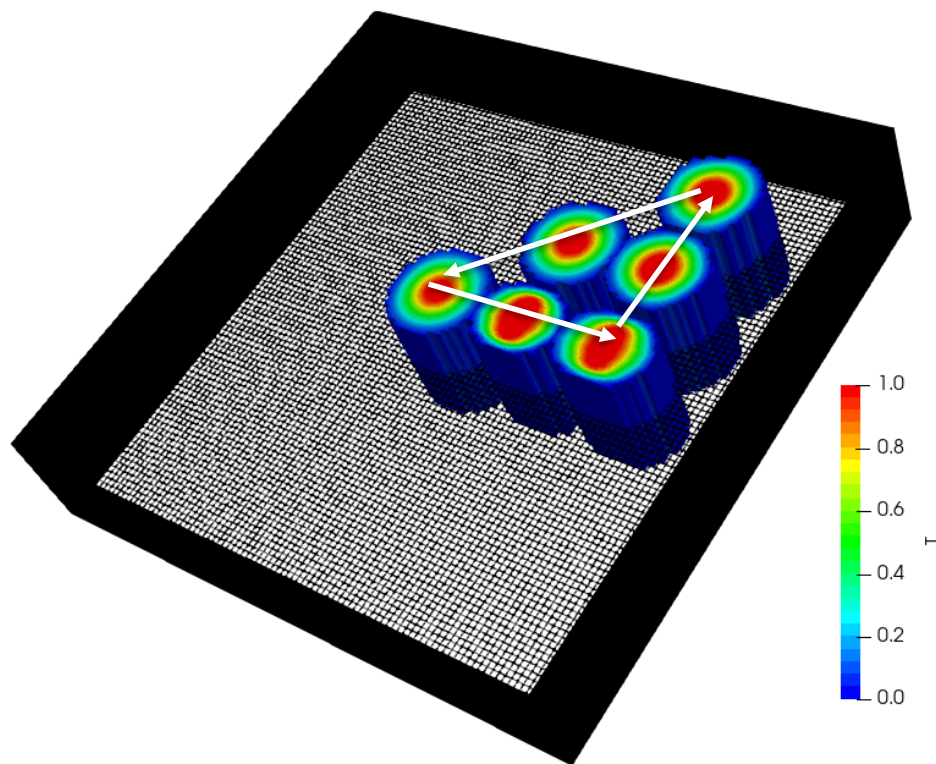


# Tutorial Five


## Discretization – part 2



Bahram Haddadi



7<sup>th</sup> edition, March 2025

 Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by-nc-sa/3.0/>

## Contributors:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek
- Clemens Gößnitzer
- Sylvia Zibuschka
- Yitong Chen
- Vikram Natarajan
- Jozsef Nagy



Technische Universität Wien  
Institute of Chemical, Environmental  
& Bioscience Engineering



Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)

This is a human-readable summary of the Legal Code (the full license).

## Disclaimer

You are free:

- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work

Under the following conditions:

- Attribution — you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that, they endorse you or your use of the work).
- Noncommercial — you may not use this work for commercial purposes.
- Share Alike — if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

- Waiver — any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain — where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- Other Rights — In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- Notice — for any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

This offering is not approved or endorsed by ESI® Group, ESI-OpenCFD® or the OpenFOAM® Foundation, the producer of the OpenFOAM® software and owner of the OpenFOAM® trademark.

Available from: [www.fluidynamics.at](http://www.fluidynamics.at)

## Background

### 1. Properties of discretization schemes

When performing numerical simulations, it is crucial to choose the right discretization scheme to ensure physically realistic results. The effectiveness of a discretization scheme depends on several key properties, including conservativeness, boundedness, and transportiveness. Understanding these properties helps in selecting the appropriate scheme for a given problem in Computational Fluid Dynamics (CFD). These properties also influence the numerical accuracy, stability, and efficiency of the simulation.

#### 1.1. Conservativeness

A discretization scheme is conservative if it ensures that the total amount of a transported quantity (e.g., mass, momentum, energy) is preserved within the solution domain. This property is fundamental for obtaining physically meaningful results in fluid dynamics and preventing artificial gain or loss of the transported variable.

To achieve conservativeness, the flux balance across each control volume must be maintained. Mathematically, this means:

- The flux of  $\phi$  leaving a control volume across a certain face must equal the flux entering the adjacent control volume through the same face.
- The discretization scheme should represent the flux through a common face consistently across adjacent control volumes.

A scheme that violates conservativeness can lead to unphysical results, such as artificial creation or loss of mass or energy. Finite volume methods naturally ensure conservation by integrating the governing equations over control volumes, ensuring that what exits one control volume enters the next.

