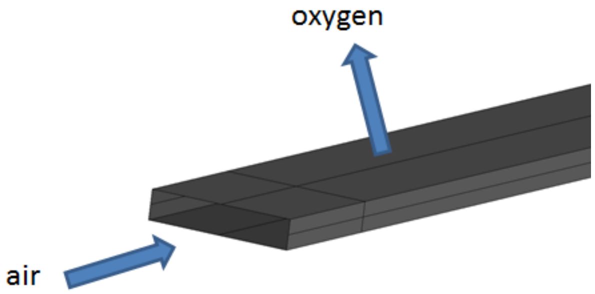
Supplementary document for the journal paper
An engineering toolbox for the evaluation of metallic flow field plates

An implicit model for a gas mixture
(Calculating mass fraction as a field variable)



Uwe Reimer and Steven Beale
Forschungszentrum Jülich GmbH, Institute of Energy and Climate Research,
IEK-3: Electrochemical Process Engineering, 52425 Jülich/ Germany

Contents

1 Introduction

This tutorial shows how to use OpenFOAM to solve the following problem: Removing one component of a mixture through a membrane. The examples are tested with OpenFOAM version 3.0.1. The gas mixture is modeled as a homogeneous medium that contains two components as shown in Figure 1. Both components do have identical properties (like density). The content of the second component (red balls in Figure 1) is described by the mass fraction y.

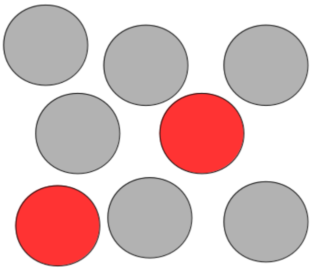


Figure 1: Gas mixture with two components.

1.1 Definition of the problem

The underlying physical model describes a gas mixture. The mixture is modeled as an effective fluid (air) that contains one special component (oxygen). The distribution of this special component is defined by its mass fraction y.

This example case shall simulate the flow of air through a channel, where one component (oxygen) is partly removed through a membrane. The membrane shall be one side wall of the channel. As a matter of simplicity air is treated as a mixture of 79 % nitrogen and 21 % oxygen. Both oxygen and nitrogen shall have the same average properties as air[1] (see Table 1).

Table 1: Physical properties of air at 300 K and 101326 Pa

density	ρ	$1177\,\mathrm{g\,m^{-3}}$
kinematic viscosity	ν	$15 \cdot 10^{-6}\,\mathrm{m^2\,s^{-1}}$
diffusion coefficient	D	$176 \cdot 10^{-7}\,\mathrm{m^2\,s^{-1}}$

A typical mass flow through a channel in a fuel cell would be $31.15\,\mathrm{g\,h^{-1}}$ of air. (This example calculation is based on a three channel serpentine flow field with an activ area of $45\,\mathrm{cm^2}$. The average current density is $0.4\,\mathrm{A\,cm^{-2}}$ and the cell is operated with air at an stoichiometry of 2.) Since the cross section is $1\,\mathrm{mm^2}$, the velocity at the inlet is $u = 1.47\,\mathrm{m\,s^{-1}}$. With these values the Reynolds number can be calculated according Equation 1 ($d = 0.001667\,\mathrm{m}$).

$$\mathrm{Re} = \frac{d\,u}{\nu} \tag{1}$$

---

[1] https://en.wikipedia.org/wiki/Mass_diffusivity

From the resulting value of Re = 163 a laminar flow regime can be expected.

1.2 Size of membrane and diffusion limitation

As oxygen is removed through a membrane, a concentration gradient between the surface of the membrane and the fluid part of the channel is created. This gradient causes a diffusion flux of oxygen from the channel towards the membrane. The diffusion coefficient in Table 1 defines an upper limit for this flux. The simulation case should stay below the diffusion limit. Therefore, the minimum size of the membrane area is estimated as a first step based on the following considerations.

1. Definition: 50 % of oxygen shall be removed. The height of the channel shall be $d = 1 \cdot 10^{-3}$ m.
2. Calculate the maximum diffusion flux (velocity) from Fick's Law.
3. Calculate the required area of the membrane based on the maximum velocity and choose the simulation geometry.

