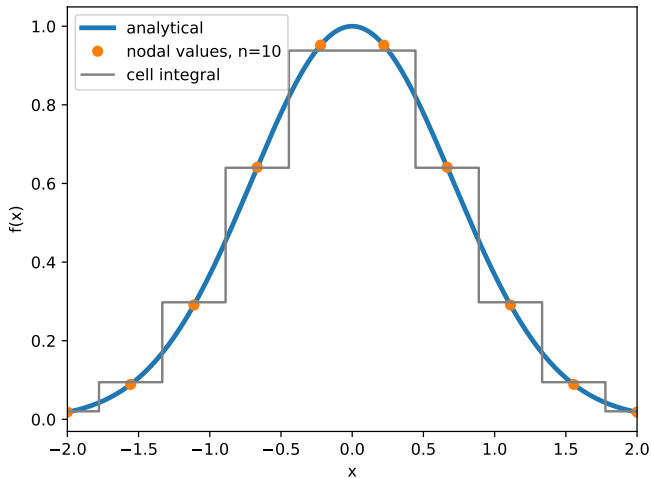
$$f(x) = e^{-x^2} \quad \text{with} \quad x \in [-2, 2]$$

2. Change the number of discretization points n for finer / coarser discretization.

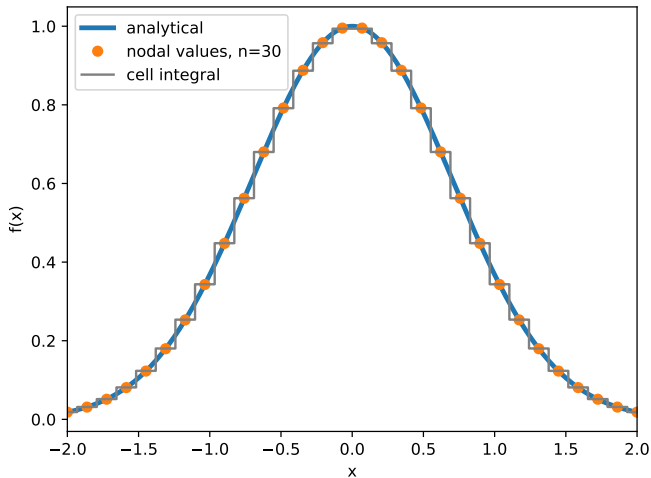
Optional:

3. Can you spot differences in the nodal vs. cell integral discretizations?
Where and why do they occur?
4. Change the analytical function.

Exercise 1 – Discretization (results)



Exercise 1 – Discretization (results)



1. Computational Fluid Dynamics

1.1 Overview

1.2 Partial Differential Equations

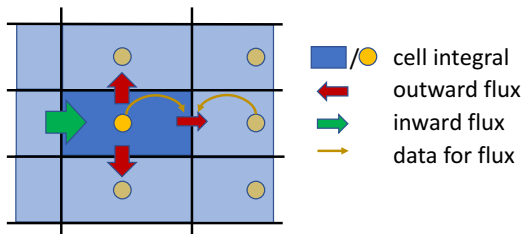
1.3 Solution approaches

1.4 Finite Volume Method

1.5 Conclusions

Basic idea

In the finite volume method (FVM) the cell integrals are considered. All value changes are due to cell boundary fluxes, therefore this is a natural way to describe conserved properties. Here, the fluxes for neighboring cells are equal, i.e. nothing gets lost; except at computational domain boundaries and with volumetric sources.



The total domain integral change is given by the flux through the computational domain boundaries (plus sources).

Weak formulation

The method is based on the weak formulation of hyperbolic PDEs:

$$\text{strong formulation} \quad \phi_t + \nabla \cdot F(\phi) = 0$$

$$\text{weak formulation} \quad \int_V (\phi_t + \nabla \cdot F(\phi)) \, dV = 0$$

Note: In general there is a weight function in the weak formulation. If the function is chosen to be the identity, the FVM arises, otherwise the finite element method (FEM) is formulated.

Example hyperbolic PDE

We will demonstrate the FVM on the following hyperbolic PDE:

$$\partial_t a + \nabla \cdot \vec{f}(a) = 0 \quad \left(\text{e.g. continuity equation with } a = \rho, \vec{f} = \rho \vec{v} \right)$$

The integral form is

$$\int_V \partial_t a + \nabla \cdot \vec{f} \, dV = \int_V \partial_t a \, dV + \int_V \nabla \cdot \vec{f} \, dV = 0$$

The discretisation is accomplished by dividing the computational domain V into non-overlapping subvolumes V_i . The same equations are true for the subdomains:

$$\int_{V_i} \partial_t a \, dV_i + \int_{V_i} \nabla \cdot \vec{f} \, dV_i = 0$$

Gauss theorem

Gauss's theorem states, that the integral of the divergence of any vector field \vec{u} in the volume V is equal to the boundary integral of the vector field on the volume's surface S :

$$\int_V \nabla \cdot \vec{u} \, dV = \int_S \vec{u} \cdot \vec{n} \, dS$$

With \vec{n} being the normal on the bounding surface S .

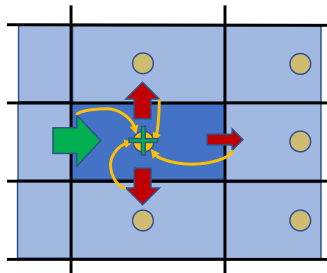
Boundary fluxes

Using Gauss's theorem the model equation for the cell integrals a_i become

$$\partial_t a_i + \int_{S_i} \vec{f} \cdot \vec{n} dS_i = 0 \quad \rightarrow \quad \partial_t a_i + \sum_j f_k n_k$$

This results in the summation of cell face values:

- ▶ The flux is a function of the solution variables, which are only integrals, approximation needed
- ▶ Determine fluxes for each cell, e.g. by averaging neighbour values
- ▶ Compute the sum of fluxes for each cell



Conservative formulation of the Navier-Stokes equations

Conservative representation of the compressible Navier-Stokes equations:

$$\partial_t \vec{s}_i + \int_{S_i} \mathbf{F} \cdot \vec{n} dS_i = 0$$

with

$$\vec{s} = \begin{pmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ \rho E \end{pmatrix} \quad \text{and} \quad \mathbf{F} = \begin{pmatrix} \rho \vec{v} \\ \rho \vec{v} v_x - p \vec{e}_x \\ \rho \vec{v} v_y - p \vec{e}_y \\ \rho \vec{v} v_z - p \vec{e}_z \\ \rho \vec{v} E + \rho \vec{v} p - \mu \vec{v} \nabla \vec{v} - k \nabla T \end{pmatrix}$$

plus the elliptic pressure equation.

1. Computational Fluid Dynamics

1.1 Overview

1.2 Partial Differential Equations

1.3 Solution approaches

1.4 Finite Volume Method

1.5 Conclusions

Conclusions

- ▶ Various types of PDEs with different challenges
- ▶ To be numerically solved, PDEs must be formulated as approximating algebraic equations, via discretization
- ▶ There exist different discretization approaches: e.g. FDM, FVM, FEM

2. Finite Difference Method

2.1 Introduction

2.2 Numerical Derivatives

2.3 Time Integration

2.4 Conclusions

2. Finite Difference Method

2.1 Introduction

2.2 Numerical Derivatives

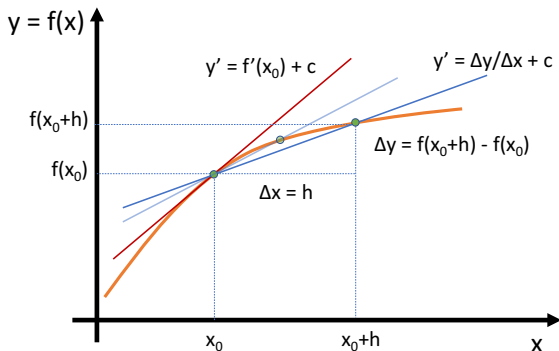
2.3 Time Integration

2.4 Conclusions

Basic idea

As derivatives are the major aspect of differential equations, a numerical approximation of those is needed.

