Numerical playground

Merry-go-round:

Pure convection of a passive scalar in a vector field – One dimensional tube.

Numerical playground

- We will not run this case, this is a visual and mental exercise only.

- You will find this case in the directory

`$PTOFC/101FVM/pureConvection`

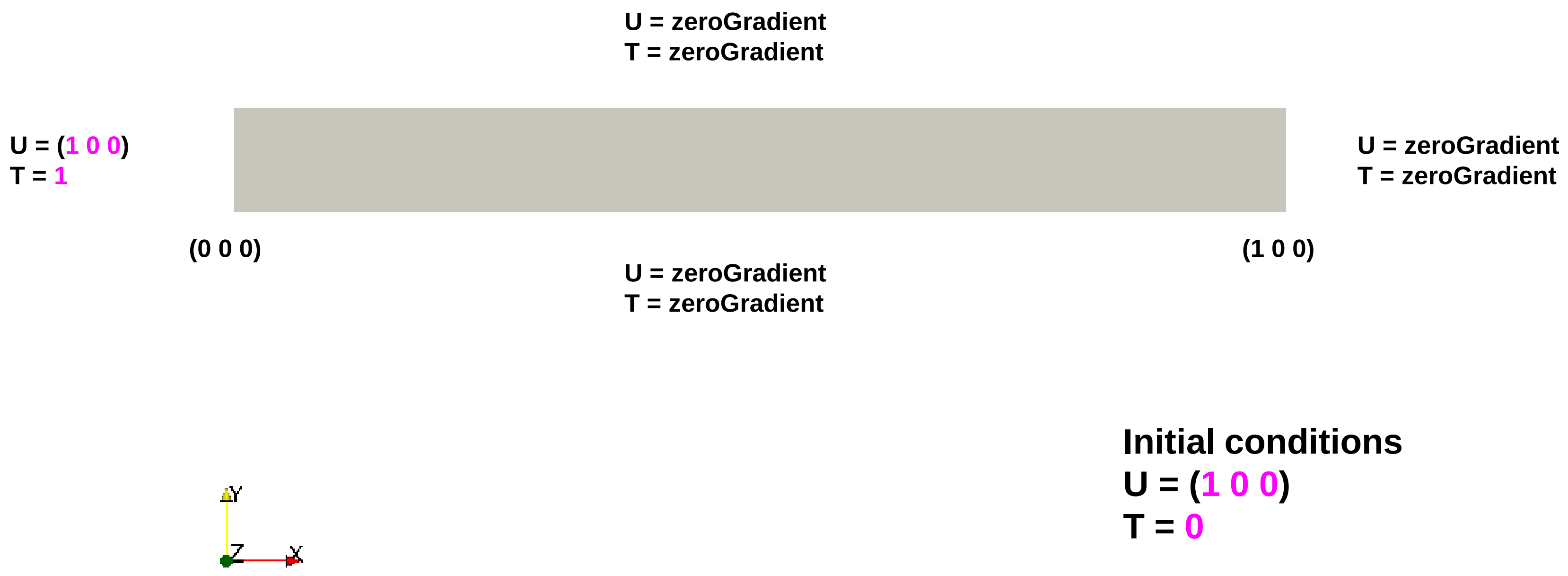
- In this directory, you will also find the *`README.FIRST`* file with the instructions of how to run the case.

- Hereafter, we will focus our eyes to focus our brain.

Numerical playground

Pure convection of a scalar in a vector field – One dimensional tube.

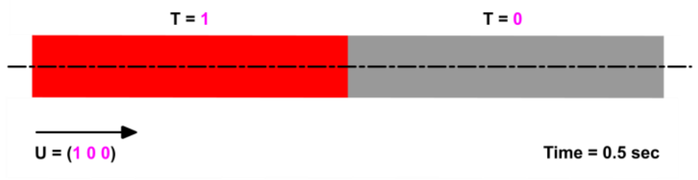
$$\frac{\partial T}{\partial t} + \nabla \cdot (\phi T) - \cancelto{0}{\nabla \cdot (\Gamma \nabla T)} = 0$$

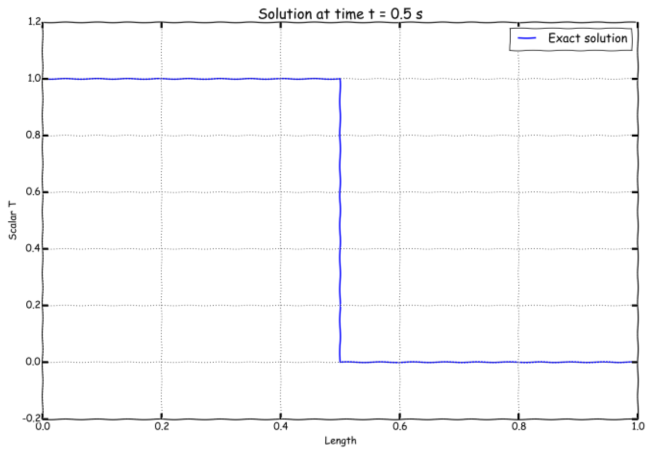


Initial conditions
U = (1 0 0)
T = 0

Numerical playground

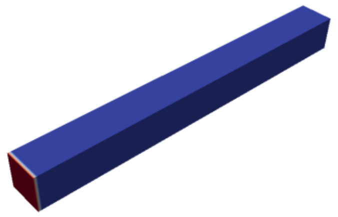
- This problem has an exact solution in the form of a traveling wave.
- We will use this case to study the different discretization schemes implemented in OpenFOAM®.
- In the figure, we show the solution for time = 0.5 s





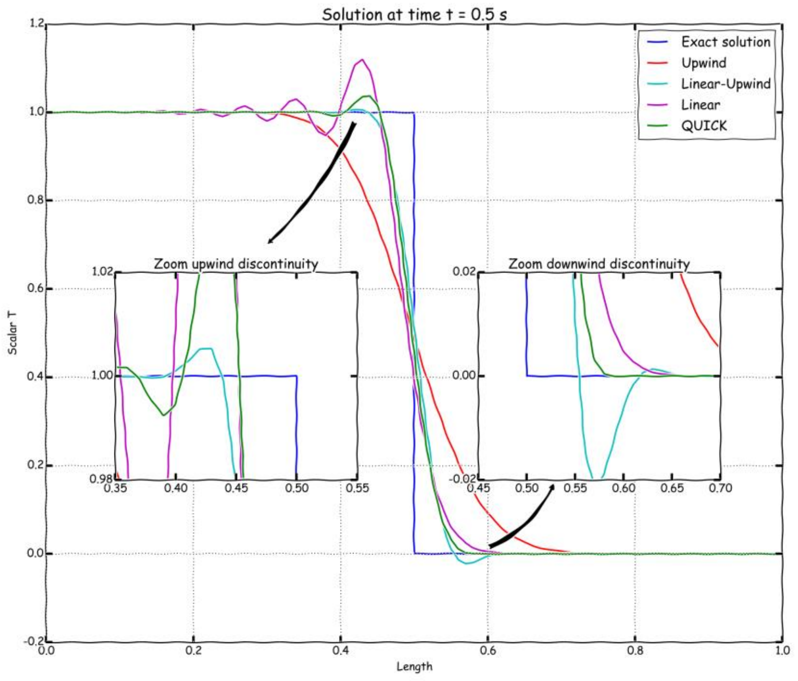
www.wolfdynamics.com/wiki/pureconvection/xani1.gif

www.wolfdynamics.com/wiki/pureconvection/xani2.gif

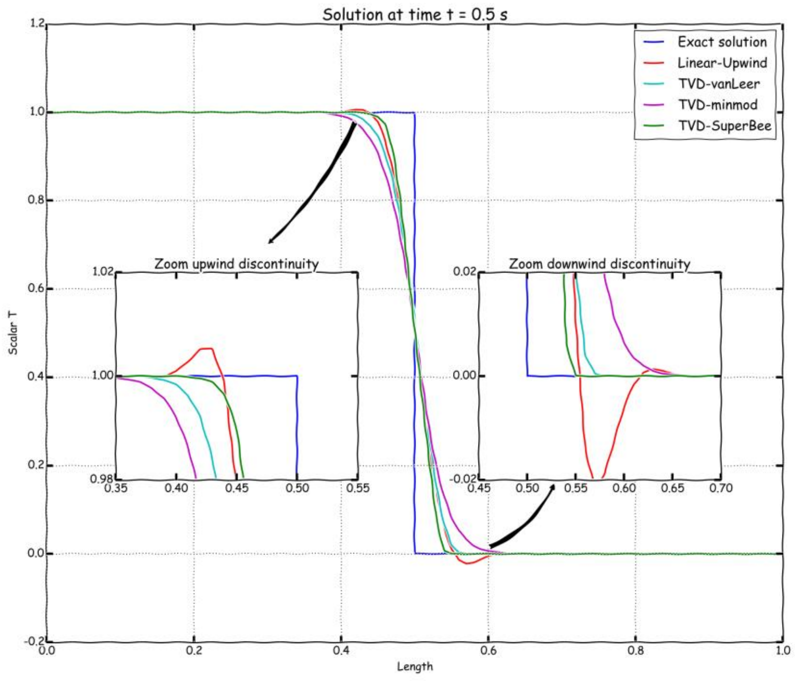


Numerical playground

Comparison of different spatial discretization schemes.
Euler in time – 100 cells – CFL = 0.1
Linear limiter functions on the Sweby diagram.

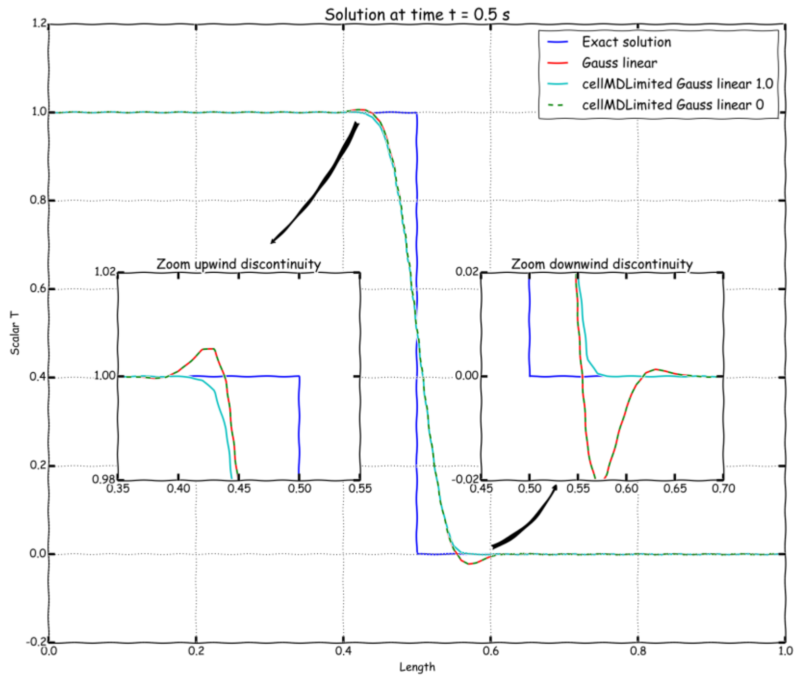


Comparison of different spatial discretization schemes.
Euler in time – 100 cells – CFL = 0.1
Non-linear limiter functions on the Sweby diagram.