#### 1.2. Boundedness

Most numerical solvers use iterative techniques to obtain the solution at each node. The solver starts with an initial guess and updates the values until convergence is achieved. To ensure a stable and physically meaningful solution, the discretization scheme must satisfy boundedness criteria.

A bounded solution means that the numerical values of  $\phi$  remain within reasonable limits, avoiding unrealistic oscillations or negative concentrations, which would be non-physical.

The sufficient condition for condition for boundedness is:

$$\frac{\sum |a_{nb}|}{|a'_p|} \begin{cases} \leq 1 & \text{at all nodes} \\ < 1 & \text{at one node at least} \end{cases}$$

Here  $a'_p$  is the net coefficient of the central node P (i.e.  $a'_p = S_p$ ),  $a_{nb}$  are the coefficient of the neighbouring nodes. If the condition is satisfied, the resulting matrix of coefficients is diagonally dominant. We need the net coefficients to be

as large as possible; this means that  $S_p$  should be always negative. If this is the case,  $S_p$  becomes positive due to the modulus sign and adds to  $a_p$ .

### 1.3. Transportiveness

Transportiveness refers to the ability of a discretization scheme to correctly account for the dominant transport mechanism in a problem. This is assessed using the Peclet number (Pe), which measures the relative strength of convection versus diffusion:

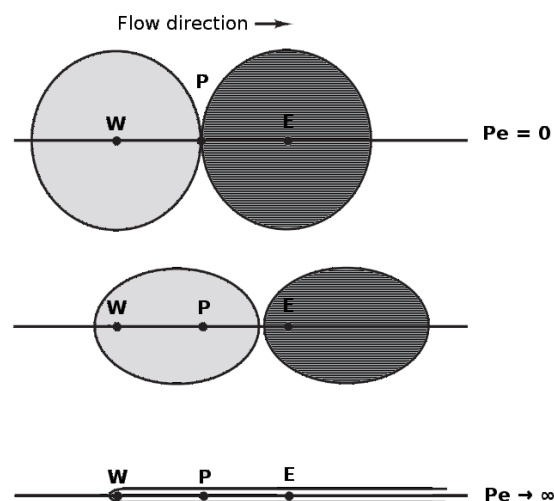
$$Pe = \frac{N_{conv}}{N_{diff}} = \frac{LU}{D}$$

*Note:  $L$  is a characteristic length scale,  $U$  is the velocity magnitude,  $D$  is a characteristic diffusion coefficient.*

The primary goal is to ensure that the transportiveness is borne out of the discretization scheme.

Let us consider the effect at a point P due to two constant sources of  $\phi$  at nearby points W and E on either side, in three cases.

1. When  $Pe = 0$  (pure diffusion), the contours of  $\phi$  are circles, as  $\phi$  is spread out evenly in all directions
2. As  $Pe$  increases, the contours become elliptical, as the values of  $\phi$  are influenced by convection
3. When  $Pe \rightarrow \infty$ , the contours become straight lines, since  $\phi$  are stretched out completely and affected only by upstream conditions



#### 4. Transportiveness property

## 2. Assessing the general discretization schemes

It is useful to compare the different types of general discretization schemes covered in Tutorial Four based on their conservativeness, boundedness and transportiveness properties.

### Different discretizing schemes assessment

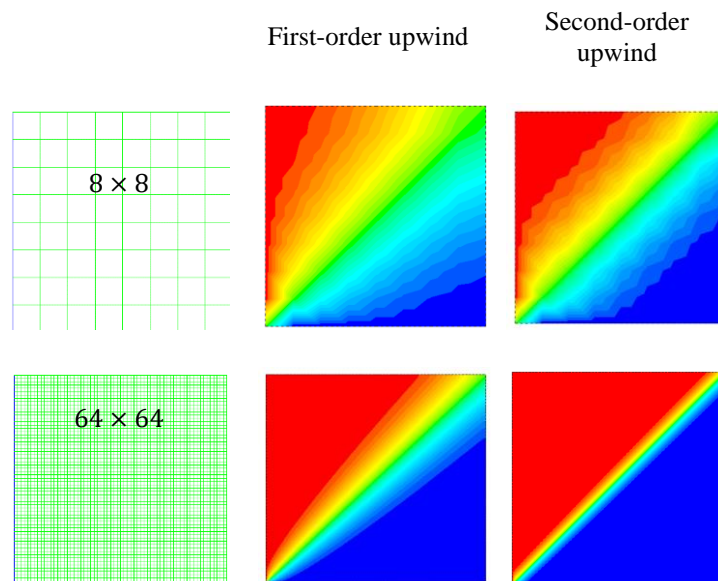
Scheme	Conser- vative	Bounded	Accuracy	Trans- portive	Remarks
Upwind	Yes	Unconditionally bounded	First order	Yes	Include false diffusion if the velocity vector is not parallel to one of the coordinate directions
Central Differencing	Yes	Conditionally bounded*	Second order	No	Unrealistic solutions at large Pe number
QUICK	Yes	Unconditionally bounded	Third order	Yes	Less computationally stable. Can give small undershoots and overshoots