The basic idea in the finite difference method (FDM) is to evaluate a function at certain locations and approximate its derivatives with this data.



Taylor expansion

The Taylor expansion may approximate any C^∞ function at an expansion point x_0 . The approximation is given in terms of h , being the vicinity around x_0 .

$$f(x_0 + h) = \sum_{i=0}^{\infty} \frac{1}{i!} f^{(i)}(x_0) \cdot h^i \quad (2.1)$$

$$= f(x_0) + f'(x_0) \cdot h + \frac{1}{2} f''(x_0) \cdot h^2 + \frac{1}{6} f'''(x_0) \cdot h^3 + \dots \quad (2.2)$$

In practice, the expansion is aborted at a given order. The expansion up to order three $\mathcal{O}(h^3)$ takes following form

$$f(x_0 + h) = f(x_0) + f'(x_0) \cdot h + \frac{1}{2} f''(x_0) \cdot h^2 + \mathcal{O}(h^3)$$

Notes:

- ▶ This approximation converges to the given function in the limit $h \rightarrow 0$.
- ▶ The rate at which the approximation converges in the above limit is called the method's order.

First derivative

To approximate the first derivative of a function $f(x)$ at $x = x_0$ the Taylor expansion may be used:

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \mathcal{O}(h^2)$$

This results in the first order approximation scheme.

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + \mathcal{O}(h)$$

Here, the exact value of the derivative is found with $h \rightarrow 0$.

2. Finite Difference Method

2.1 Introduction

2.2 Numerical Derivatives

2.3 Time Integration

2.4 Conclusions

First derivative

In the case of a discrete function, with $h = \Delta x$, the derivative is approximated by

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} + \mathcal{O}(\Delta x)$$

The above formula is called forward difference, while the backward scheme is of equal quality

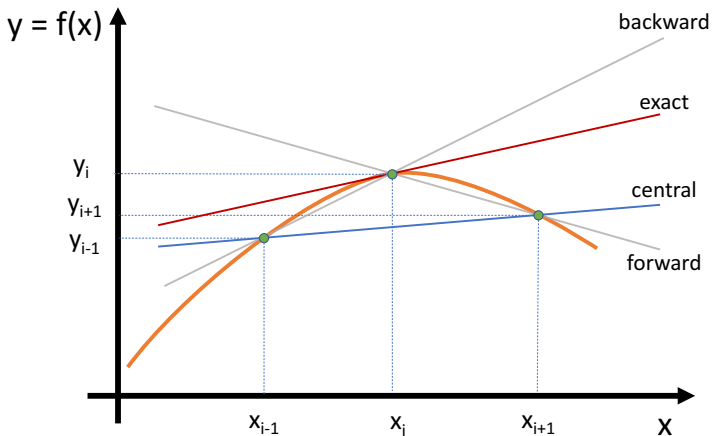
$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{\Delta x} + \mathcal{O}(\Delta x)$$

Non-symmetric schemes can be used at domain boundaries, where there exist no neighboring data in the boundary's direction.

The combination of the Taylor expansion at more points leads to higher order approximations, like the second order central difference scheme

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1}))}{2\Delta x} + \mathcal{O}(\Delta x^2)$$

First derivative – schematic



Second derivative

The same way as higher order schemes can be constructed, approximation schemes for higher derivatives can be formulated, like:

- ▶ Central scheme for second derivative

$$f''(x_i) = \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1}))}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

- ▶ Forward scheme for second derivative

$$f''(x_i) = \frac{2f(x_i) - 5f(x_{i+1}) + 4f(x_{i+2}) - f(x_{i+3}))}{\Delta x^2} + \mathcal{O}(\Delta x)$$

Note: $\Delta x^2 = (\Delta x)^2$

Error metrics

There exists a wide range of metrics to evaluate the difference of data sets or functions. Two major ones are the $L1$ and $L2$ norms. Given two sets a and b with n data points, like in a discrete function, they are defined as

$$\epsilon_i = a_i - b_i$$

$$L1 : \quad \|\epsilon\|_1 = \sum_{i=1}^n |\epsilon_i| \quad \text{and} \quad L2 : \quad \|\epsilon\|_2 = \sqrt{\sum_{i=1}^n \epsilon_i^2}$$

Based on those, the error metrics are formulated:

- ▶ Mean absolute error (MAE)

$$\epsilon_{MAE} = \frac{1}{n} \|\epsilon\|_1 = \frac{1}{n} \sum_{i=1}^n |\epsilon_i|$$

- ▶ Root mean square error (RMSE)

$$\epsilon_{RMSE} = \frac{1}{\sqrt{n}} \|\epsilon\|_2 = \sqrt{\frac{1}{n} \sum_{i=1}^n \epsilon_i^2}$$

Example 2 – Numerical derivative

Input file: 02_numerical_derivative.py

Goal: Compute and visualize the numerical approximation of the derivative of a given 1D function.

Tasks:

1. Execute the Python script to compute the numerical derivative of

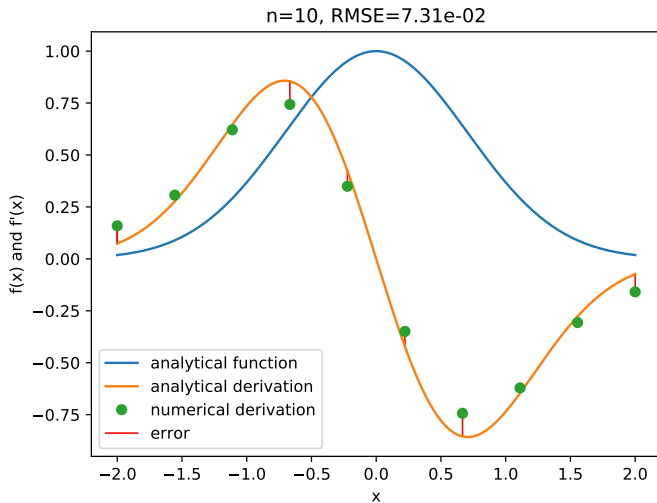
$$f(x) = e^{-x^2} \quad \text{with} \quad x \in [-2, 2]$$

2. Change the number of discretization points n to refine or coarsen the discretization. Note the change in the RMSE. Can you observe a pattern in the change of the RMSE?

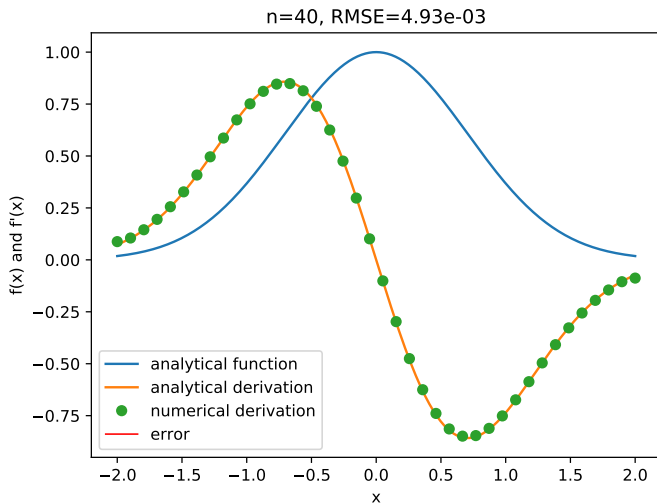
Optional:

3. Which numerical scheme is implemented? How are the boundary values computed?
4. Add the computation and analysis of the second derivative.