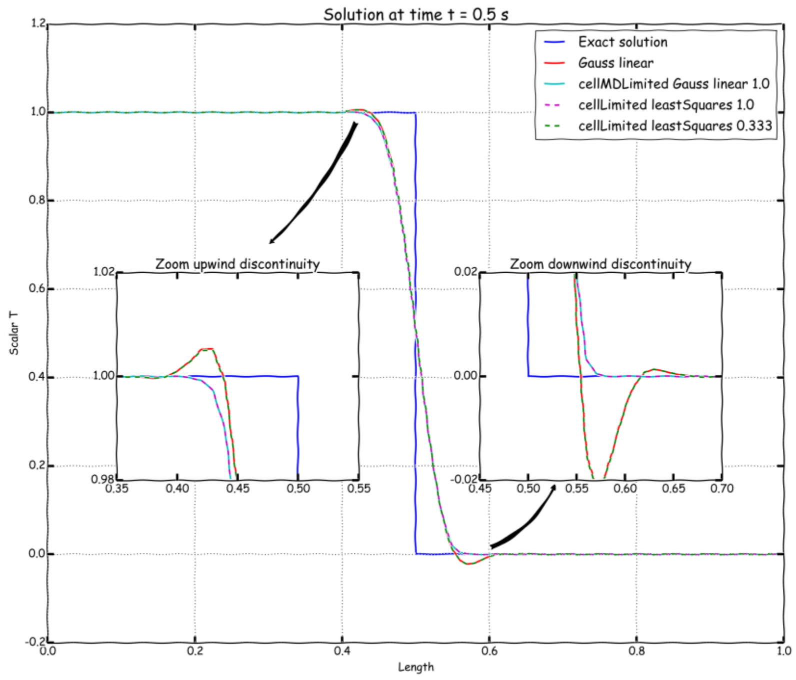


Numerical playground

Comparison of different gradient limiters.
Linear upwind in space – Euler in time – 100 cells – CFL 0.1

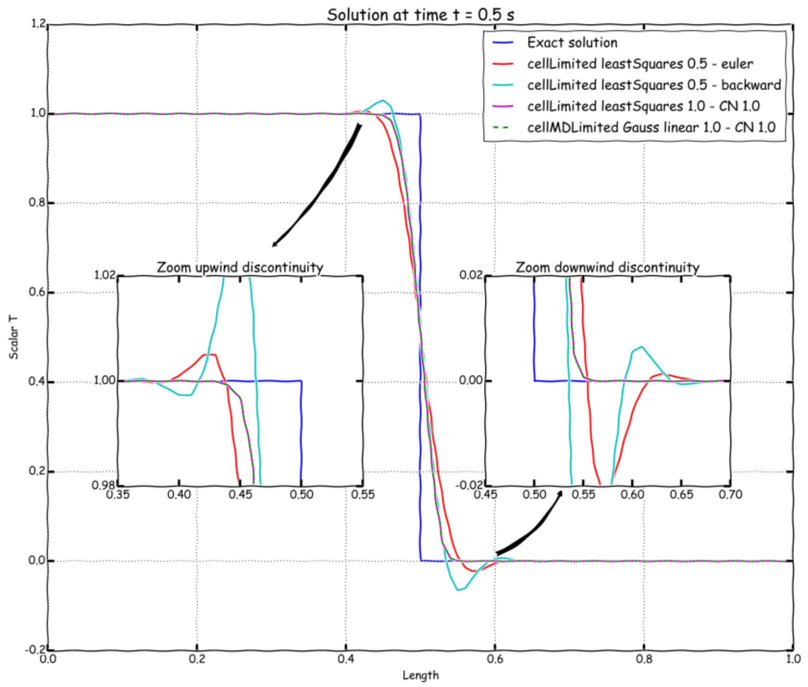


Comparison of different gradient limiters.
Linear upwind in space – Euler in time – 100 cells – CFL 0.1

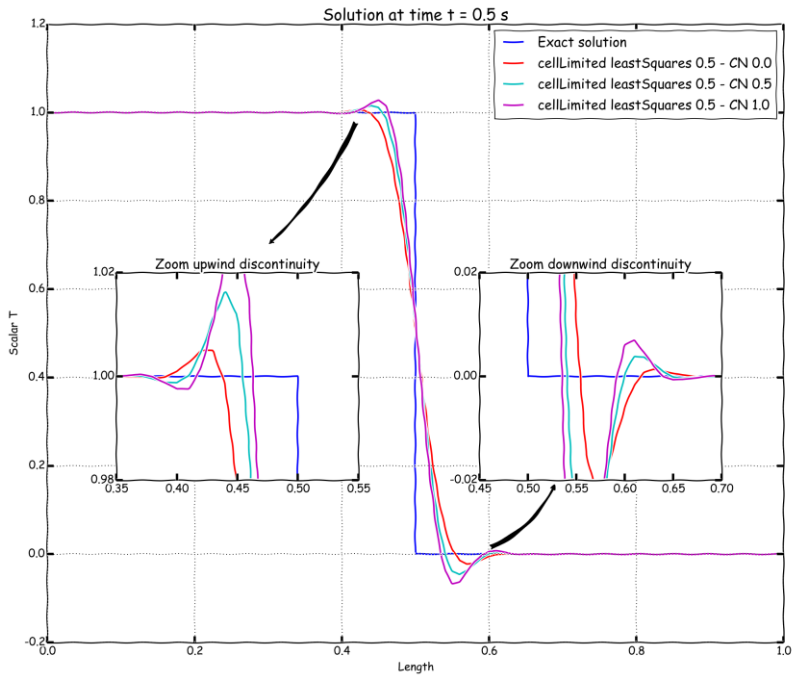


Numerical playground

Comparison of different temporal discretization schemes and gradient limiters.
Linear upwind in space – 100 cells – CFL 0.1

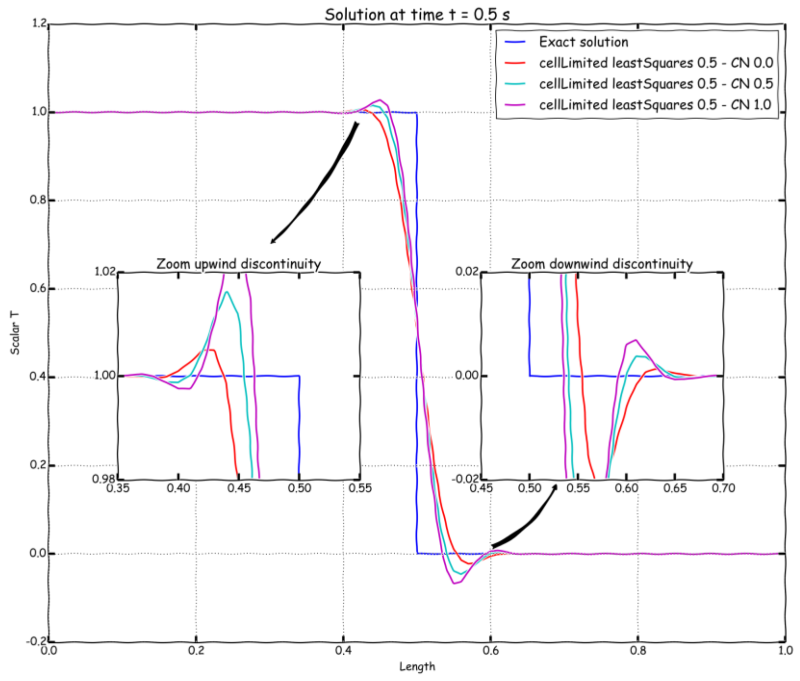


Comparison of Crank Nicolson blending factor using cellLimited leastSquares 0.5 gradient limiter.
Linear upwind in space – 100 cells – CFL 0.1

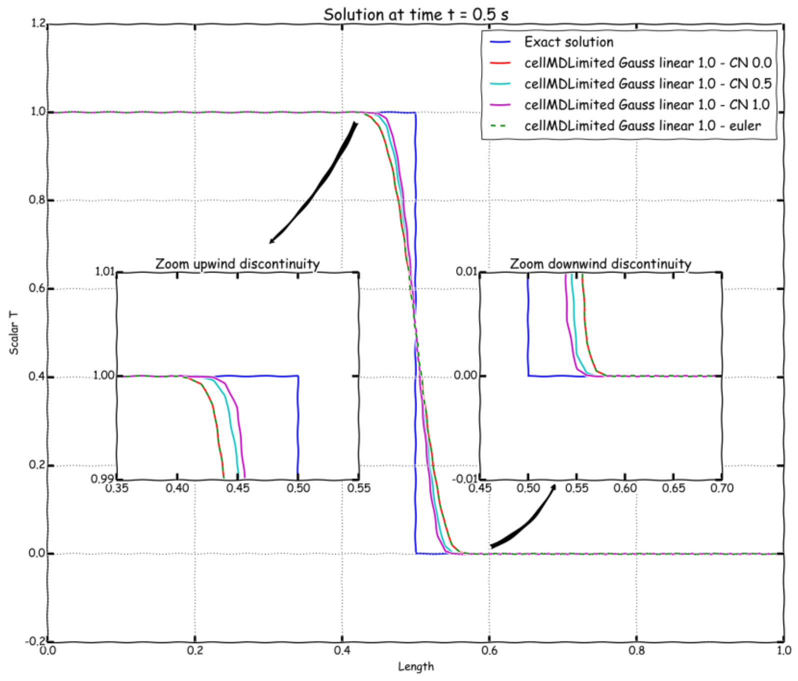


Numerical playground

Comparison of Crank Nicolson blending factor using cellLimited leastSquares 0.5 gradient limiter. Linear upwind in space – 100 cells – CFL 0.1

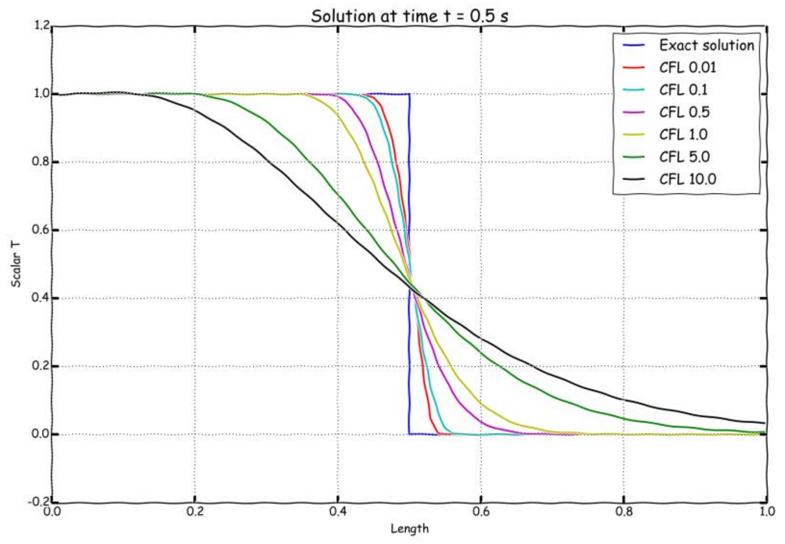


Comparison of Crank Nicolson blending factor using cellMDLimited Gauss linear 1.0 gradient limiter. Linear upwind in space – 100 cells – CFL 0.1



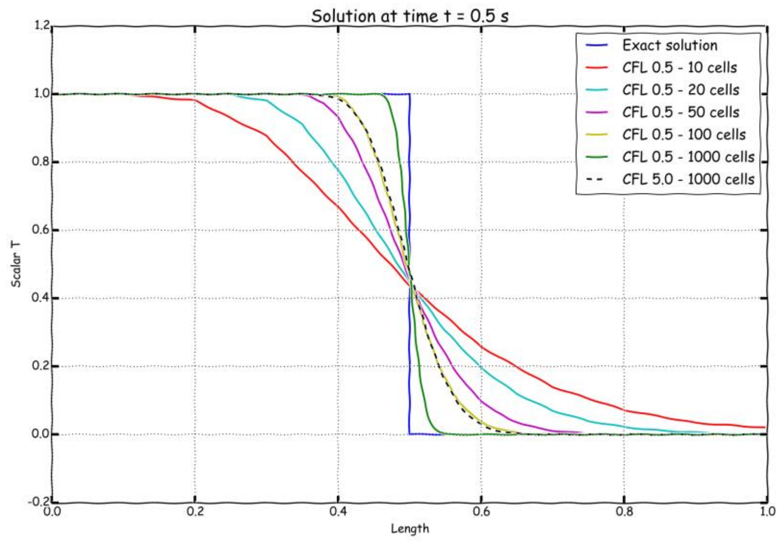
Numerical playground

Comparison of different time-step size (different CFL number).
Linear upwind in space – Euler in time – 100 cells



$$\boxed{CFL = \frac{u\Delta t}{\Delta x}}$$

Comparison of different mesh sizes.
Linear upwind in space – Euler in time



Numerical playground

- This case was for your eyes and brain only, but we encourage you to reproduce all the previous results,
    - Use all the time discretization schemes.
    - Use all the spatial discretization schemes.
    - Use all the gradient discretization schemes.
    - Use gradient limiters.
    - Use different mesh resolution.
    - Use different time-steps.

- Sample the solution and compare the results.
- Try to find the best combination of numerical schemes.
- Remember, in the *`README.FIRST`* file you will find the instructions of how to run the case.

Numerical playground

Exercises

- Which one of the following schemes is useless:
    - **upwind**
    - **downwind**
    - **linear**
- Compare the solution obtained with the following schemes:
    - **upwind**
    - **linearUpwind**
    - **MUSCL**
    - **QUICK**
    - **cubic**
    - **UMIST**

  Are all of them bounded? Are they second order accurate?

- Use the **linearUpwind** method with **Gauss linear** and **leastSquares** for gradient computations, which method is more accurate?
- Imagine that you are using the **linearUpwind** method with no gradient limiters. How will you stabilize the solution if it becomes unbounded?
- When using gradient limiters, what is clipping?
- Use the **vanLeer** method with a CFL number of 0.1 and a CFL number of 0.9, did both solutions converge? Are both solutions bounded?

Numerical playground

Exercises

- By the way, the solver `scalarTransportFoam` does not report the CFL number on the screen. How will you compute the CFL number in this case?