\* *Pe should be less than 2.*

### 3. Numerical (false) diffusion

Numerical diffusion is an artificial diffusion effect that occurs when the flow direction is not aligned with the computational grid. It is a numerical artifact that introduces additional diffusion into the system and primarily affects convection-dominated flows with high Peclet numbers (Pe).

False diffusion is more prominent when using first-order upwind schemes. It decreases with finer grids, but using higher-order schemes (e.g., QUICK) is a more effective way to reduce it. False diffusion can distort flow structures, leading to non-physical results, especially in high-speed flows. Using a high-resolution grid or aligning the mesh with the flow direction can help mitigate numerical diffusion.



Numerical diffusion

#### 4. Numerical behavior of OpenFOAM® discretization schemes

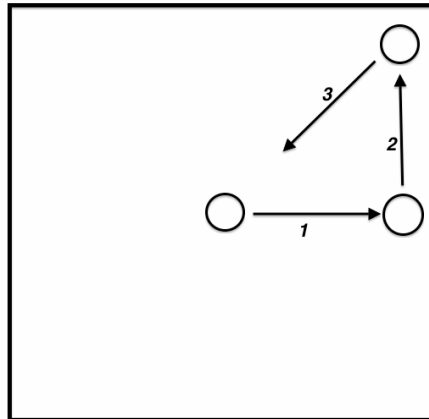
The choice of discretization scheme for this tutorial should depend critically on the numerical behavior of the scheme. Using higher order schemes, numerical diffusion errors can be reduced, however it requires higher computational efforts.

Scheme	Numerical behavior
upwind	First order, bounded
linear	Second order, unbounded
linearUpwind	First/second order, bounded
QUICK	Second order or higher, bounded
cubic	Fourth order, unbounded

## functions Solver – circle

### Tutorial outline

Use the functions solver, do simulate the movement of a circular scalar spot region (radius = 1 m) at the middle of a  $100 \times 100$  cell mesh ( $10 \text{ m} \times 10 \text{ m}$ ), then move it to the right (3 m), to the top (3 m) and diagonally.



Schematic sketch of the problem

### Objectives

- Choosing the best discretization scheme.

### Data processing

Examine your simulation in ParaView.

## 1. Pre-processing

### 1.1. Compile tutorial

Create the new case in your working directory like in tutorial four.

### 1.2. 0 directory

To move the circle to right change the `internalField` to `(1 0 0)` in the U file for setting the velocity field towards the right.

### 1.3. system directory

Modify the *blockMeshDict* for creating a 2D geometry with 100 × 100 cells mesh.

```
// *****
*****//
convertToMeters 1;

vertices
(
    (-5 -5 -0.01)
    (5 -5 -0.01)
    (5 5 -0.01)
    (-5 5 -0.01)
    (-5 -5 0.01)
    (5 -5 0.01)
    (5 5 0.01)
    (-5 5 0.01)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (100 100 1) simpleGrading (1 1 1)
);
edges
(
);
boundary
(
    sides
    {
        type patch;
        faces
        (
            (1 2 6 5)
            (0 4 7 3)
            (3 7 6 2)
            (0 1 5 4)
        );
    }
    empty
    {
        type empty;
        faces
        (
            (5 6 7 4)
            (0 3 2 1)
        );
    }
);
// *****
*****//
```



Choose a discretization scheme based on the results from the previous example and set it in the *fvSchemes*.

In the *setFieldsDict* patch a circle to the middle of the geometry using the following lines.

```
// * * * * *
* * * * *//

defaultFieldValues (volScalarFieldValue T 0 );

regions
(
cylinderToCell
{
    p1 ( 0 0 -1 );
p2 ( 0 0 1 );
    radius 0.5;
    fieldValues
    (
volScalarFieldValue T 1
    ) ;
}
);

// * * * * *
* * * * *//
```

*cylinderToCell* command is used to patch a cylinder to the region, *p1* and *p2* show the two ends of cylinder center line, in the *radius* the radius is set.

Check *controlDict*, in the first part of simulation, where the circle should move to the right set the *startFrom* to *startTime* and *startTime* to 0. By a simple calculation, it can be seen that the *endTime* should be 3s (to move the circle from center to the right side). Similar calculations need to be done for the two other parts, except the *startTime* is set to the *endTime* of previous part, and new *endTime* should be that part “simulation time” plus *endTime* of the previous part.