Maximum velocity
The maximum diffusion flux can be calculated by using Fick's Law (Equation 2). For a simple estimation the conditions at the outlet of the channel are considered, where the mass fraction of oxygen should be $Y_{\text{bulk}} = 0.5 \cdot 0.21$. The maximum flux will occur if the concentration at the surface becomes zero, i. e. $Y_{\text{surf}} = 0$.

$$u_{\text{max}} = D \cdot \frac{(Y_{\text{bulk}} - Y_{\text{surf}})}{d} \quad = 176 \cdot 10^{-7} \frac{0.5 \cdot 0.21}{1 \cdot 10^{-3}} \quad = 1.85 \cdot 10^{-3}\,\text{m/s} \tag{2}$$

Size of membrane
The area A of the membrane can be calculated according Equation 3. Here, $\dot{m} = 908 \cdot 10^{-6}$ g/s is the total mass flux of oxygen through the membrane (50 % of the initial content in air).

$$A = \frac{\dot{m}}{\rho \cdot u_{\text{max}}} \quad = 417 \cdot 10^{-6}\,\text{m}^2 \tag{3}$$

In order to resemble a straight channel, the following geometry is chosen (see Table 2). The width was chosen to be larger than the channel heigth, because this way it is easier to plot the resulting graphs for this tutorial. At inlet and outlet of the channel an additional section is added, nameley *inlet relaxation zone* and *outlet relaxation zone*. These zones are required because in CFD simuations usually a flat velocity profile is defined at the inlet and a constant pressure at the outlet. The typical flow profile will be fully developed after a short distance behind the inlet region. The resulting channel has a membrane area of $A = 425 \cdot 10^{-6}\,\text{m}^2$.

1.3 Boundary conditions

The inlet velocity of air can be calculated from the mass flow of air and the channel geometry in Table 2. A value of $u = 1.469$ m/s results. The mass fraction of oxygen at the inlet is $Y = 0.21$. Pure oxygen shall be removed through the membrane at a constant velocity. The velocity for the boundary condition of the membrane outlet is given in Table 3.

Table 2: Channel dimensions

width	5 mm
height	1 mm
length	85 mm
inlet relaxation zone	5 mm
outlet relaxation zone	5 mm

Table 3: Oxygen velocity for the membrane boundary

fraction of oxygen to be removed	velocity u / $\mathrm{m\,s^{-1}}$
0.1	3.63E-04
0.2	7.26E-04
0.3	1.09E-03
0.4	1.45E-03
0.5	1.82E-03
0.9	3.27E-03

1.4 Mesh

For geometry and mesh generation the software SALOME is used. The Python-script *OneChannel_v2_2.py* was used, which is contained in the documentation in the folder *b1/mesh/*. A cross section is shown in Figure 2.

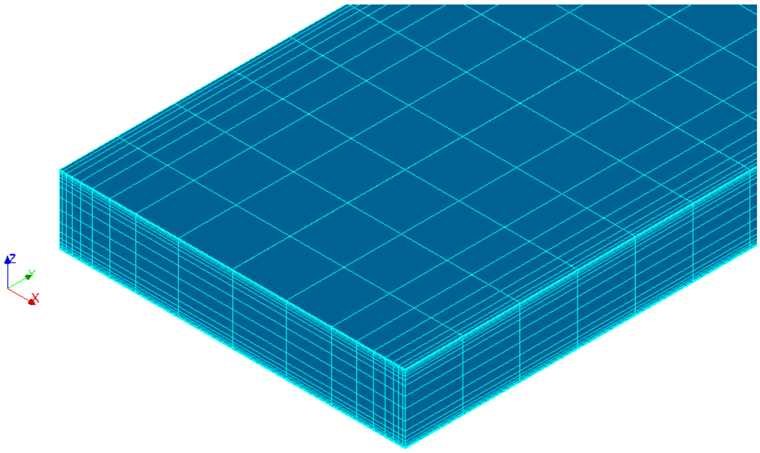


Figure 2: Section of channel with mesh resolution

2 Tutorial case b1: How to add mass fractions to solver icoFoam

This tutorial is based on the OpenFOAM solver 'icoFoam'. The idea is a modified version of a similar article entitled *How to add temperature*[2]. The main differences are that the normal wall velocity is not zero at the patch corresponding to the membrane wall. The main items to be accomplished are first, to copy and test that the local installation of OpenFOAM can compiles correctly. Once that is accomplished, various small changes are necessary to the solver files themselves which are detailed below. Finally, new fields have to be added to the initial and boundary conditions in '/0' and some alterations to the systems files made.