  (Hint: you can take a look at the post-processing slides or the utilities directory)
- Which one is more diffusive, spatial discretization or time discretization?
- Are all time discretization schemes bounded?
- If you are using the Crank-Nicolson scheme, how will you avoid oscillations?
- Does the solution improve if you reduce the time-step?
- Use the **upwind** scheme and a really fine mesh. Do the accuracy of the solution improve?
- From a numerical point of view, what is the Peclet number? Can it be compare to the Reynolds number?

$$Pe = \frac{LU}{D} = \frac{\text{convection effects}}{\text{diffusion effects}}$$

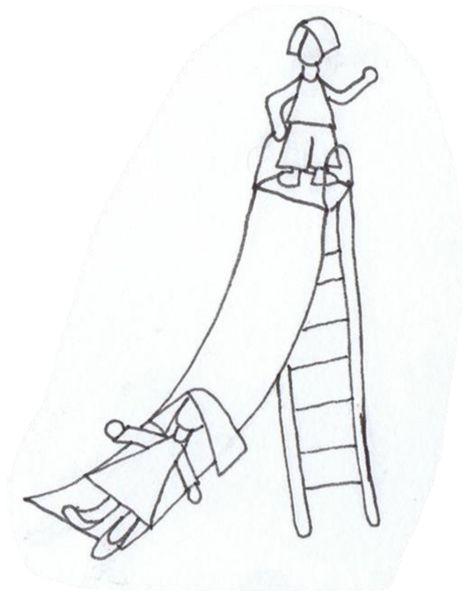
- If the Peclet number is more than 2, what will happen with your solution if you were using a **linear** scheme?

  (Hint: to change the Peclet number you will need to change the diffusion coefficient)
- Pure convection problems have analytical solutions. You are asked to design your own tutorial with an analytical solution in 2D or 3D.
- Try to break the solver using a time step less than 0.005 seconds. You are allow to modify the original mesh and use any combination of discretization schemes.

Numerical playground

Slide:

2D Laplace equation in a square domain.

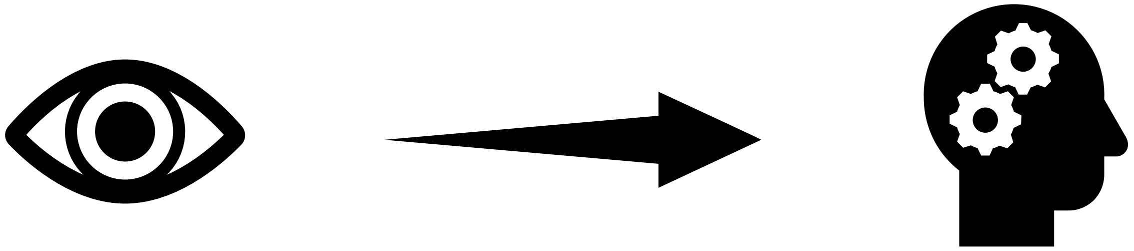


Numerical playground

- We will not run this case, this is a visual and mental exercise only.
- You will find this case in the directory

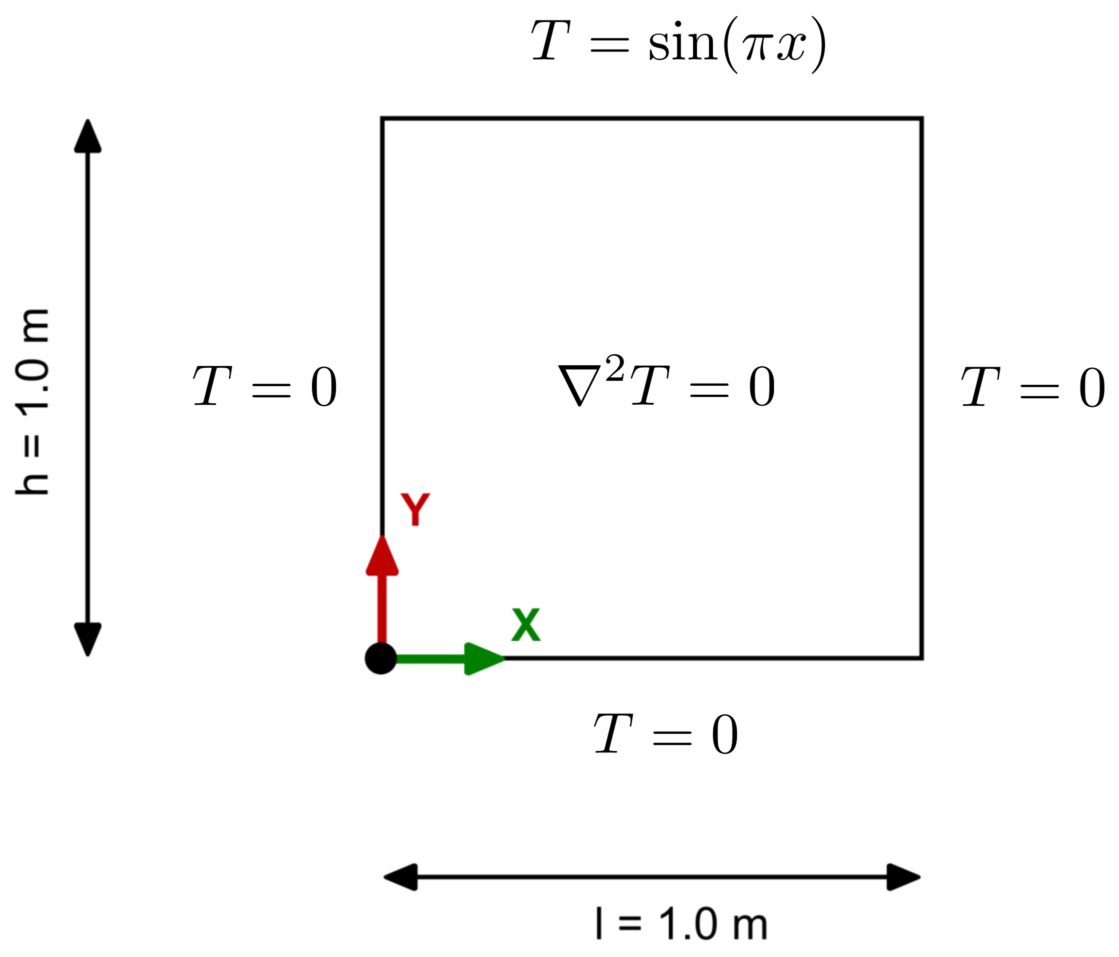
`$PTOFC/101FVM/laplace`

- In this directory, you will also find the *`README.FIRST`* file with the instructions of how to run the case.
- Hereafter, we will focus our eyes to focus our brain.



Numerical playground

2D Laplace equation in a square domain

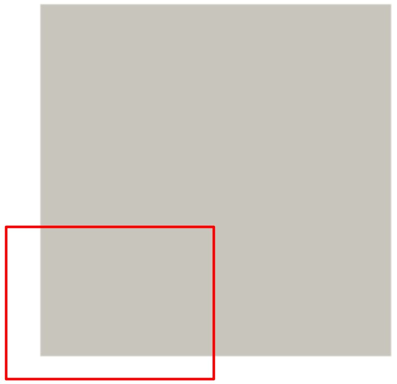


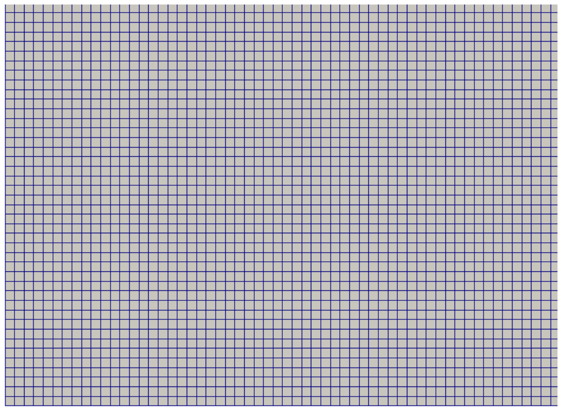
Numerical playground

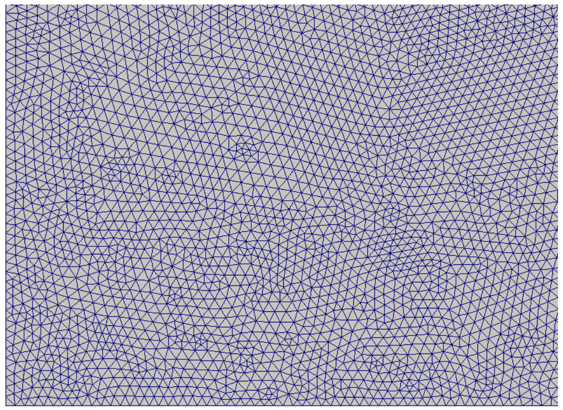
2D Laplace equation in a square domain

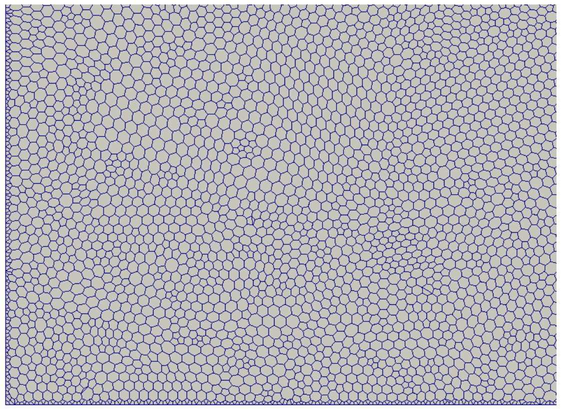
- This case consist of one domain and three different element types.

Domain




Hexahedral mesh

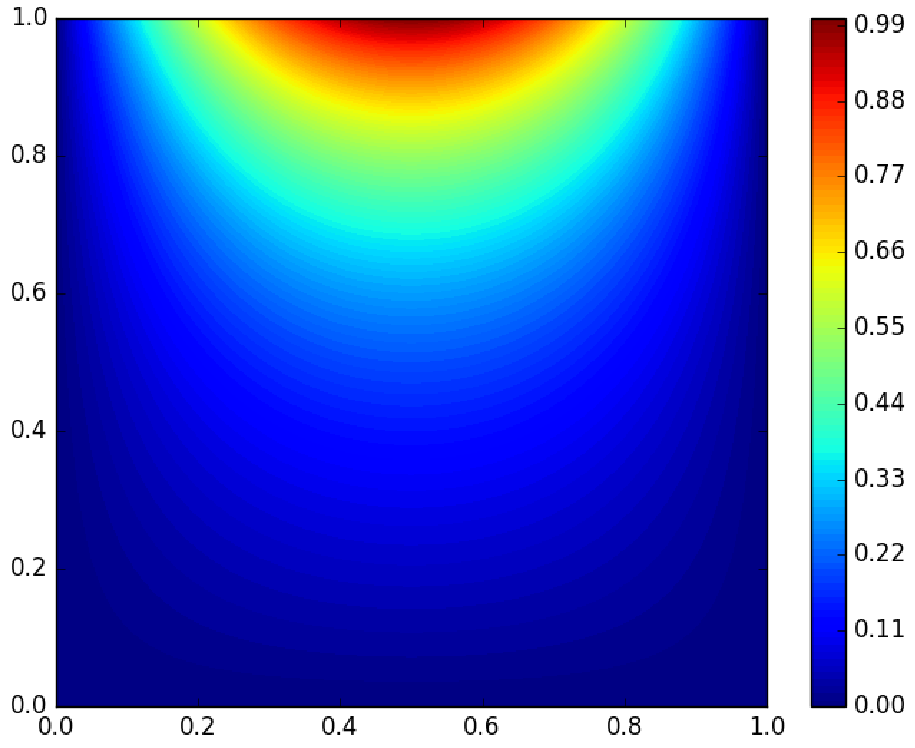

Triangular mesh


Polyhedral mesh

Numerical playground

2D Laplace equation in a square domain

- We will study the influence of the element type on the gradients computation.
- We will study the influence of the **gradSchemes** method and **laplacianSchemes** method on the solution.