**Note:** In the functions file set *D* to zero (no diffusion!).

## 2. Running Simulation

```
>blockMesh
```

```
>setFields
```

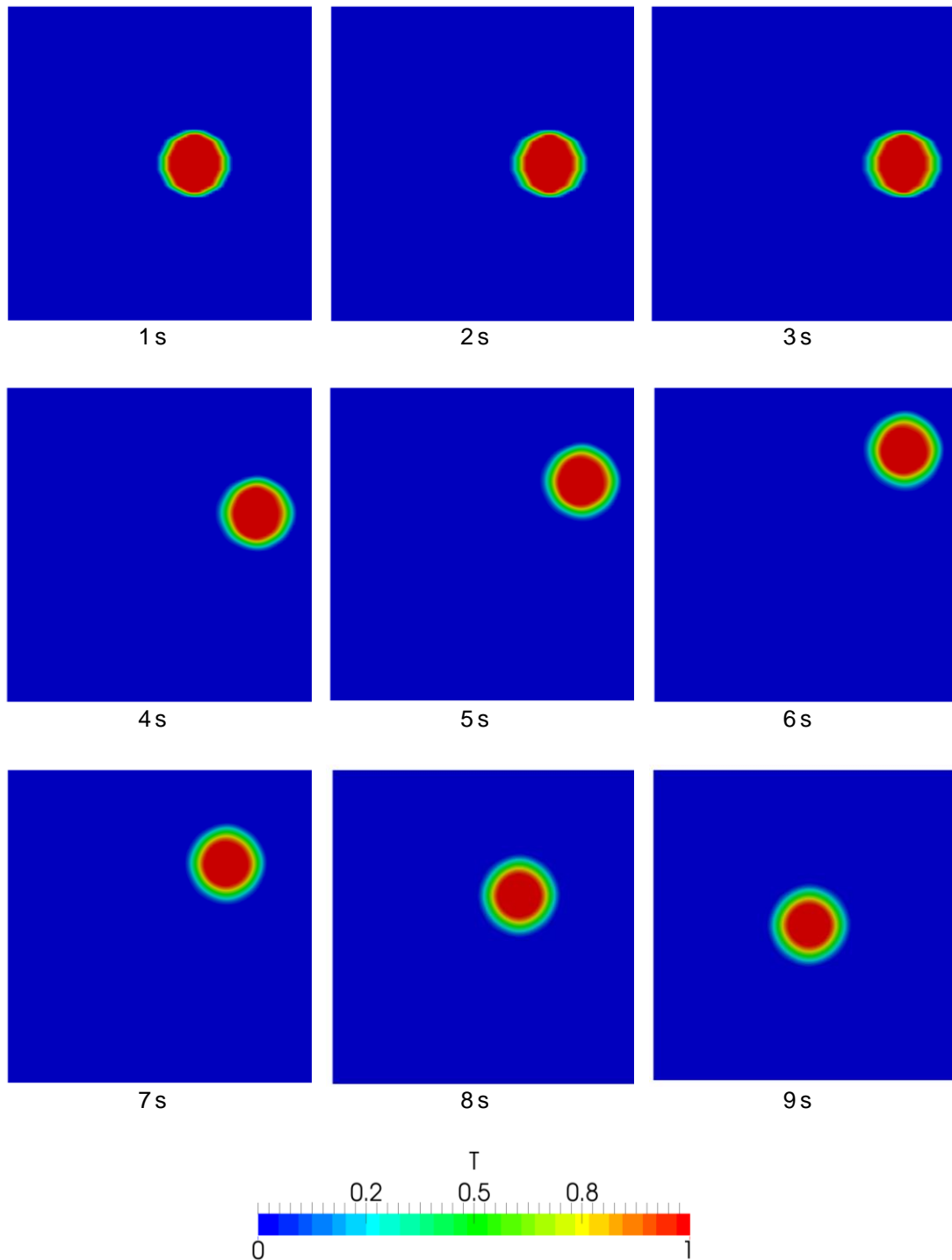
```
>foamRun -solver functions
```

For running further parts (moving the circle to top, and then diagonally), in the 0 folder in the U file change the *internalField* velocity to (0 1 0) so the circle moves up, and to (-1 -1 0) to move the circle diagonally back to the original position.

**Note:** In the *controlDict* file, *subSolverTime* is set to 0 and therefore even if the *startTime* is set to latestTime, the simulation will read the U file from time 0!

## 3. Post-processing

The simulation results are as follows:



Position of the circle at different time steps