Example 2 – Numerical derivative – results



Example 2 – Numerical derivative – results



Example 2 – Numerical derivative – results

n	ϵ_{RMSE}	factor
10	7.31e-2	
20	1.84e-2	3.9
40	4.93e-3	3.73
80	1.41e-3	3.50

Notes:

- ▶ The error goes down as n rises, i.e. Δx gets smaller.
- ▶ Making Δx half size, reduces the error by a factor of about 4.

Convergence

In general, convergence describes the behavior of an approximation method to asymptotically represent the exact solution.

Convergence of discretization

- ▶ Numerical derivatives move towards exact derivatives as $h \rightarrow 0$
- ▶ Numerical solution of a PDE converges towards one solution; this allows to estimate approximation parameter
- ▶ Notes: a) LES does not converge towards DNS; b) simulations do not converge towards experimental data

Convergence of iterative methods

- ▶ Iterative solvers generally improve their solutions with each iteration
- ▶ Iterative solvers, e.g. linear systems, pressure coupling, are often part of an enclosing method or used for steady state solutions

Order of accuracy

The rate at which a method converges towards a solution is represented by its order.

A method is called n -th order, if the error ϵ is a function of the discretization h :

$$\epsilon = h^n$$

An often used nomenclature is the 'big O' notation: $\mathcal{O}(h^n)$ to indicate the order of accuracy.

Example 3 – Convergence of spatial discretization (I)

Input file: `03_derivative_convergence.py`

Goal: Compute and visualize the convergence of the derivative approximation (see example 02).

Tasks:

1. Execute the Python script and compare the error output and error plot.
2. Which order of accuracy do you observe? Does the error plot support your observation?
3. Change the scaling of the plot from linear to double logarithmic, i.e. replace the `plot` call with a `loglog` call. Which order can you now deduce from the plot?

Example 3 – Convergence of spatial discretization (II)

Input file: `03_derivative_convergence.py`

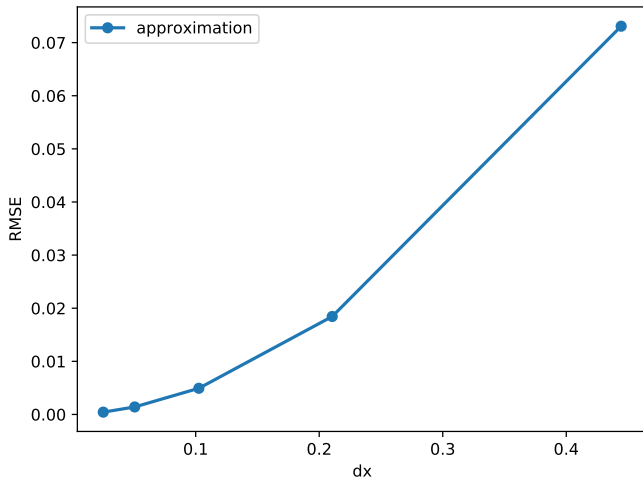
Optional:

4. Increase the number of refinements `n_refinement` to 15. Does the order change? If so, why?
5. How are the derivatives computed at the boundaries? Which order is implemented? Use the alternative which is commented out. Does this help?
6. Change the number of refinements `n_refinement` to 25. What do you observe?

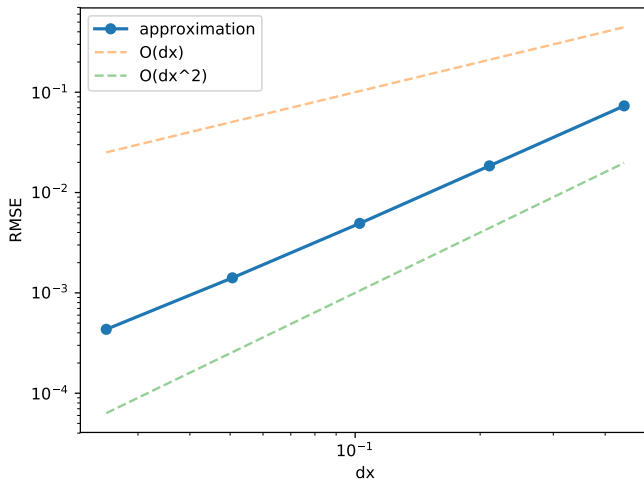
Example 3 – Convergence of spatial discretization – results

Δx	ϵ	factor	order
4.44e-01	7.31e-02		
2.11e-01	1.84e-02	3.96	1.99
1.03e-01	4.93e-03	3.74	1.90
5.06e-02	1.41e-03	3.49	1.80
2.52e-02	4.33e-04	3.26	1.70

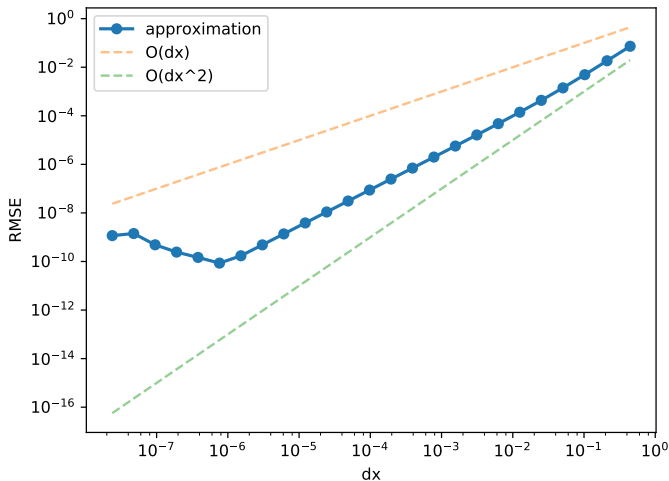
Example 3 – Convergence of spatial discretization – results



Example 3 – Convergence of spatial discretization – results



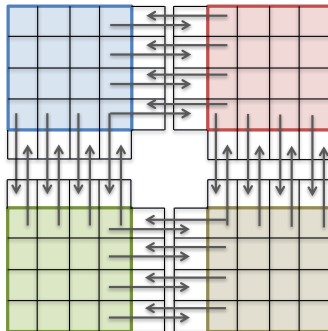
Example 3 – Convergence of spatial discretization – results



Ghost cells and domain decomposition

When the total computational domain is a set of meshes, like needed for parallel execution, the evaluation of derivatives at the mesh boundaries needs neighbor information.

- ▶ A common practice is to add additional layers of points, called ghost cells or halo.
- ▶ The exchange of the halo data is the main overhead in parallel processing.



2. Finite Difference Method

2.1 Introduction

2.2 Numerical Derivatives

2.3 Time Integration

2.4 Conclusions

Overview

For time dependent PDEs, a time integration, or time marching, scheme is needed. There exists a wide range of schemes with individual properties.

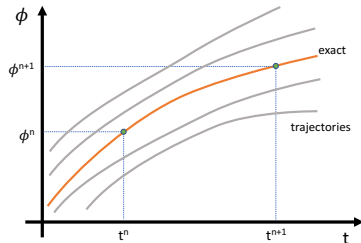
In general the temporal derivative is discretized, in the simplest case ($\phi = \phi(x, t)$) as:

$$\partial_t \phi = f(\phi) \quad \rightarrow \quad \frac{\phi^{n+1} - \phi^n}{\Delta t} = f(\phi)$$

with $\phi^n = \phi(x, t^n)$ and $t^n = n \cdot \Delta t$.