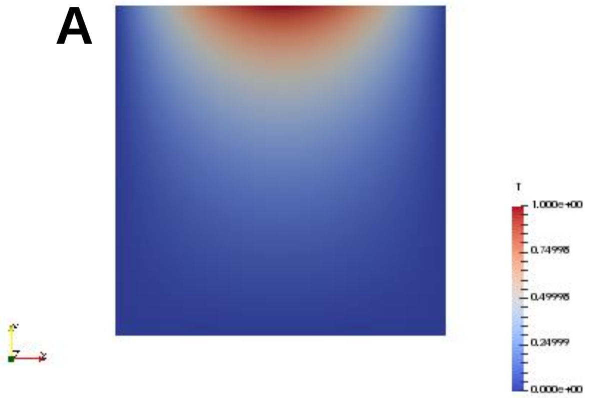


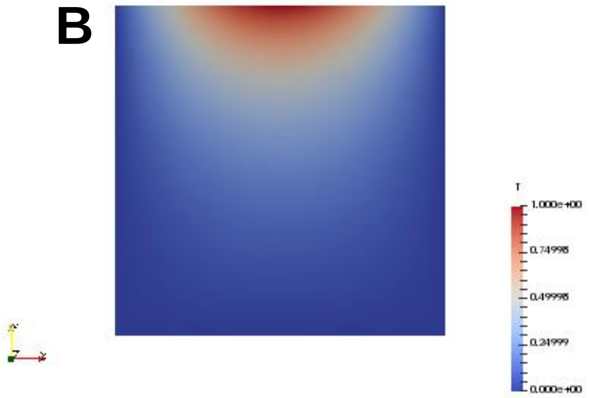
This problem has the following analytical solution:

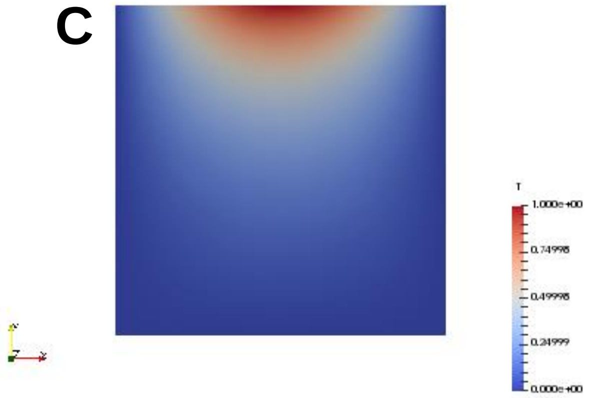
$$T(x,y) = \frac{\sin(\pi x) \times \sinh(\pi y)}{\sinh(\pi)}$$

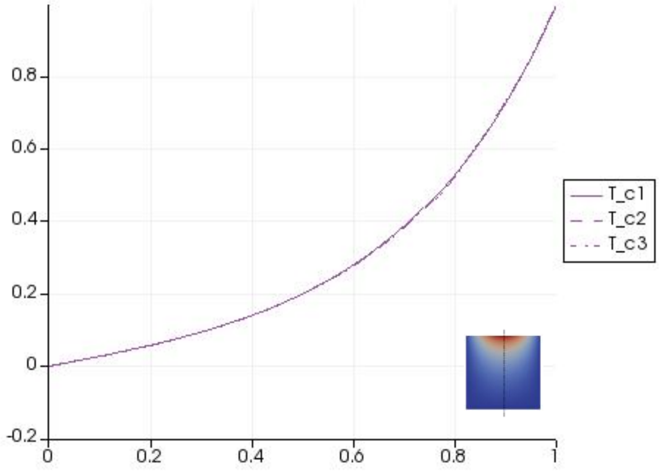
Numerical playground

2D Laplace equation in a square domain

A


B


C




T field

gradSchemes:
Gauss linear

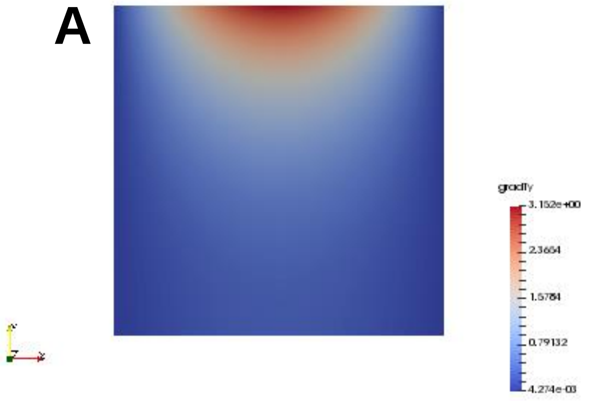
laplacianSchemes:
Gauss linear orthogonal

A. Hexahedral mesh
B. Triangular mesh
C. Polyhedral mesh

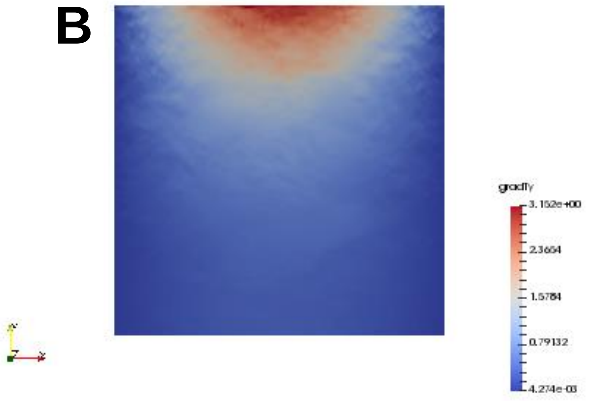
Numerical playground

2D Laplace equation in a square domain

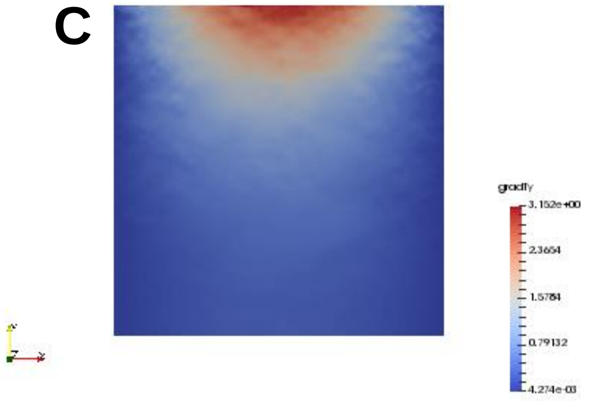
A

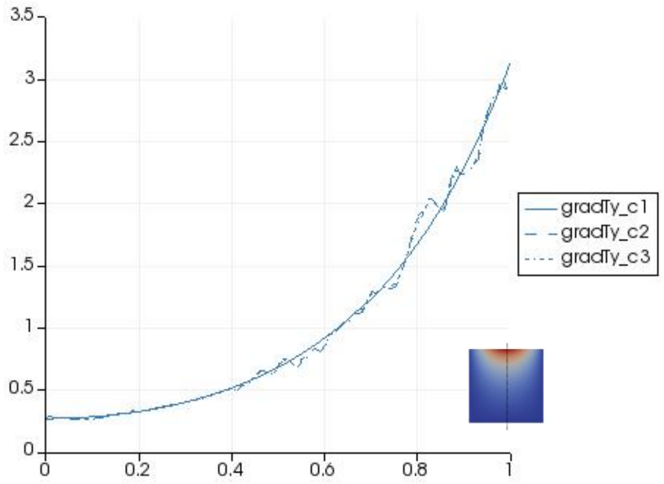


B



C





$\mathrm{grad}_y(T)$ **field**

gradSchemes:
Gauss linear

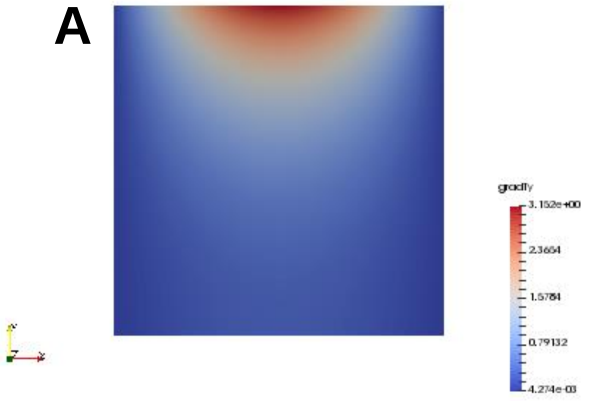
laplacianSchemes:
Gauss linear orthogonal

A. Hexahedral mesh
B. Triangular mesh
C. Polyhedral mesh

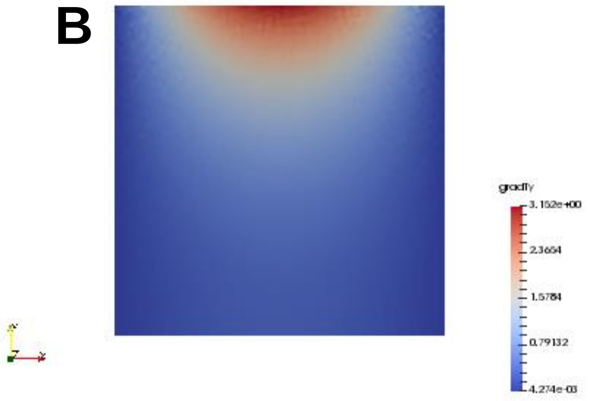
Numerical playground

2D Laplace equation in a square domain

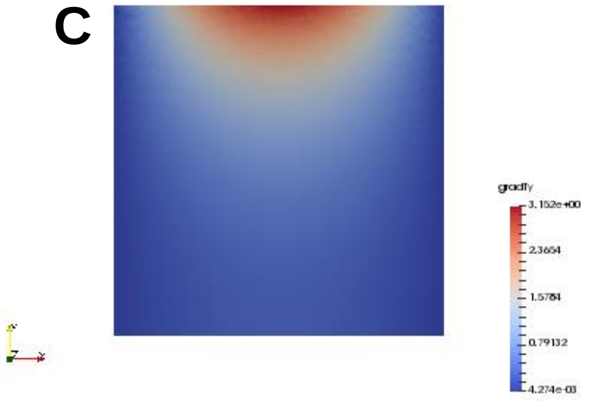
A

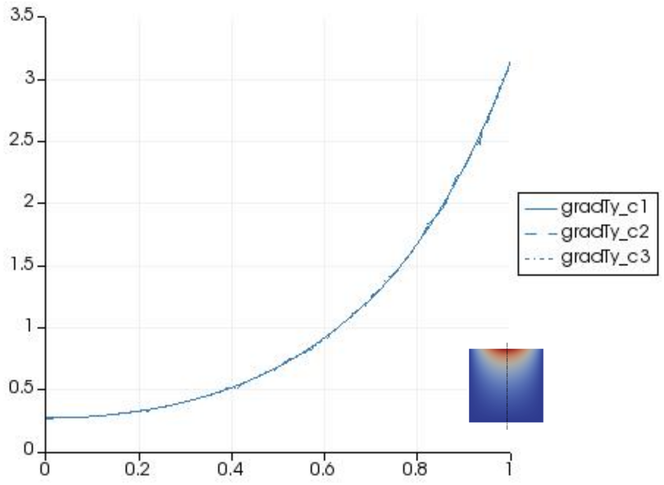


B



C





$\mathrm{grad}_y(T)$ **field**

gradSchemes:
Gauss linear

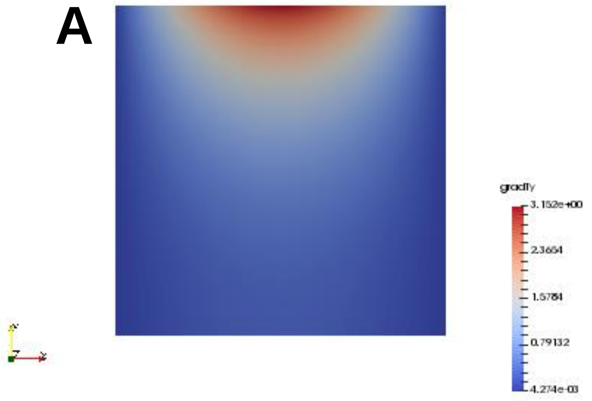
laplacianSchemes:
Gauss linear limited 1