2.1 Copy and recompile icoFoam

This will go through the steps of creating a personal version of icoFoam in the user's subdirectory. The first step is to insure that your installation of OpenFOAM works properly and compiles the unedited solver. First, bring up a console and move to your OpenFOAM installation folder. Your particular OpenFOAM installation folder will have a version number following. OpenFOAM has organized the solvers separate from the source code of OpenFOAM calling them 'applications'. Inside the 'applications' folder, there are subdirectories including one for solvers. This tutorial is based on icoFoam, which we will copy to our own location:

```
/usr/group/OpenFOAM-3.0.1/applications/solvers/incompressible
```

Copy icoFoam directory locally and rename it.

```
/private/u.reimer/mysource/micoFoam
```

Now, some alterations need to be made to the make files in order for everything to compile and not overwrite the original solver. First, rename the primary file to your new solver name and delete the old dependency file. Now go into the Make subdirectory and open *files* with an editor. Change it to read:

```
micoFoam.C
EXE = /private/u.reimer/mybin/micoFoam
```

Now, test that the renamed solver (and your installation of OpenFOAM) works:

```
wmake
```

If everything worked correctly, your new solver binary should appear in the
/private/u.reimer/mybin/micoFoam directory.

2.2 Create the mass fraction class

Open the *micoFoam.C* with a text editor. First, edit the *Application* at the top of the file to reflect the new name. Following the flow of the program, one notices that the header file *createField.H* is called prior to the solution loop. This file was copied with the solver and has the specific information pertaining to what variables will be solved. Open *createFields.H* in your editor. The first items loaded is the kinematic viscosity from the transportProperties dictionary file. We will add a new transport

---

[2] https://openfoamwiki.net/index.php/How_to_add_temperature_to_icoFoam

property related to the mass fraction diffusion coefficient. We will store this whole-field, so that we can change the boundary values, by adding the class *diff* as follows:

```
dimensionedScalar nu
(
    "nu",
    dimViscosity,
    transportProperties.lookup("nu")
);

// s.beale
// Add store diffusion coefficient field-wise
volScalarField diff
(
        IOobject
        (
                "diff",
                runTime.timeName(),
                mesh,
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
        ),
        mesh
);
// s.beale
```

Later on, we will need to add a *diff* file in */run/0* to initialise. Following this in the file there are lines which pertain to the creation of the pressure p and velocity U fields. We will add a new field for mass fraction y. The fastest way to do this is to copy and paste the pressure lines and then edit them appropriately like so:

```
Info<< "Reading␣field␣U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

// s.beale
// Adding mass fraction as a field variable
Info<< "Reading␣field␣y\n" << endl;
volScalarField y
(
        IOobject
        (
                "y",
                runTime.timeName(),
                    mesh,
                    IOobject::MUST_READ,
                    IOobject::AUTO_WRITE
          ),
          mesh
);
// s.beale
```

Save the changes. You have now created a new field variable *y* with variable diffusion coefficient *diff*.

2.3 Solve the mass fractions equation

The next step is to add a new equation describing the transport of the species. Return to editing *micoFoam.C*. Because the mass fraction transport depends on the velocity field, we will add the equation after the momentum equation is solved (after the PISO loop), but before the time step is written. Edit your file so it looks like this:

```
               #include "continuityErrs.H"

               U = HbyA - rAU*fvc::grad(p);
               U.correctBoundaryConditions();
          }

                    // s.beale
                    //Add these lines to solve for y
                    fvScalarMatrix yEqn
                    (
                              fvm::ddt(y)
                              + fvm::div(phi, y)
                              - fvm::laplacian(diff, y)
                    );
                    yEqn.solve();
                    y.correctBoundaryConditions();
                    // s.beale

          runTime.write();
```

These lines add a new equation for the mass fraction and make use of the face flux variable, phi, which is already used in the momentum equation solution. Save the changes and run wmake.

```
wmake
```

Your computer should then produce a newly compiled binary.