Notes:

- ▶ Δt does not have to be constant
- ▶ The point in time of the evaluation of f is crucial



Euler method

The most simple schemes are the forward and backward Euler methods:

$$\partial_t \phi = f(\phi)$$

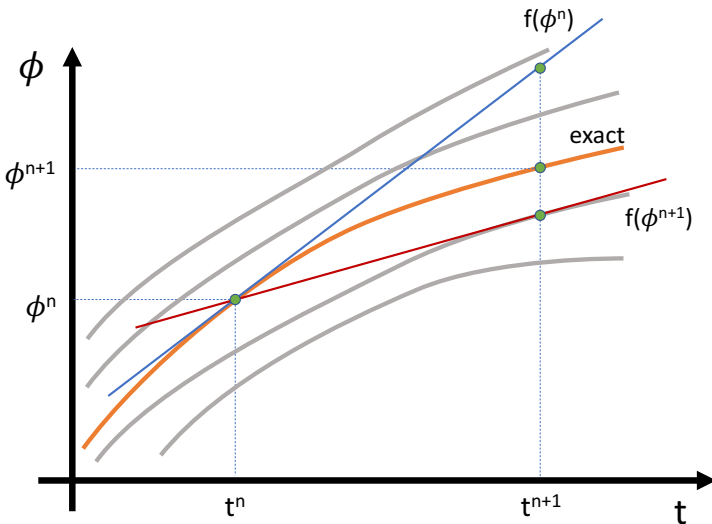
- ▶ Forward Euler

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = f(\phi^n) \quad \rightarrow \quad \phi^{n+1} = \phi^n + \Delta t \cdot f(\phi^n)$$

- ▶ Backward Euler

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = f(\phi^{n+1}) \quad \rightarrow \quad \phi^{n+1} = \phi^n + \Delta t \cdot f(\phi^{n+1})$$

Euler methods – schemes



Euler method – Explicit method – forward Euler

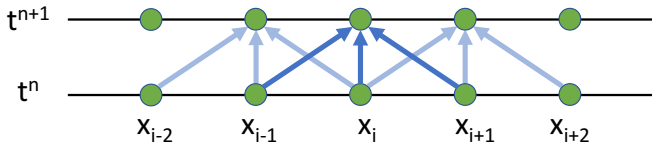
Consider a simple diffusion equation:

$$\partial_t \phi = \lambda \partial_{xx} \phi$$

- Forward Euler: This scheme can be directly – explicitly – evaluated.

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \lambda \frac{\phi_{i-1}^n - 2\phi_i^n + \phi_{i+1}^n}{\Delta x^2}$$

$$\phi_i^{n+1} = \phi_i^n + \Delta t \lambda \frac{\phi_{i-1}^n - 2\phi_i^n + \phi_{i+1}^n}{\Delta x^2}$$



Euler method – Implicit method – backward Euler

Consider a simple diffusion equation:

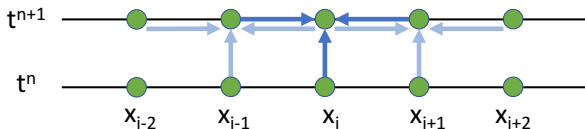
$$\partial_t \phi = \lambda \partial_{xx} \phi$$

- Backward Euler: Here a linear equation system must be solved for ϕ^{n+1} .

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \lambda \frac{\phi_{i-1}^{n+1} - 2\phi_i^{n+1} + \phi_{i+1}^{n+1}}{\Delta x^2}$$

$$\phi_i^{n+1} - \frac{\Delta t}{\Delta x^2} \lambda \left(\phi_{i-1}^{n+1} - 2\phi_i^{n+1} + \phi_{i+1}^{n+1} \right) = \phi_i^n$$

$$A\phi^{n+1} = \phi^n$$



Stability

An important property of a time integrator is its stability.

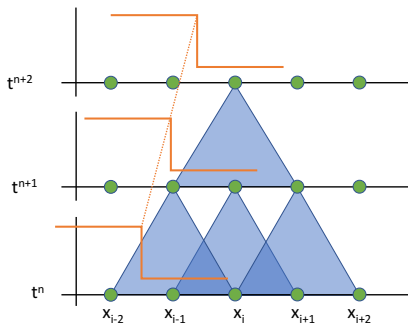
Given a PDE to be solved and a time integrator, a simple (linear) stability analysis can be conducted, e.g. von Neuman stability analysis.

The outcome is the growth factor, which indicates the growth rate of small disturbances. If this factor is larger than 1, then the scheme is unstable, as fluctuations will infinitely rise.

- ▶ Explicit schemes tend to be unstable or conditionally stable, i.e. if a condition for the time step is met
- ▶ Implicit schemes tend to be unconditionally stable, however they tend to damp the solution

Propagation speed (I)

A constant velocity advection problem demonstrates the stability condition.



- Model equation

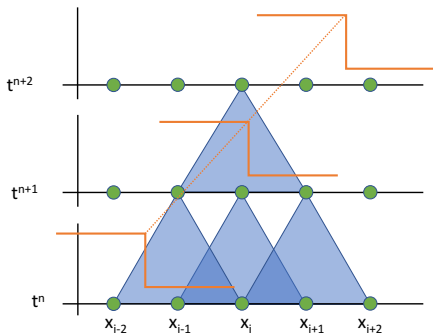
$$\partial_t \phi + \partial_x (v_0 \phi) = 0$$

- Small Δt
- Distance traveled per Δt :

$$\delta x = v_0 \Delta t < \Delta x$$

- The information moves less than Δx in a time step Δt and is captured by the neighbouring grid point
- No solution information is lost during time integration

Propagation speed (II)



- ▶ Large Δt
- ▶ Distance traveled per Δt :

$$\delta x = v_0 \Delta t > \Delta x$$

- ▶ The information moves more than Δx in a time step Δt and can therefore not be captured anymore
- ▶ Information is lost during time integration

Courant-Friedrichs-Lewy (CFL) condition

In general, there exist stability conditions for explicit schemes, which relates the maximal information travel speed v_{max} and the grid velocity $v_g = \Delta x / \Delta t$:

$$v_{max} < CFL \cdot v_g$$

Given a mesh resolution Δx and maximal velocities, the above condition limits the time step:

$$\Delta t < CFL \cdot \frac{\Delta x}{v_{max}}$$

Notes:

- ▶ The flow velocity may be computed in various ways (L_∞ , L_1 , L_2 norms), diffusion velocity $\propto 1/\Delta x$
- ▶ There also exist other constraints on Δt : mass density constraint and volume constraint.
- ▶ The value of the CFL number depends on the time integration scheme.

Implications of the CFL condition in FDS

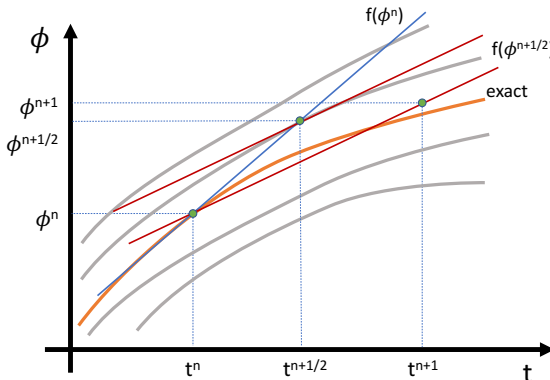
The condition for Δt used in FDS is

$$0.8 \leq CFL = \Delta t \left(\frac{\|\vec{v}\|}{\Delta x} + |\nabla \cdot \vec{u}| \right) \leq 1.0$$

- ▶ If the CFL number grows above the upper limit, the time step is set to 90% of the allowed, if it falls below the lower limit, then it is increased by 10%.
- ▶ In general, the time step is reduced by a factor of 2 when the mesh spacing is reduced by a factor of 2, i.e. the total computational effort increases by a factor of 16.

Predictor-corrector scheme

Predictor-corrector schemes use intermediate solutions (predictions) to correct the time integration.



In two step schemes, the intermediate solution is marked with a *, e.g. ϕ^* .

Overview of other methods

There exists a whole zoo of time integration methods. All with different stability properties, orders of accuracy and memory requirements.