A. Hexahedral mesh
B. Triangular mesh
C. Polyhedral mesh

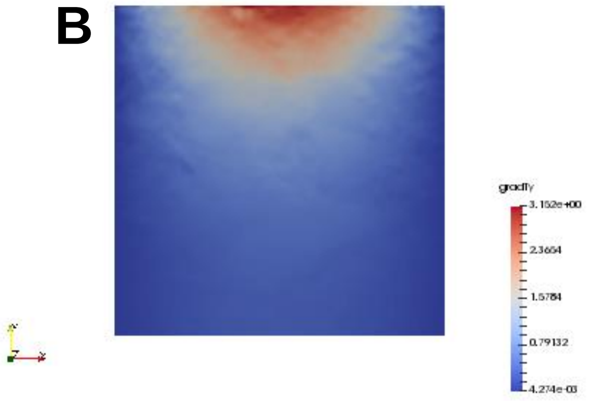
Numerical playground

2D Laplace equation in a square domain

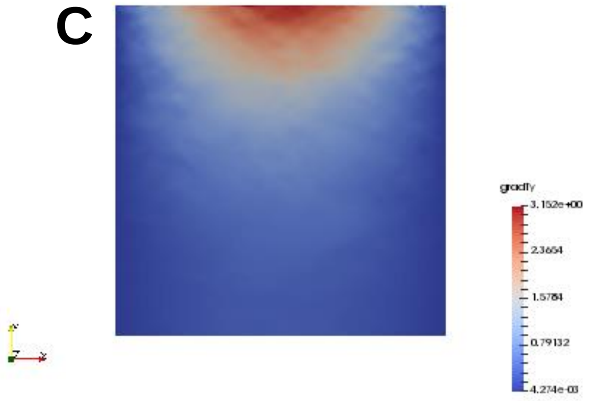
A

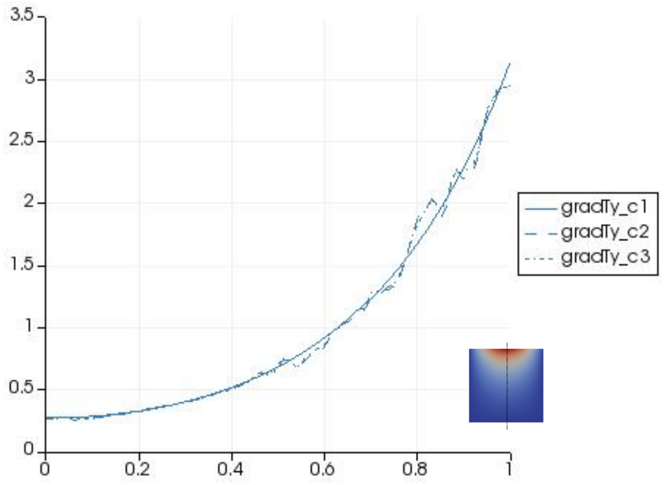


B



C





$\mathrm{grad}_y(T)$ **field**

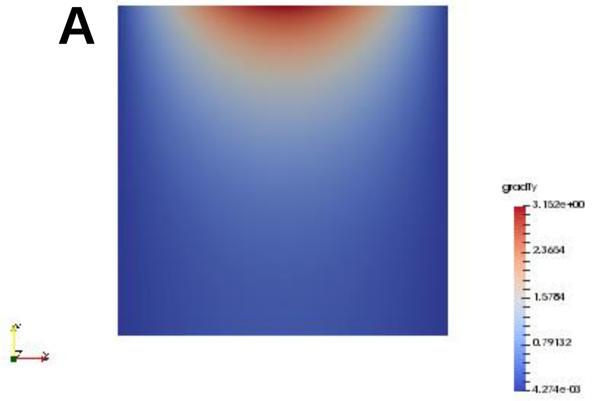
gradSchemes:
Gauss leastSquares

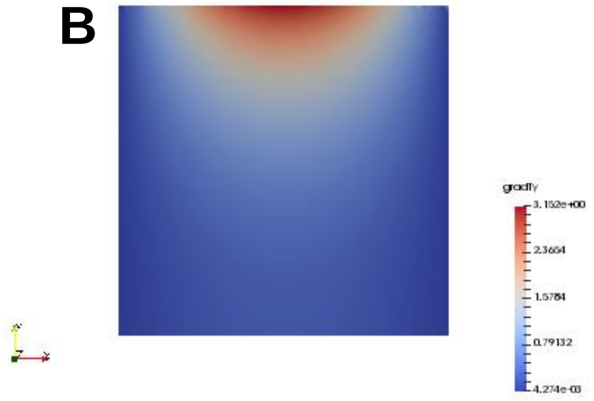
laplacianSchemes:
Gauss linear orthogonal

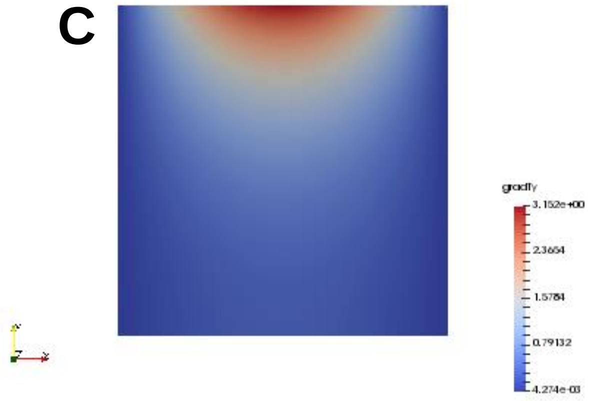
A. Hexahedral mesh
B. Triangular mesh
C. Polyhedral mesh

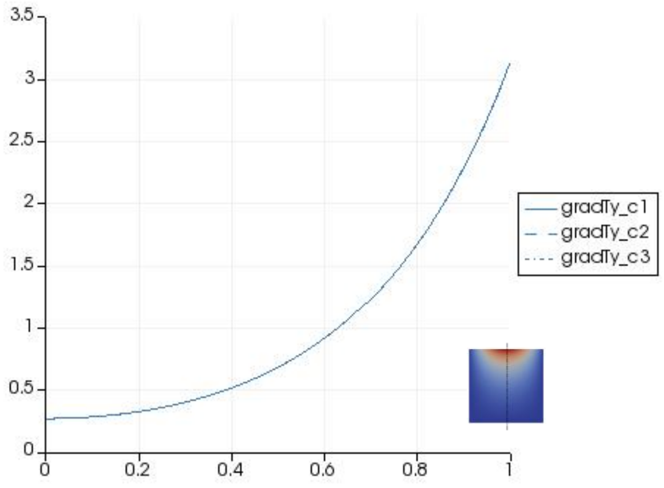
Numerical playground

2D Laplace equation in a square domain

A


B


C




$\text{grad}_y(T)$ **field**

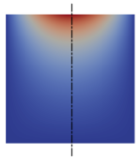
gradSchemes:
Gauss leastSquares

laplacianSchemes:
Gauss linear limited 1

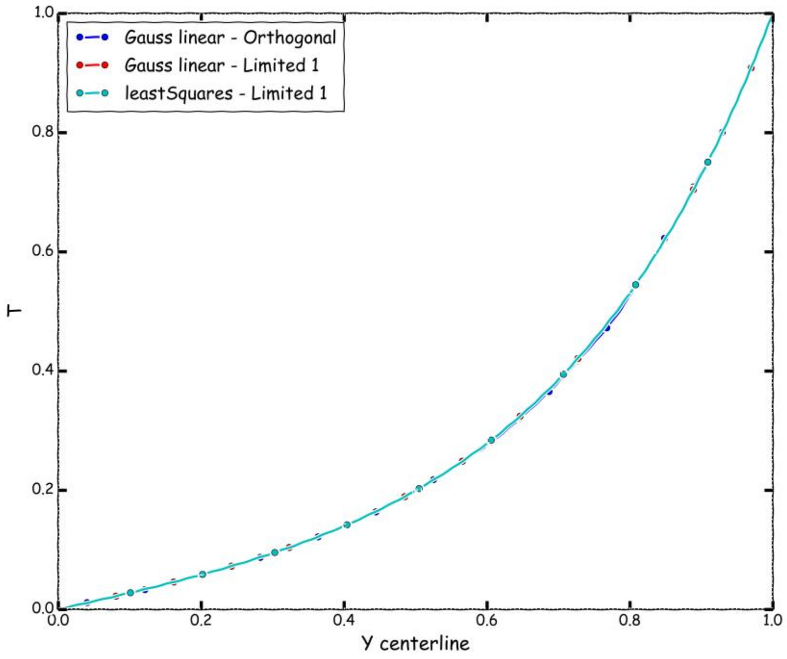
A. Hexahedral mesh
B. Triangular mesh
C. Polyhedral mesh

Numerical playground

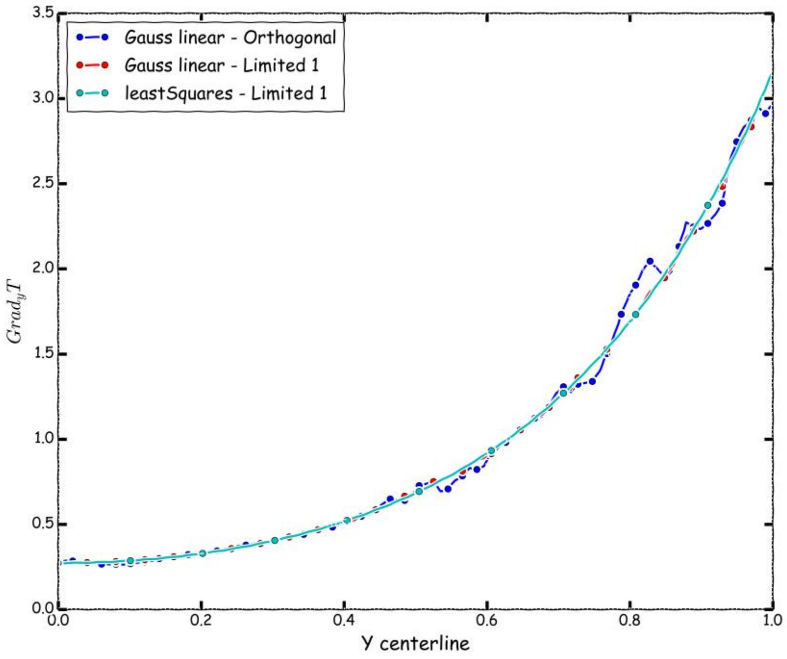
2D Laplace equation in a square domain



Sampling location – Polyhedral mesh



Scalar T



$\mathrm{grad}_y(T)$

Numerical playground

- This case was for your eyes and brain only, but we encourage you to reproduce all the previous results.
- In the subdirectory **`c1`** you will find the hexahedral mesh, in the subdirectory **`c2`** you will find the triangular mesh, and in the subdirectory **`c3`** you will find the polyhedral mesh.
- Use the script `runallcases.sh` to run all the cases automatically.
- When launching `paraFoam` it will give you a warning, accept the default option (yes).
- In `paraFoam`, go to the `File` menu and select `Load State`. Load the state located in the directory **`paraview`** (*`state1.pvsm`*).
- In the window that pops out, give the location of the *`*.foam`* files inside each subdirectory (*`c1/c1.foam`*, *`c2/c2.foam`*, and *`c3/c3.foam`*).
- The file *`state1.pvsm`* will load a preconfigured state with all the solutions.
- If you are interested in running the cases individually, enter the subdirectory and follow the instructions in the *`README.FIRST`* file.

Numerical playground

Exercises

- Run the case using all gradient discretization schemes available. Which scheme gives the best results?
- According to the previous results, which element type is the best one? Do you think that the choice of the element type is problem dependent (e.g., direction of the flow)?
- Use the **leastSquares** method for gradient discretization, and the **corrected** and **uncorrected** method for Laplacian discretization. Do you get the same results in all the meshes? How can you improve the results?

  (Hint: look at the corrections)
- Does it make sense to do more non-orthogonal corrections using the **uncorrected** method?
- Run a case only 1 iteration. Do you get a converged solution? Is there a difference between 1 and 100 iterations? Compare the solutions.
- Use a different interpolation method for the diffusion coefficient. Do you get the same results?
- Try to break the solver (this is a difficult task in this case). You are allow to modify the original mesh and use any combination of discretization schemes.