2.4 Add new files for initial and boundary conditions

The directory structure and files are provided in the folder 'b1'.
Files to modify:

- */system/fvSchemes*
- */system/fvSolution*

Files to create new:

- */0/y*
- */0/diff*

2.4.1 Directory /system

file */system/fvSchemes*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss limitedLinearV 1;
        div(phi,y)              Gauss upwind; //s.beale: Add this
}

laplacianSchemes
{
    default         none;
    laplacian(nu,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
        laplacian(diff,y) Gauss linear corrected; //s.beale: Add this
}
```

```

interpolationSchemes
{
    default           linear;
    interpolate(HbyA) linear;
}

snGradSchemes
{
    default           corrected;
}

fluxRequired
{
    default           no;
    p                 ;
}
```

file */system/fvSolution*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

    U
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0;
    }

        //s.beale: add this...
        y
        {
                solver BICCG;
                preconditioner DILU;
                tolerance 1e-10;
                relTol 0;
        };
        //s.beale: done editing...

}

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

file */system/controlDict*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;

startFrom       latestTime;

startTime       0.0;

stopAt          endTime;

endTime         0.10;

deltaT          0.00005;

writeControl    timeStep;

writeInterval   200;

purgeWrite      5;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

The values for *deltaT* and *endTime* have been adjusted in order to yield a Courant Number less than 1. This can be checked in the file *solver.log*.

```
Courant Number mean: 0.105321 max: 0.260274
```

2.4.2 Directory /0

This directory should contain the follwing files.

- diff
- p
- U
- y

File */0/diff*: here, the diffusion coefficient is set to $D = 176 \cdot 10^{-7}\,\mathrm{m^2\,s^{-1}}$ by initializing the internal field.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      diff;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 176.0e-7;

boundaryField
{
    wall
    {
        type            zeroGradient;
    }

    inlet
    {
        type            zeroGradient;
                value                   uniform 1e-15;          // disallow
                    diffusion at inlet
    }

    outlet
    {
        type            zeroGradient;
    }

    membrane
    {
        type                    fixedValue;
                value                   uniform 1e-15;          // disable
                    diffusion at membrane
    }

}
```

file */0/p*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    wall
    {
        type            zeroGradient;
    }

    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    membrane
    {
        type            zeroGradient;
    }

}
```

File */0/U*: Here, the membrane outlet velocity is defined as 20 % of the mass of oxygen (see Table 3 on page 4).

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 3.0 0);

boundaryField
{
    wall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    inlet
    {
        type            fixedValue;
        value           uniform (0 1.469 0);
    }

    outlet
    {
        type            zeroGradient;
    }

    membrane
    {
        type            fixedValue;
        value           uniform (0 0 0.00182);
    }

}
```

File */0/y*: Here, inlet composition is defined as $y = 0.21$. The membrane outlet is defined as pure oxygen ($y = 1.00$).

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      y;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0.21;

boundaryField
{
    wall
    {
        type            zeroGradient;
    }

    inlet
    {
        type            fixedValue;
        value           uniform 0.21;
    }

    outlet
    {
        type            zeroGradient;
    }

    membrane
    {
        type            fixedValue;
        value           uniform 1.0;    // pure substance (e.g. oxygen) is removed
    }

}
```

2.5 Case b1: results

2.5.1 Check for convergence

The output of the solver is collected into the file 'solver.log' In the present example the solver was used to solve for pressure, velocity and mass fraction. Therefore, these parameters will be checked. OpenFOAM already provides a script that separates the output of the solver into separate x-y-data files. Type **'foamLog solver.log'** at the command line and the contents of solver.log is separated an all files are written to the directory 'logs'. The following files from the directory 'logs' are used.