- ▶ Θ -method: a combination of the forward and backward Euler schemes. The parameter Θ expresses the weighting, i.e. $\Theta = 0$ fully explicit forward Euler, $\Theta = 1$ fully implicit backward Euler, $\Theta = 0.5$ semi-implicit Crank-Nicolson.
- ▶ Runge-Kutta methods: family of explicit and implicit multistep methods
- ▶ Backward differentiation formulas (BDF): fully implicit methods based on previous solution steps

Example 4 – Wave equation (I)

In the case of isothermal compressible flows, with no convection, diffusion and source terms, the equations reduce to

$$\partial_t \rho = -\nabla \cdot (\rho \vec{v})$$

$$\partial_t \vec{v} = -\nabla p$$

A linear ansatz for the solution, compressibility and the ideal gas law, lead in 1D to the sound wave equation for density $\delta\rho$ and velocity δv fluctuations

$$\partial_t \delta\rho = c \partial_x \delta v$$

$$\partial_t \delta v = c \partial_x \delta\rho$$

with $c = \sqrt{\gamma \frac{RT}{m_M}}$ and the fundamental solutions:

$$\delta\rho(x, t) = \delta\rho_0 \sin(kx - \omega t) \quad \text{and} \quad \delta v(x, t) = \delta v_0 \sin(kx - \omega t)$$

Example 4 – Wave equation (II)

Input file: 04_wave.py

Goal: Solve the wave equation with different time integration schemes.

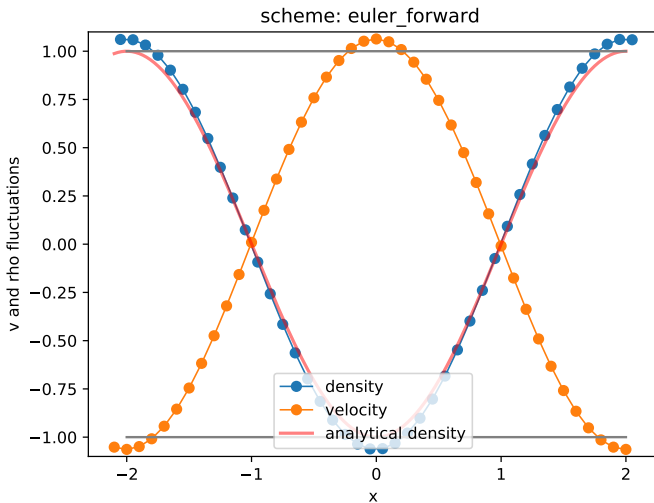
Tasks:

1. Execute the Python script and observe solution computed by the forward Euler scheme.
2. Change the time integration scheme to backward Euler and Crank-Nicolson. E.g. `scheme='euler_backward'`
3. Change the initial conditions to a Gauss peak: `initial='gauss'`.

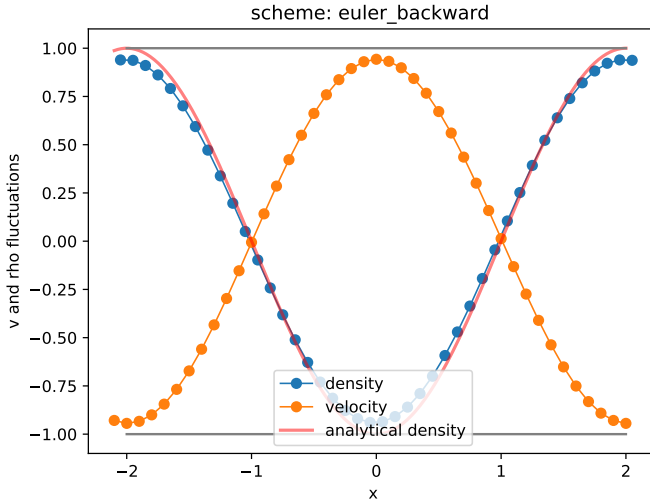
Optional:

4. Compare the implementations of the explicit and implicit solvers. Can you identify the steps needed to solve the linear system in the implicit Euler solver?
5. Are the spatial discretizations (e.g. in the explicit Euler) first or second order?
6. What does the 'leap frog' scheme do? `scheme='leap'`

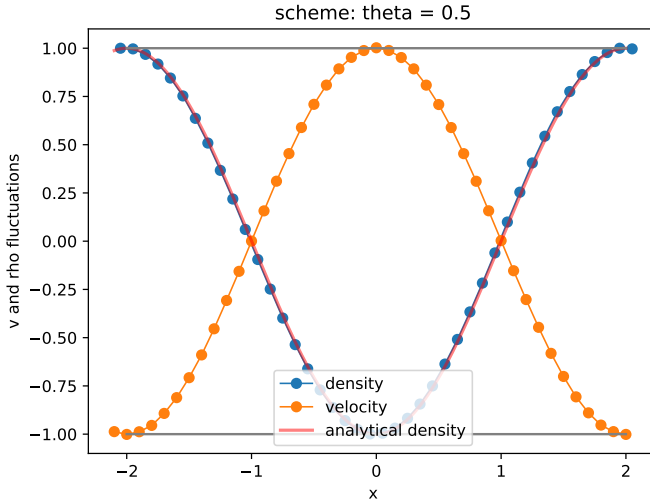
Example 4 – results – forward Euler



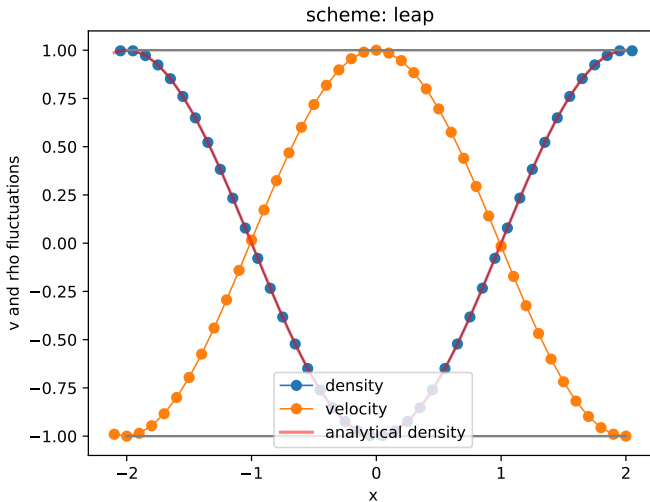
Example 4 – results – backward Euler



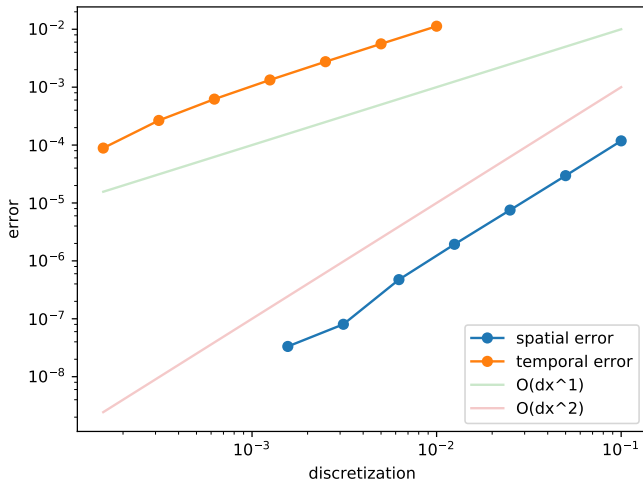
Example 4 – results – Crank-Nicolson



Example 4 – results – leap frog



Example 4 – results – convergence in space and time



2. Finite Difference Method

2.1 Introduction

2.2 Numerical Derivatives

2.3 Time Integration

2.4 Conclusions

Conclusions

- ▶ The finite difference method is a direct discretization method of differential equations.
- ▶ The accuracy of approximation depends on the chosen discretization size.
- ▶ The time integration may also follow an FDM approach, however, stability conditions apply.