Numerical playground

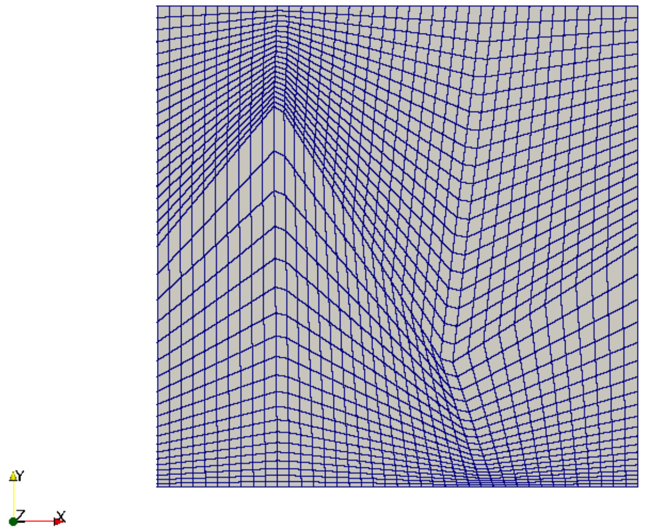
Swing:

Flow in a lid-driven square cavity – Re = 100

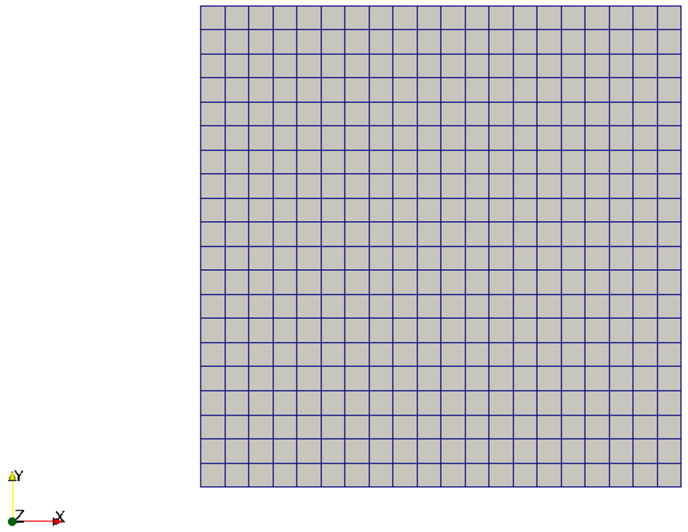
Effect of grading and non-orthogonality on the accuracy of the solution

Numerical playground

Flow in a lid-driven square cavity – Re = 100
Non-orthogonal mesh vs. orthogonal mesh



Non-orthogonal mesh
The overall quality of this mesh is good (in terms of non-orthogonality and skewness), <u>but by no standard this is a good mesh.</u>

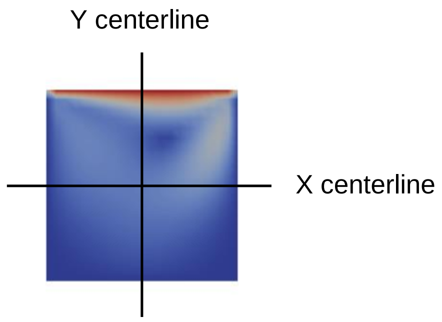


Orthogonal mesh
This is a perfect mesh

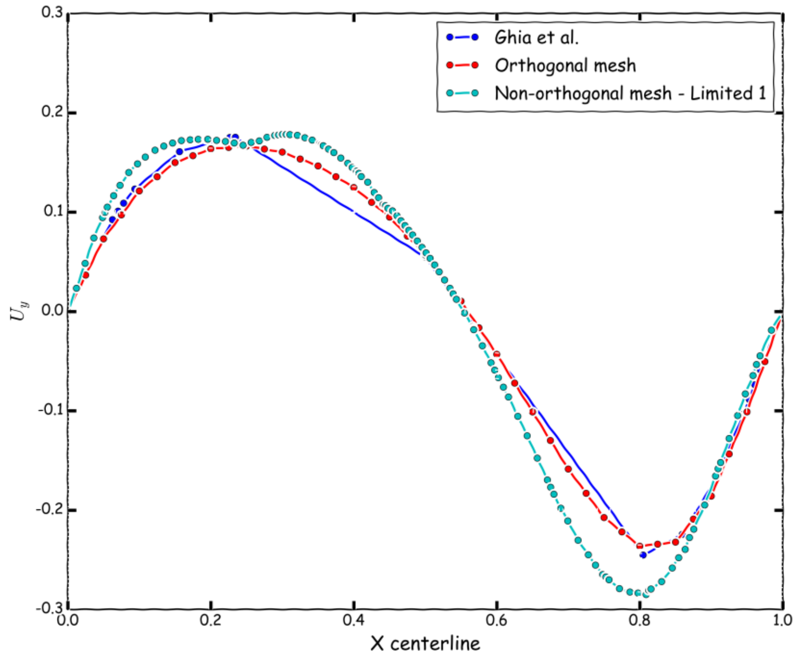
- We will use this case to learn how to adjust the numerical schemes according to mesh non-orthogonality and grading.

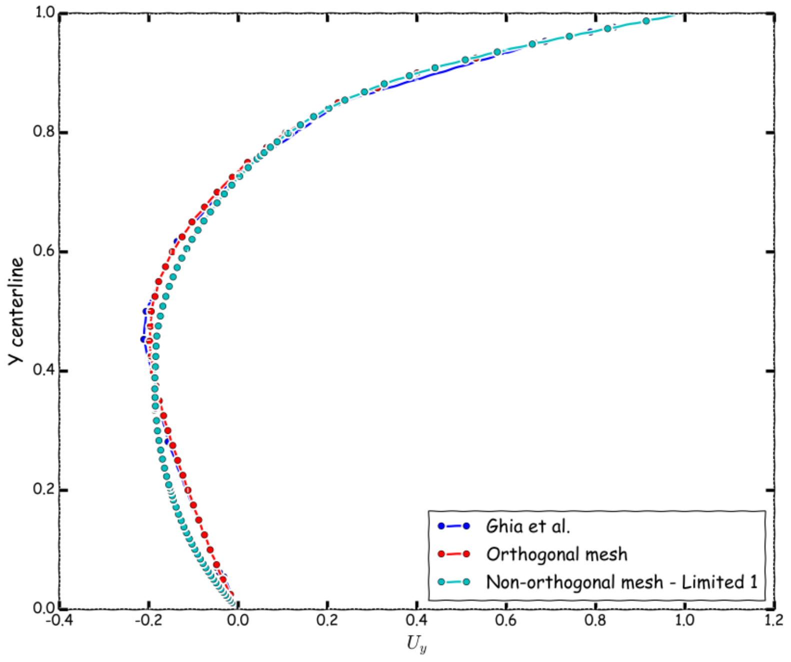
Numerical playground

LaplacianSchemes orthogonal – No corrections



- And as CFD is not only about pretty colors, we should also validate the results

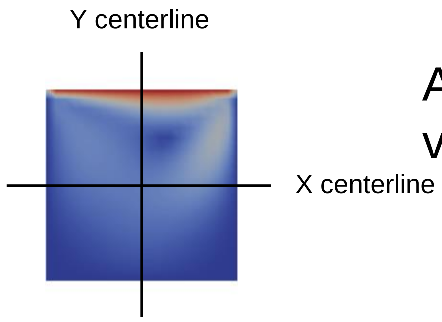




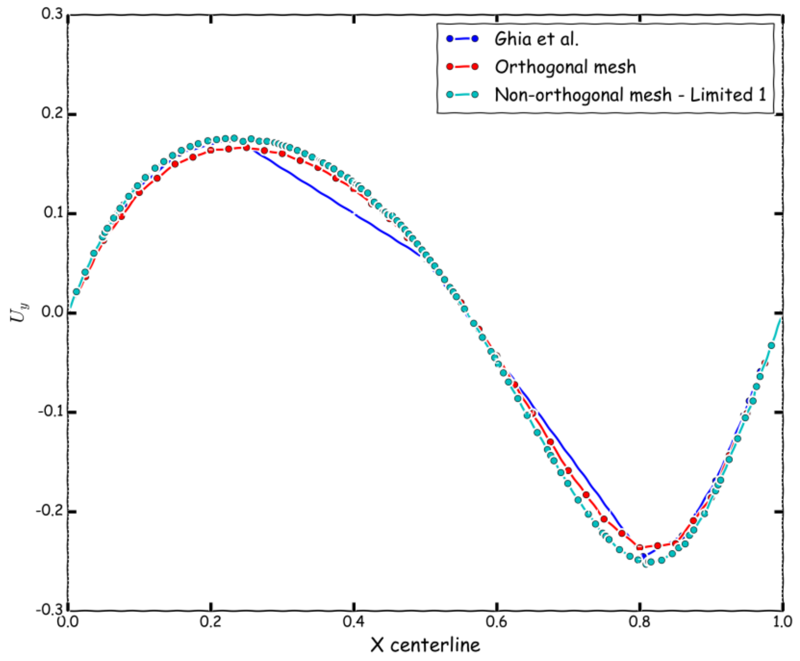
High-Re Solutions for incompressible flow using the navier-stokes equations and a multigrid method
U. Ghia, K. N. Ghia, C. T. Shin.
Journal of computational physics, 48, 387-411 (1982)

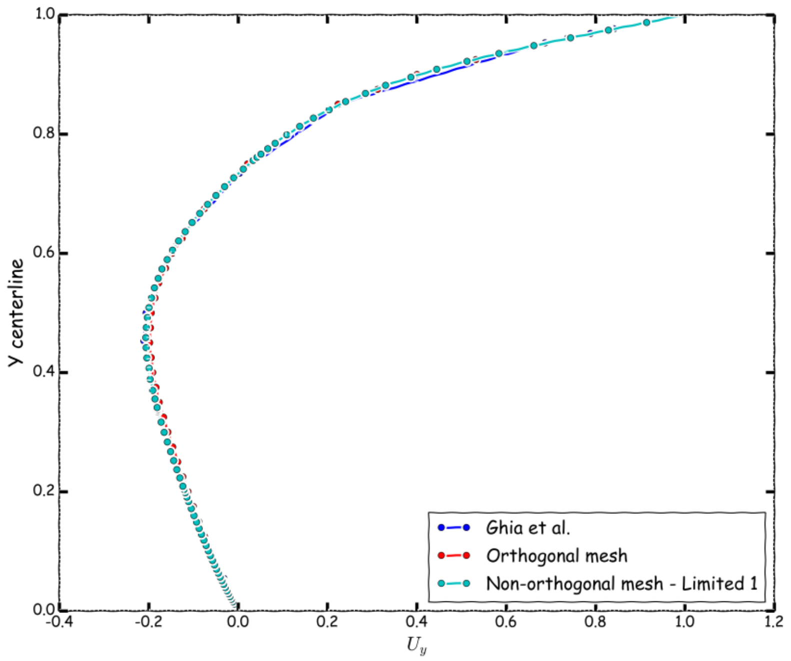
Numerical playground

LaplacianSchemes limited 1 – Non-orthogonal corrections



And as CFD is not only about pretty colors, we should also validate the results





High-Re Solutions for incompressible flow using the navier-stokes equations and a multigrid method
U. Ghia, K. N. Ghia, C. T. Shin.
Journal of computational physics, 48, 387-411 (1982)

Numerical playground

How to adjust the numerical method to deal with non-orthogonality

```
ddtSchemes
{
    default         backward;
}

gradSchemes
{
    default         Gauss linear;
    //default         cellMDLimited Gauss linear 1;

    grad(p)         Gauss linear;
}

divSchemes
{
    default         none;
    //div(phi,U)      Gauss linearUpwind default;
    div(phi,U)      Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear orthogonal;
    //default         Gauss linear limited 1;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;
    //default         limited 1;
}
```

- In the dictionary *fvSchemes* we can enable non-orthogonal corrections.
- Non-orthogonal corrections are chosen using the keywords **laplacianSchemes** and **snGradSchemes**.
- These are the **laplacianSchemes** and **snGradSchemes** schemes that you will use most of the times:
    - **orthogonal:** second order accurate, bounded on perfect meshes, without non-orthogonal corrections.
    - **corrected:** second order accurate, bounded depending on the quality of the mesh, with non-orthogonal corrections.
    - **limited** ψ**:** second order accurate, bounded depending on the quality of the mesh, with non-orthogonal corrections.
    - **uncorrected:** second order accurate, without non-orthogonal corrections. Stable but more diffusive than limited and corrected.

Numerical playground

How to adjust the numerical method to deal with non-orthogonality

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }
}

PISO
{
    nCorrectors     1;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

- Additionally, in the dictionary *`fvSolution`* we need to define the number of **PISO** corrections (**nCorrectors**) and non-orthogonal corrections (**nNonOrthogonalCorrectors**).
- You need to do at least one **PISO** correction. Increasing the number of **PISO** correctors will improve the stability and accuracy of the solution at a higher computational cost.
- For orthogonal meshes, 1 **PISO** correction is ok. But as most of the time you will deal with non-orthogonal meshes, doing 2 **PISO** corrections is a good choice.
- If you are using a method with non-orthogonal corrections (**corrected** or **limited 1-0.5**), you need to define the number of non-orthogonal corrections (**nNonOrthogonalCorrectors**).
- If you use 0 **nNonOrthogonalCorrectors**, you are computing the initial approximation using central differences (accurate but unstable), with no explicit correction.
- To take into account the non-orthogonality of the mesh, you will need to increase the number of corrections (you get better approximations using the previous correction).
- Usually 2 **nNonOrthogonalCorrectors** is ok.

Numerical playground

- We will now illustrate a few of the discretization schemes available in OpenFOAM® using a model case.
- We will use the lid-driven square cavity case to study the effect of grading and non-orthogonality on the accuracy of the solution
- This case is located in the directory:

```
$PTOFC/101FVM/nonorthoCavity/
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_solver`.
- We highly recommend to open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM® commands.
- If you are already comfortable with OpenFOAM®, use the automatic scripts to run the cases.

Numerical playground

What are we going to do?

- This is the same case as the one we used during the first tutorial session.
- The only difference is that we have modified the mesh a little bit in order to add grading and non-orthogonality.
- After generating the mesh, we will use the utility `checkMesh` to control the quality of the mesh. Is it a good mesh?
- We will use this case to learn how to adjust the numerical schemes according to mesh non-orthogonality and grading.
- To find the solution we will use `icoFoam`.
- After finding the numerical solution we will do some sampling.
- At the end, we will do some plotting (using gnuplot or Python) and scientific visualization.

Numerical playground

Running the case

- You will find this tutorial in the directory **`$PTOFC/101FVM/nonorthoCavity`**
- In the terminal window type:

```
1. $> foamCleanTutorials
2. $> blockMesh
3. $> checkMesh
4. $> icoFoam | log
5. $> postProcess -func sampleDict -latestTime
6. $> gnuplot gnuplot/gnuplot_script
7. $> paraFoam
```

Numerical playground

To run the case, follow these steps

- First run the case using the original dictionaries. Did it crash right?
- Now change the **laplacianSchemes** and **snGradSchemes** to **limited 1**. It crashed again but this time it ran a few more time-steps, right?
- Now increase the number of **nNonOrthogonalCorrectors** to 2. It crashed again but it is running more time-steps, right?
- Now increase the number of **PISO** corrections to 2 (**nCorrectors**). Did it run?
- Basically we enabled non-orthogonal corrections, we computed better approximations of the gradients, and we increased the number of **PISO** corrections to get better predictions of the field variables (**U** and **p**).
- Now set the number of **nNonOrthogonalCorrectors** to 0. Did it crash right? This is telling us that the mesh is sensitive to the gradients.
- Now change the **laplacianSchemes** and **snGradSchemes** to **limited 0** (uncorrected). In this case we are not using non-orthogonal corrections, therefore there is no need to increase the value of **nNonOrthogonalCorrectors**.
- We are using a method that uses a wider stencil to compute the Laplacian, this method is more stable but a little bit more diffusive. Did it run?
- At this point, compare the solution obtained with corrected and uncorrected schemes. Which one is more diffusive?

Numerical playground

- When it comes to **laplacianSchemes** and **snGradSchemes** this is how we proceed most of the times (a robust setup),

```
laplacianSchemes
{
        default        Gauss linear limited 1;
}

snGradSchemes
{
        default        limited 1;
}
```

```
PISO
{
        nCorrectors      2;
        nNonOrthogonalCorrectors 1;
}
```

- This method works fine for meshes with non-orthogonality less than 75.
- If the non-orthogonality is more than 75, you should consider using **limited 0.5**, and increasing **nCorrectors** and **nNonOrthogonalCorrectors**.
- When the non-orthogonality is more than 85, the best solution is to redo the mesh.

Numerical playground

Exercises

- Using the non-orthogonal mesh and the original dictionaries, try to run the solver reducing the time-step. Do you get a solution at all?
- Try to get a solution using the method **limited 1** and two **nNonOrthogonalCorrectors** (leave **nCorrectors** equal to 1).

  (Hint: try to reduce the time-step)
- If you managed to get a solution using the previous numerical scheme. How long did it take to get the solution? Use the robust setup, clock the time and compare with the previous case. Which one is faster? Do you get the same solution?
- Instead of using the non-orthogonal mesh, use a mesh with grading toward all edges. How will you stabilize the solution?

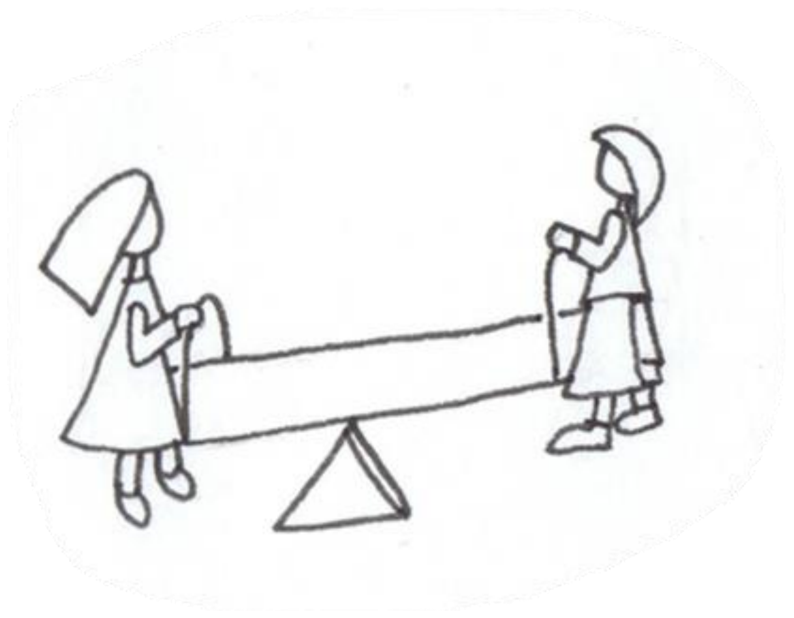
  (Hint: take a look at the blockMesh slides in order to add grading to the mesh)
- Try to get a solution using a time-step of 0.05 seconds. Use the original discretization schemes for the gradient and convective terms.

  (Hint: increase nCorrectors and nNonOrthogonalCorrectors)
- Try to break the solver and interpret the output screen. You are allow to modify the original mesh and use any combination of discretization schemes.

Numerical playground

Seesaw:

Sod’s shock tube.



Numerical playground

Sod's shock tube

- This case has an analytical solution and plenty of experimental data.
- This is an extreme test case used to test solvers.
- Every single commercial and open source solver use this case for validation of the numerical schemes.
- The governing equation of this test case are the Euler equations.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0$$

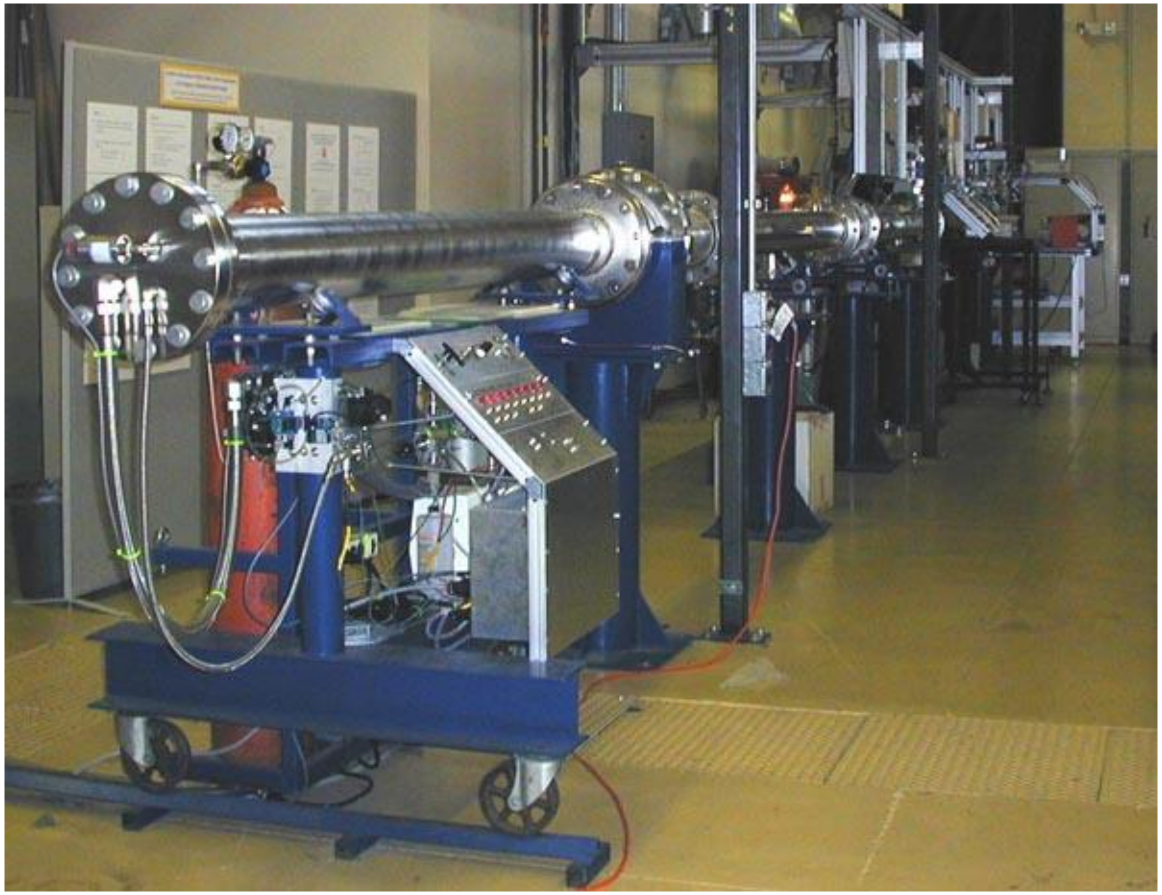
$$\frac{\partial (\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U} \mathbf{U}) + \nabla p = 0$$

$$\frac{\partial (\rho e_t)}{\partial t} + \nabla \cdot (\rho e_t \mathbf{U}) + \nabla \cdot (p \mathbf{U}) = 0$$

$$p = \rho R_g T$$

Numerical playground

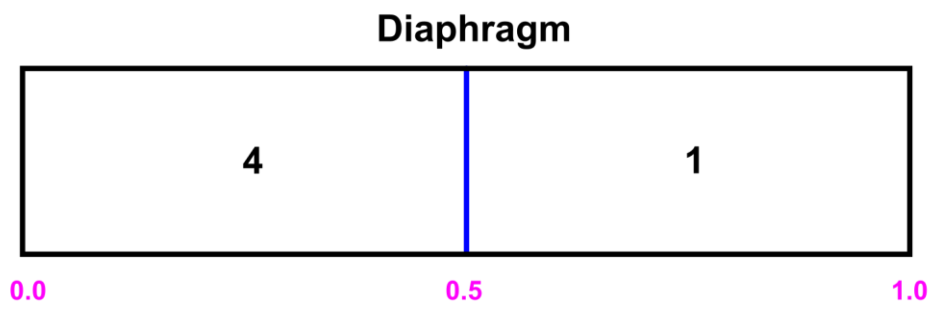
High Purity Photolysis Shock Tube (NASA Tube)



Shock tube. The driver section, including vacuum pumps, controls, and helium driver gas.
Photo credit: Stanford University. http://hanson.stanford.edu/index.php?loc=facilities_nasa

Numerical playground

Sod's shock tube



All walls are slip

$$\mathbf{U_4} = \mathbf{U_1} = 0$$

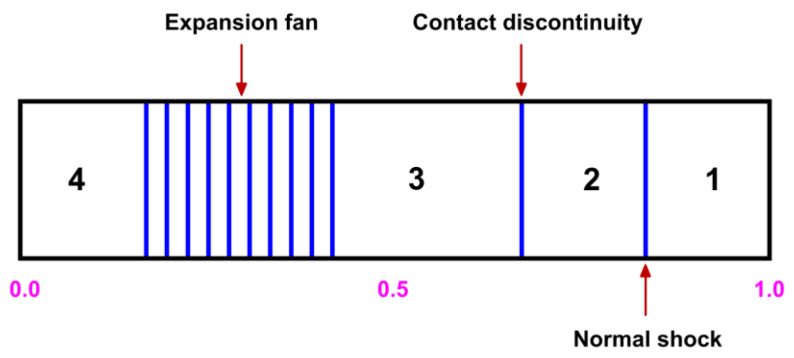
$$p_4 = 1, \quad p_1 = 0.1$$

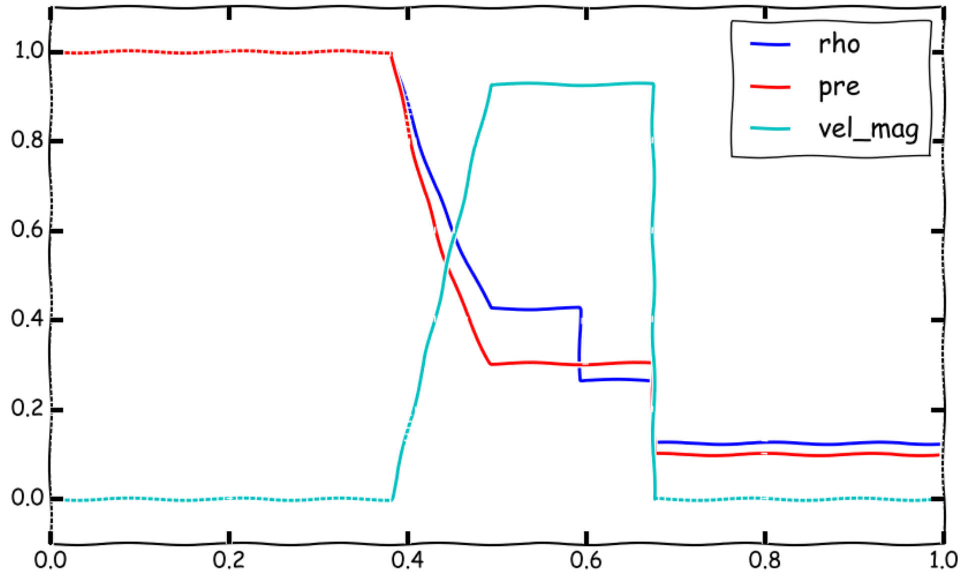
$$T_4 = 0.00348, \quad T_1 = 0.00278$$

Boundary conditions and initial conditions

Numerical playground

Sod's shock tube



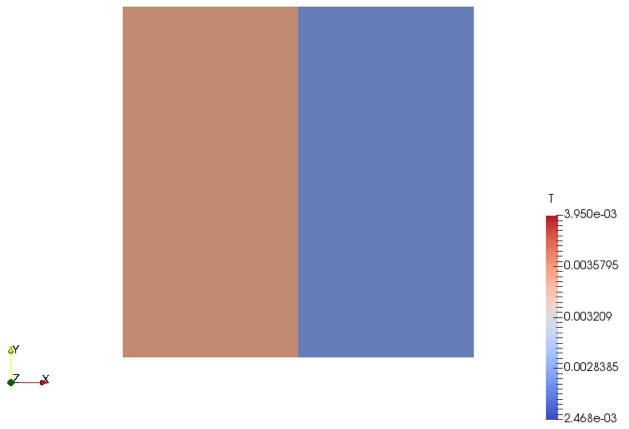


Analytical solution

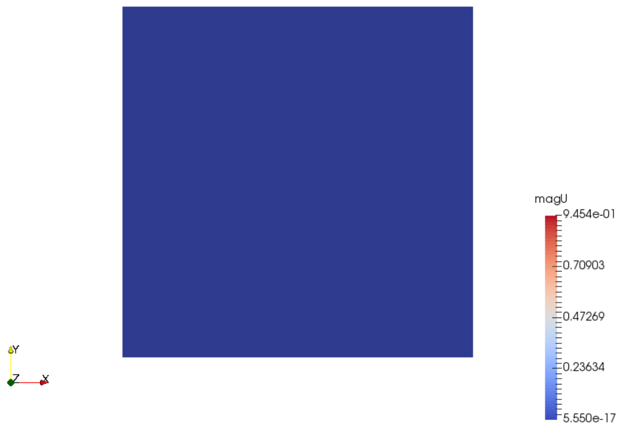
Numerical playground

Sod’s shock tube

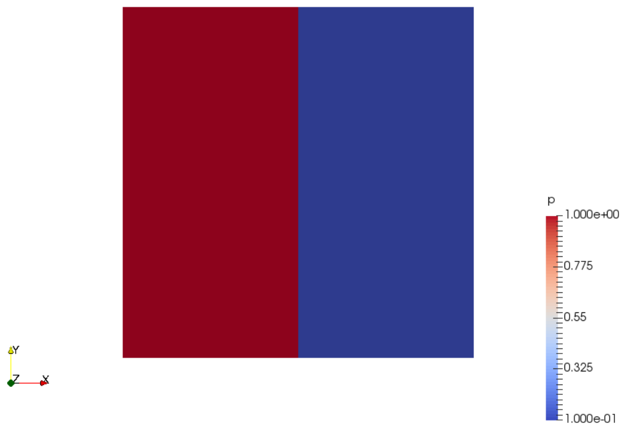
www.wolfdynamics.com/wiki/shocktube/aniT.gif



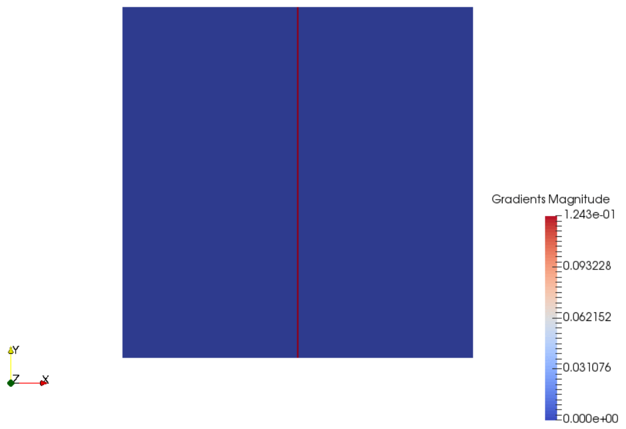
www.wolfdynamics.com/wiki/shocktube/aniU.gif



www.wolfdynamics.com/wiki/shocktube/anip.gif

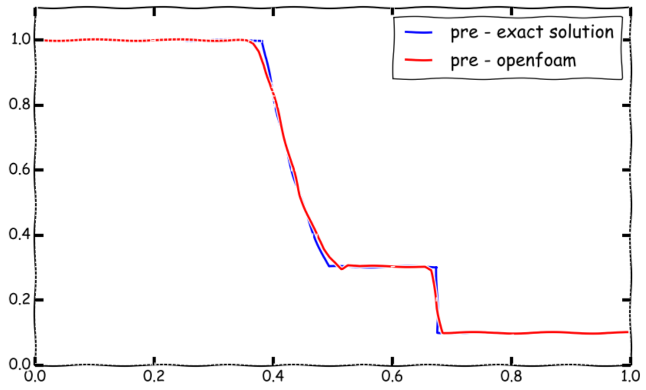


www.wolfdynamics.com/wiki/shocktube/anigt.gif

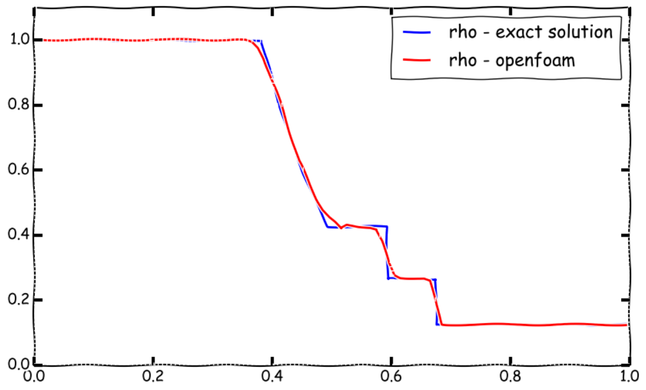


Numerical playground

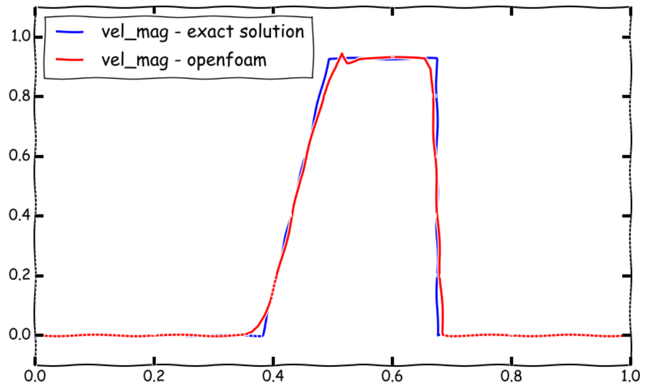
Sod’s shock tube



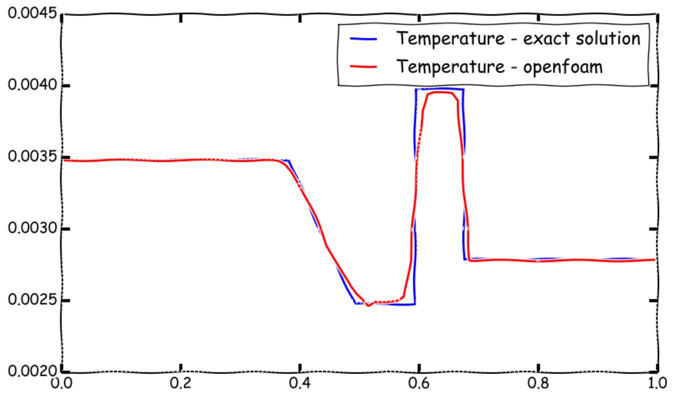
Pressure field



Density field



Velocity magnitude field



Temperature field

Numerical playground

- We will now illustrate a few of the discretization schemes available in OpenFOAM® using a severe model case.
- We will use the Sod's shock tube case.
- This case is located in the directory:

```
$PTOFC/101FVM/shockTube/
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_solver`.
- We highly recommend to open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM® commands.
- If you are already comfortable with OpenFOAM®, use the automatic scripts to run the cases.

Numerical playground

What are we going to do?

- Now is your turn.
- You are asked to select the best discretization scheme for the physics involve. Remember: accuracy, stability and boundedness.
- We will compare your numerical solution with the analytical solution.
- At this point, we are very familiar with the numerical schemes. It is up to you to choose the best setup.
- You can start using the original dictionaries.
- To find the numerical solution we will use `sonicFoam`.
- `sonicFoam` is a transient solver for trans-sonic/supersonic, laminar or turbulent flow of a compressible gas.
- After finding the numerical solution we will do some sampling.
- At the end, we will do some plotting (using gnuplot or Python) and scientific visualization.

Numerical playground

Running the case

- You will find this tutorial in the directory **`$PTOFC/101FVM/schockTube`**
- Before running the case, you will need to choose the discretization scheme.
- In the terminal window type:

```
1.  $> foamCleanTutorials
2.  $> blockMesh
3.  $> checkMesh
4.  $> rm -rf 0
5.  $> cp -r 0_org 0
6.  $> setFields
7.  $> sonicFoam | tee log
8.  $> foamCalc mag U
9.  $> postProcess -func sampleDict -latestTime
10. $> paraFoam
```

Numerical playground

Running the case

- In step 3 we generate the mesh using `blockMesh`.
- In step 5 and 6 we copy the original files to the directory **0**. We do this to keep a backup of the original files as the file *`0/U`* will be overwritten.
- In step 7 we initialize the solution using `setFields`.
- In step 8 we run the simulation and save the log file.
- In step 9 we use the utility `foamCalc` to compute the magnitude of the velocity vector **U**. The output is saved as **magU**.
- In step 10 we use the utility `postProcess` to do sampling of the field variables.
- Finally, in step 11 we visualize the solution using `paraFoam`.

Numerical playground

Running the case

- To plot the analytical solution against the numerical solution, go to the directory **`python`** and run the Python script.
- In the terminal window type:

```
1. | $> cd python
2. | $> python sodschocktube.py
```

- The Python script will save four .png files with the solution.
- Feel free to explore and adapt the Python script to your needs.
- Remember, Python must be installed in order to use the script.
- We use Anaconda Python 2.7

Numerical playground

Exercises

- Run the case using different time discretization schemes.
- Run the case using different gradient discretization schemes.
- Run the case using different convective discretization schemes for the term **div(phi,U)**.
- Run the case using different convective discretization schemes for the terms **div(phi,e)** and **div(phi,K)**. What are the variables **e** and **K**?
- Extend the case to 2D and 3D. Do you get the same solution?
- Try to run a 2D case using a triangular mesh and adjust the numerical scheme to get an accurate and stable solution.
- Try to run the 1D case using an explicit solver. For the same CFL number, do you have the same time step size as for the implicit solver?

  (Hint: look for the solver with the word Central)
- Try to break the solver (this is extremely easy in this case). You are allow to modify the original mesh and use any combination of discretization schemes.

Some FVM/CFD references

- As we mentioned earlier this is not a FVM/CFD course, but we highly advise you to take some time and study the theory in depth.
- There is vast amount of literature in the field of FVM/CFD. We will give you some of our favorite references, which are closed related to what you will find in OpenFOAM®.
- Therefore, we are involuntarily omitting other references which are equally or even more important.
    - **The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction With OpenFOAM and Matlab**
      F. Moukalled, L. Mangani, M. Darwish. 2015, Springer-Verlag
    - **Finite Volume Methods for Hyperbolic Problems**
      R. Leveque. 2002, Cambridge University Press
    - **Computational Gasdynamics**
      C. Laney. 1998, Cambridge University Press
    - **Computational Techniques for Multiphase Flows**
      G. H. Yeoh, J. Tu. 2009, Butterworth-Heinemann
    - **An Introduction to Computational Fluid Dynamics**
      H. K. Versteeg, W. Malalasekera. 2007, Prentice Hall
    - **Computational Fluid Dynamics: Principles and Applications**
      J. Blazek. 2006, Elsevier Science
    - **Computational Methods for Fluid Dynamics**
      J. H. Ferziger, M. Peric. 2001, Springer
    - **Numerical Heat Transfer and Fluid Flow**
      S. Patankar. 1980, Taylor & Francis
    - **A Finite Volume Method for the Prediction of Three-Dimensional Fluid Flow in Complex Ducts**
      M. Peric. PhD Thesis. 1985. Imperial College, London
    - **Error analysis and estimation in the Finite Volume method with applications to fluid flows**
      H. Jasak. PhD Thesis. 1996. Imperial College, London
    - **Computational fluid dynamics of dispersed two-phase flows at high phase fractions**
      H. Rusche. PhD Thesis. 2002. Imperial College, London
    - **High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws**
      P. K. Sweby SIAM Journal on Numerical Analysis, Vol. 21, No. 5. (Oct., 1984), pp. 995-1011