3. Scalar Transport

3.1 Introduction

3.2 Flux Limiter

3.3 Conclusions

3. Scalar Transport

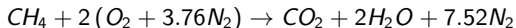
3.1 Introduction

3.2 Flux Limiter

3.3 Conclusions

Lumped species

FDS uses a lumped species approach, where related primitive species Y_α , e.g. nitrogen, oxygen, and carbon dioxide, are transported together. The classification into three lumped species Z_A (air), Z_F (fuel) and Z_P (products) leads in the case of methane combustion



to the following relation

$$\begin{pmatrix} 0.77 & 0.00 & 0.73 \\ 0.23 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.15 \\ 0.00 & 0.00 & 0.12 \end{pmatrix} \cdot \begin{pmatrix} Z_A \\ Z_F \\ Z_P \end{pmatrix} = \begin{pmatrix} Y_{N_2} \\ Y_{O_2} \\ Y_{CH_4} \\ Y_{CO_2} \\ Y_{H_2O} \end{pmatrix}$$

This approach allows to significantly reduce the cost of the computation of the transport process.

Transport equation

The transport of scalars is an advection-diffusion equation including source terms. In the case of the lumped species Z_α it takes the following form:

$$\partial_t(\rho Z_\alpha) + \nabla \cdot (\rho Z_\alpha \vec{u}) = \nabla \cdot (\rho D_\alpha \nabla Z_\alpha) + \dot{m}'''_\alpha + \dot{m}'''_{b,\alpha}$$

with the diffusion coefficient D_α and the mass sources \dot{m}'''_α .

The solution of this set of equations must satisfy the realizability condition, which is

$$Y_\alpha \geq 0 \quad \text{and} \quad \sum Y_\alpha = 1$$

Example 5 – Scalar transport – model equations

To demonstrate the challenge in solving the transport equation for a scalar field, the following simplified 1D advection equation is used:

$$\partial_t \phi + \partial_x (v_0 \phi) = 0$$

Here the advection velocity v_0 is constant and therefore the exact solution is given by shifting the initial conditions by $-vt$:

$$\phi(x, t) = \phi(x - vt, 0)$$

Example 5 – Scalar transport – exercise (I)

Input file: `05_advection.py`

Goal: Solve a simple 1D advection equation.

Tasks:

1. Execute the Python script and observe the transport of the initial field. What do you observe, that does not satisfy you?
2. Does a smaller time step (Δt) improve the solution?
3. Try out a stepwise initial function: `core='step'`.
4. Execute both cases with the so called upwind scheme: `scheme='upwind'`
5. What happens if your parameters do not satisfy the stability criterion for the upwind scheme?

$$\left| v_0 \frac{\Delta t}{\Delta x} \right| \leq 1$$

6. What solution do you achieve with the upwind scheme and $\Delta t = \Delta x / v_0$?

Example 5 – Scalar transport – exercise (II)

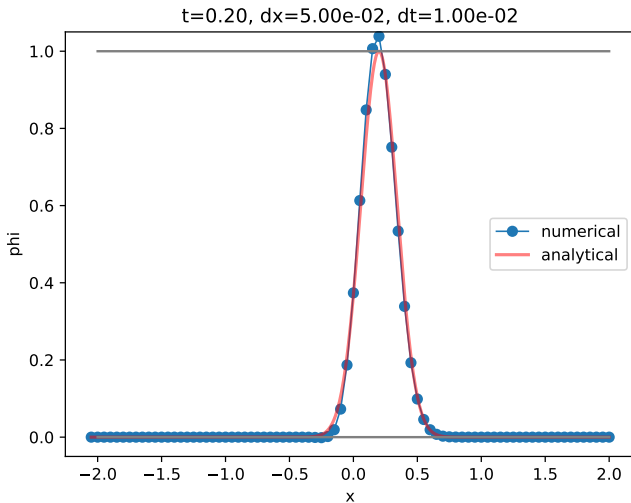
Input file: 05_advection.py

Goal: Solve a simple 1D advection equation.

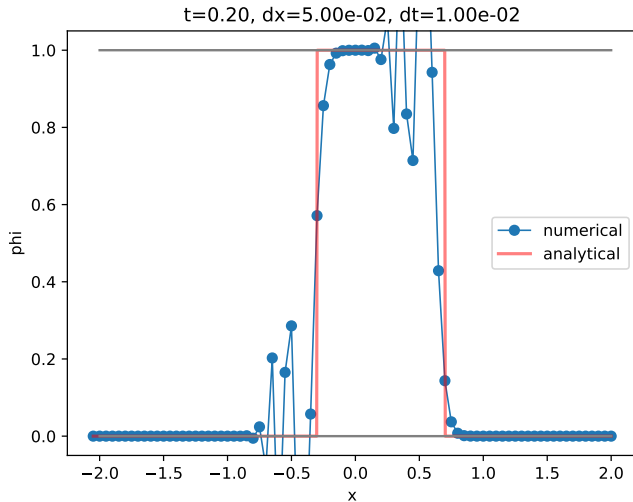
Optional:

7. Read the function `update_upwind`. What is the order of accuracy of the upwind scheme?
8. The current implementation of the upwind scheme supports only positive velocities. Look at the current implementation (`update_upwind`) and add the handling of negative velocities.

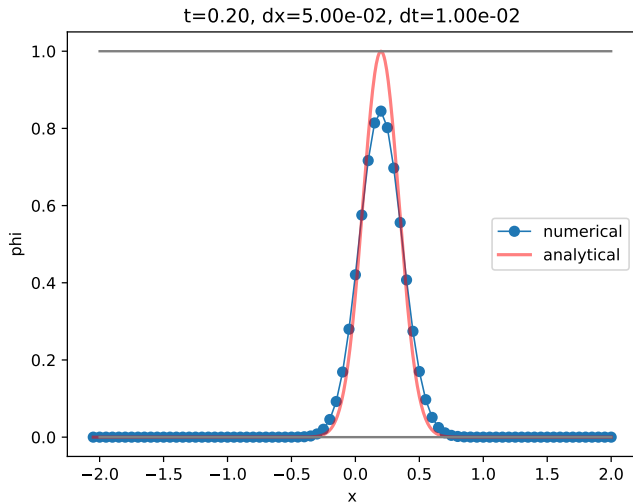
Example 5 – Scalar transport – results – central difference scheme – Gauss



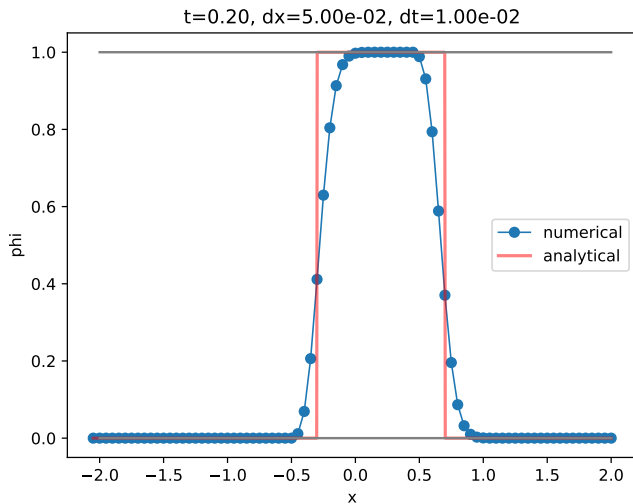
Example 5 – Scalar transport – results – central difference scheme – step



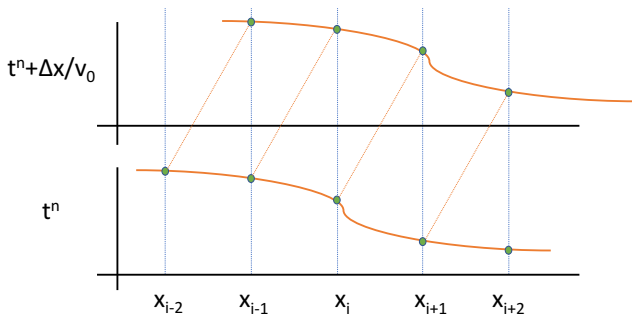
Example 5 – Scalar transport – results – upwind scheme – Gauss



Example 5 – Scalar transport – results – upwind scheme – step



Upwind scheme (I)



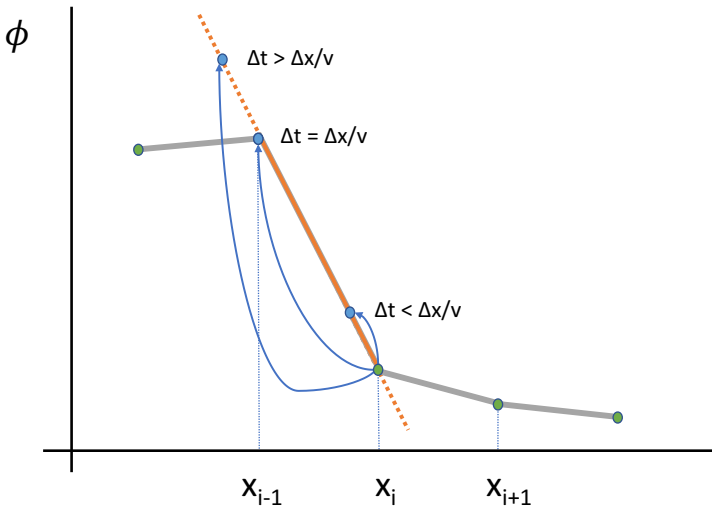
For a positive ($v_0 > 0$) advection velocity the upwind scheme is given by

$$\phi_i^{n+1} = \phi_i^n - v_0 \Delta t \frac{\phi_i^n - \phi_{i-1}^n}{\Delta x}$$

In the case of $\Delta t = \Delta x / v_0$:

$$\phi_i^{n+1} = \phi_{i-1}^n$$

Upwind scheme (II)



3. Scalar Transport

3.1 Introduction

3.2 Flux Limiter

3.3 Conclusions

Basic idea

The advection equation can be represented in the conservative form for $\phi = \rho Z$ as

$$\partial_t(\phi) + \nabla \cdot \vec{F} = DS(\rho, Z) \quad \text{with the flux} \quad \vec{F} = \phi \vec{u}$$

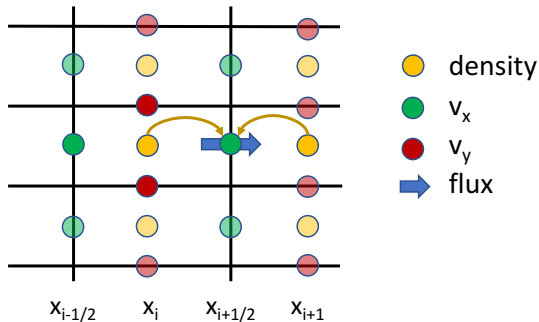
where for sake of simplicity the index α is omitted and the diffusion and source terms are represented by DS .

The basic idea in the flux limiting schemes is to handle the flux in a way to prevent increasing oscillations.

The so called total variation diminishing (TVD) schemes preserve (or reduce) the variation of the scalar field by adjusting the flux.

Staggered grid

Instead of a cell centered evaluation of the flux in the advection equation, it is evaluated at the cell faces. As FDS uses a staggered grid, where vector quantities are located on the cell faces, the second order divergence evaluation is similar to the FVM approach.



Flux interpolation

The flux handling in FDS is done by limiting the transported scalar values in the flux evaluation. In 1D, it results in:

$$\partial_t \phi + \frac{F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}}}{\Delta x} = DS \quad \rightarrow \quad \partial_t \phi + \frac{\overline{\phi^{FL}}_{i+\frac{1}{2}} u_{i+\frac{1}{2}} - \overline{\phi^{FL}}_{i-\frac{1}{2}} u_{i-\frac{1}{2}}}{\Delta x}$$

The interpolation of the transported scalar is based on local variations and in the upstream (sign of $u_{i+\frac{1}{2}}$) direction

$$\delta\phi_{loc,i+\frac{1}{2}} = \delta\phi_{loc} = \phi_{i+1} - \phi_i$$

$$\delta\phi_{up,i+\frac{1}{2}} = \delta\phi_{up} = \begin{cases} \phi_i - \phi_{i-1} & \text{if } u_{i+\frac{1}{2}} > 0 \\ \phi_{i+2} - \phi_{i+1} & \text{if } u_{i+\frac{1}{2}} < 0 \end{cases}$$

Flux limiter

The general representation of different limiter schemes can be expressed via the limiter function $B(r)$ with r being the ratio of successive variations

$$r = \frac{\delta\phi_{up}}{\delta\phi_{loc}} \quad \text{and} \quad \overline{\phi^{FL}}_{i+\frac{1}{2}} = \begin{cases} \phi_i + B(r)\frac{1}{2}\delta\phi_{\alpha} & \text{if } u_{i+\frac{1}{2}} > 0 \\ \phi_{i+1} - B(r)\frac{1}{2}\delta\phi_{\alpha} & \text{if } u_{i+\frac{1}{2}} < 0 \end{cases}$$

Flux Limiter Scheme	$B(r)$	α	FLUX_LIMITER
central difference	1	loc	0
Gudunov	0	loc	1
Superbee (LES default)	$\max(0, \min(2r, 1), \min(r, 2))$	loc	2
MINMOD	$\max(0, \min(1, r))$	loc	3
CHARM (DNS default)	$(3\frac{1}{r} + 1) / (r(\frac{1}{r} + 1)^2)$	up	4

Example 6 – Scalar transport with FDS

Input directory: 06_fds_flux_limiter

Input file: move_slug.fds

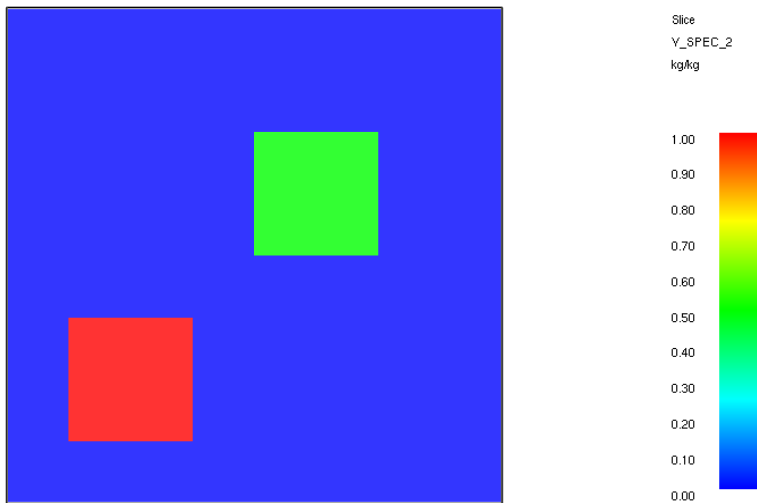
Verification input file with a reduced simulation time of 1 s.

Goal: Investigate the different flux limiter available in FDS.

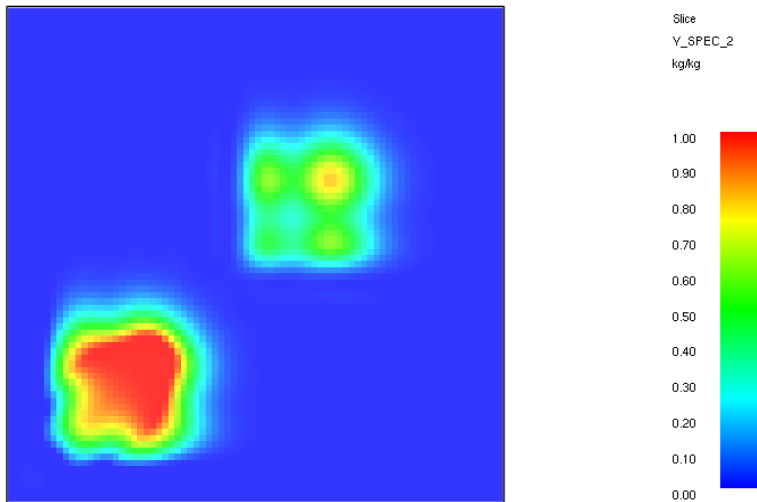
Tasks:

1. Read the FDS input file. What is the simulation scenario? What should happen?
2. Run FDS with the input file.
3. Change to `FLUX_LIMITER` to central differences and Godunov schemes. Do you observe the expected behaviour?

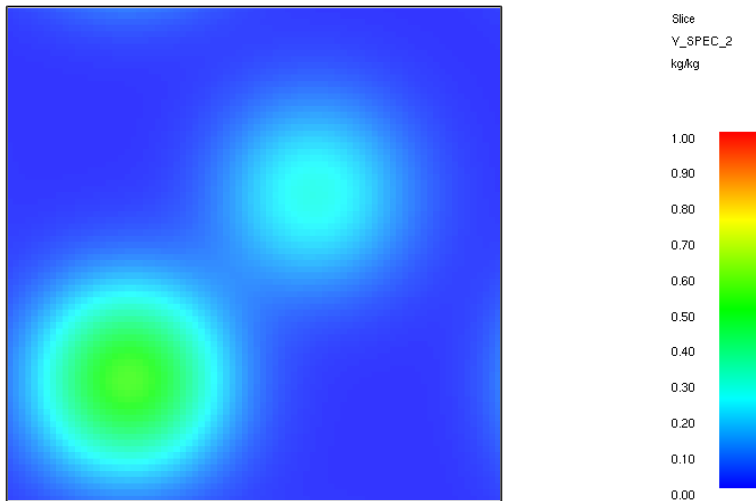
Example 6 – Scalar transport with FDS – results – initial values



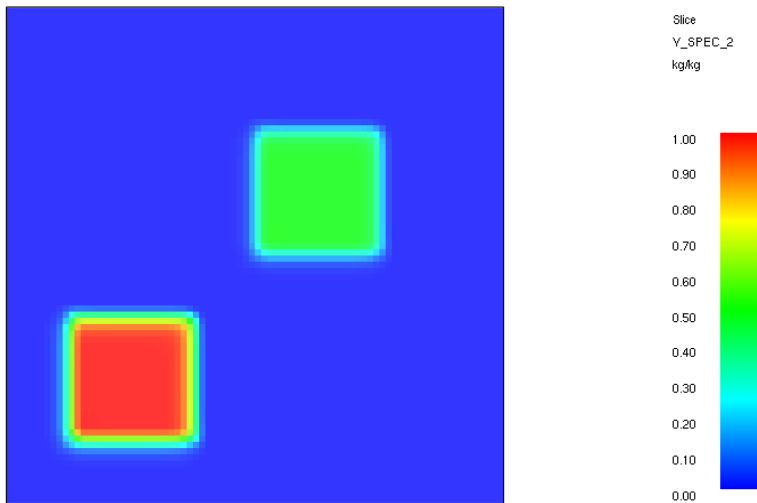
Example 6 – Scalar transport with FDS – results – FLUX_LIMITER=0



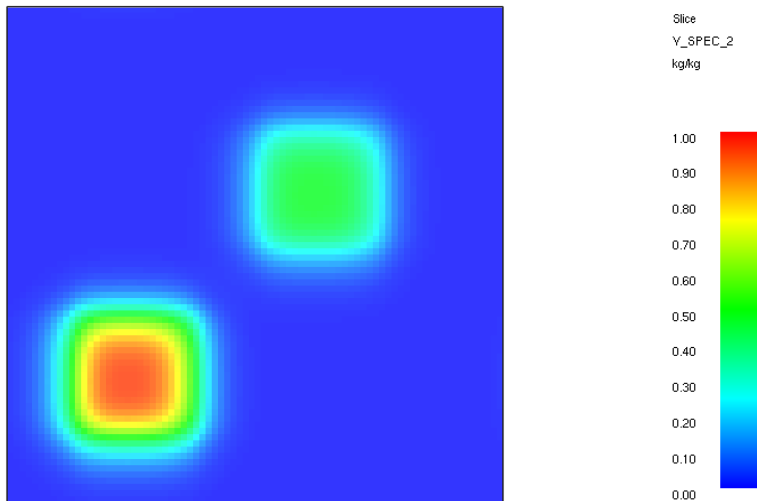
Example 6 – Scalar transport with FDS – results – FLUX_LIMITER=1



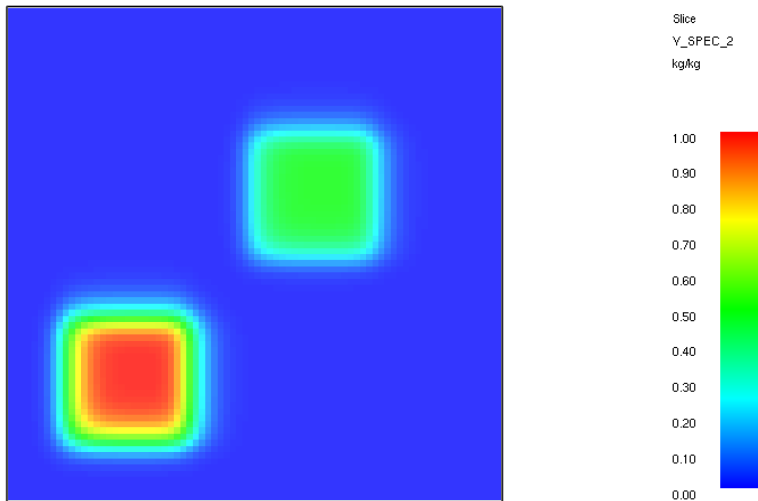
Example 6 – Scalar transport with FDS – results – FLUX_LIMITER=2



Example 6 – Scalar transport with FDS – results – FLUX_LIMITER=3



Example 6 – Scalar transport with FDS – results – FLUX_LIMITER=4



Enforcing realizability

Although TVD prevent fluctuations, they can do it in general only in 1D, but not in 3D. Therefore spurious fluctuations may lead to e.g. negative densities.

The transport scheme must support the realizability:

$$Y_\alpha \geq 0 \quad \text{and} \quad \sum Y_\alpha = 1$$

However, if $(\rho Y)_\alpha$ obeys $(\rho Y)_\alpha \geq 0$, then Y_α is guaranteed to be realizable.

Thus, there exists a minimal density threshold ρ_{min} , that should be preserved.

Scalar boundedness correction

In 1D, the correction $\delta\rho$ for densities with a computed value ρ_i^* below ρ_{min} should satisfy:

- ▶ boundedness: $\rho_i = \rho_i^* + \delta\rho_i \geq \rho_{min}$
- ▶ conservation: $\sum_i \delta\rho_i V_i = 0$
- ▶ minimal variation: $\sum_i |\delta\rho_i|$

It is implemented as a diffusion operation, with

- ▶ $\delta\rho_i = \rho_{min} - \rho_i^*$
- ▶ $\delta m_{i\pm 1} = -c_i (\rho_{i\pm 1}^* - \rho_i^*) V_i$

and the artificial diffusion coefficient c_i

$$c_i = \frac{\rho_{min} - \rho_i^*}{\rho_{i-1}^* - 2\rho_i^* + \rho_{i+1}^*}$$

3. Scalar Transport

3.1 Introduction

3.2 Flux Limiter

3.3 Conclusions

Conclusions

- ▶ The advection of scalar values is generally challenging topic.
- ▶ Limiting the flux, via TVD schemes like upwind or Superbee schemes, allows for stable solutions.
- ▶ Still, fluctuations in density enforce corrections methods.