```
Time-0
UxFinalRes-0   pFinalRes-0   pIters-0   yFinalRes-0
UxIters-0      pFinalRes-1   pIters-1   yIters-0
UyFinalRes-0   pFinalRes-2   pIters-2
UyIters-0      pFinalRes-3   pIters-3
UzFinalRes-0   pFinalRes-4   pIters-4
UzIters-0      pFinalRes-5   pIters-5
```

Again, the software GNUPLOT is used to generate the graphs. The GNUPLOT-scripts are contained in the documentation package in the folder 'scripts-gnu'. All graphs are generated at once by using the batch file. The resulting *.png files are located at the top of the directory (because the names start with an underscore). The following graphs correspond to the simultaion to remove 40 % of oxygen (see Table 3 on page 4). It can be seen that the solution of pressure and velocity converged after $5 \cdot 10^{-3}$. The behavior of the mass fraction is difficult to interpret. Convergence seems to be very fast and the graphs merely show fluctuations of the solver.

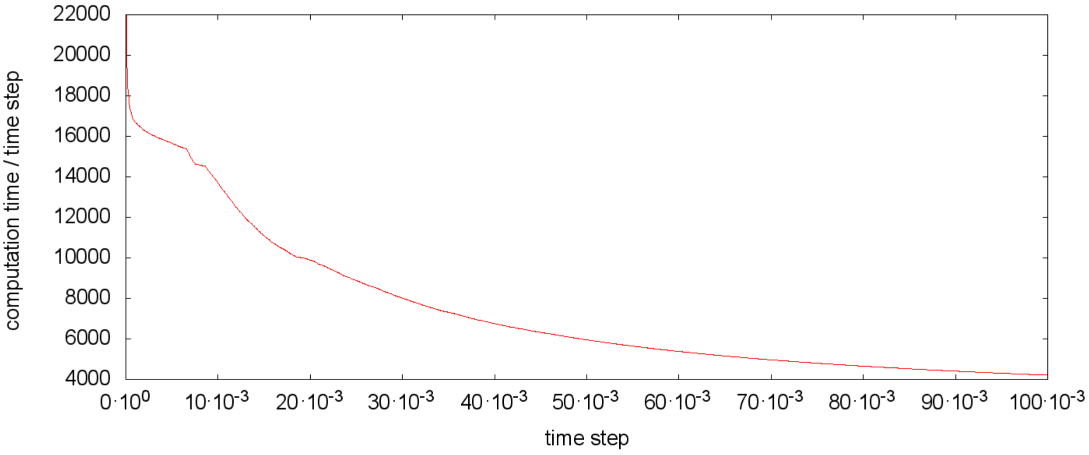


Figure 3: Computation time

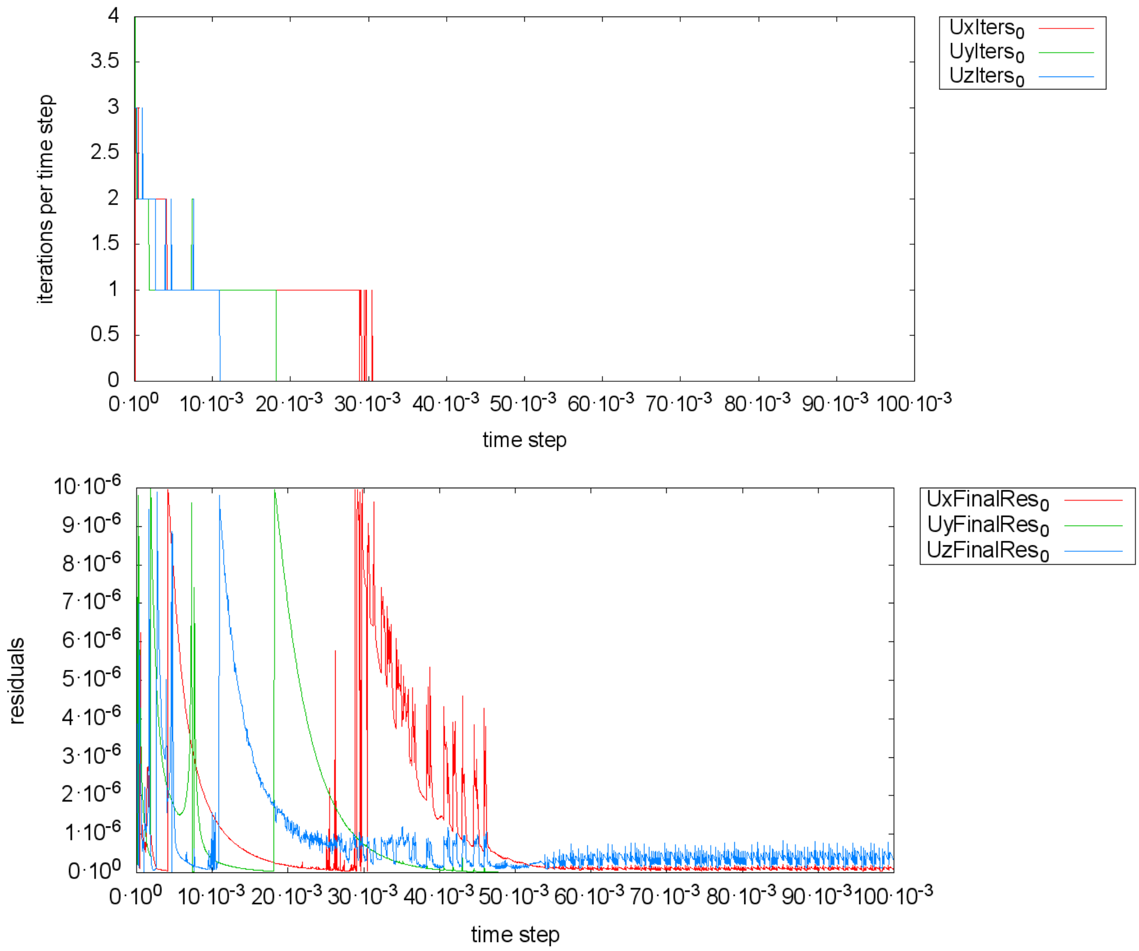


Figure 4: Top: Number of iterations for velocity; Bottom: Residuals of velocity

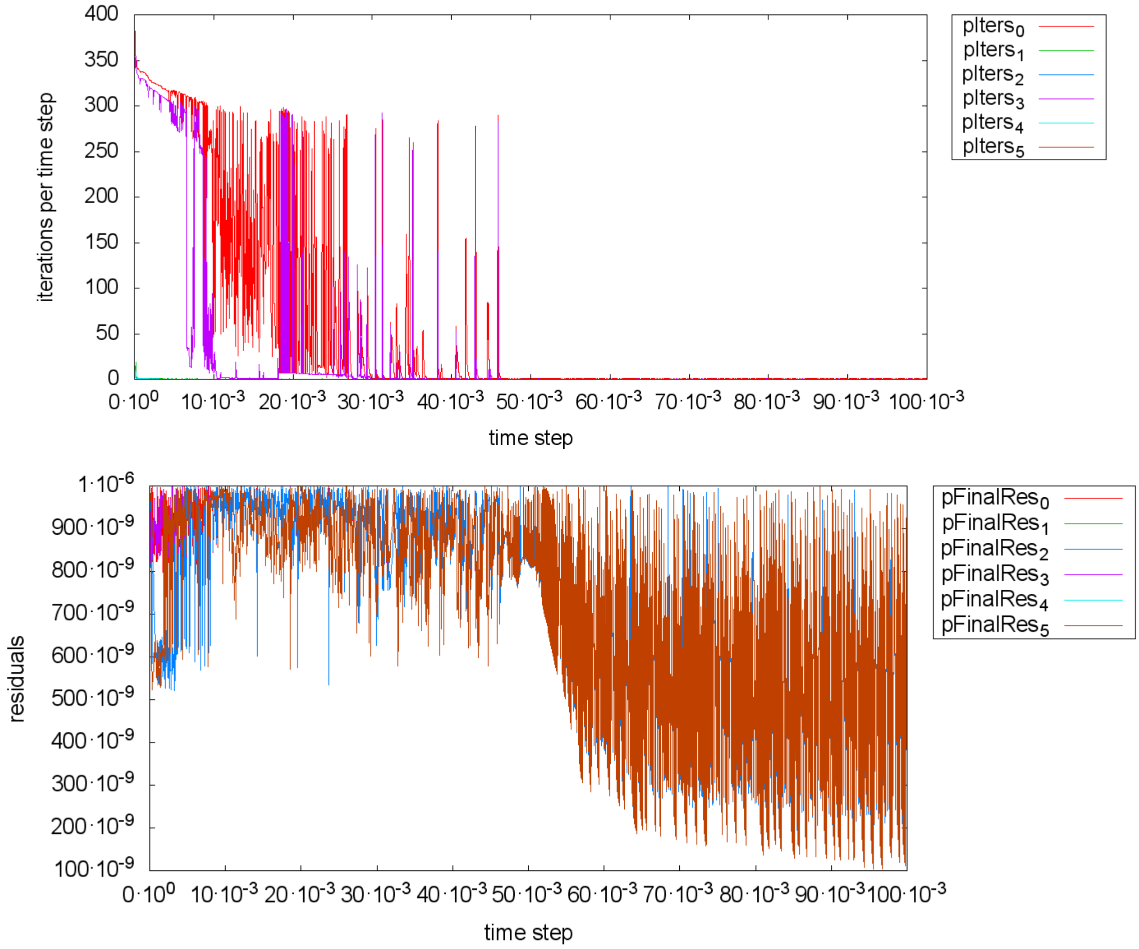


Figure 5: Top: Number of iterations for pressure; Bottom: Residuals of pressure

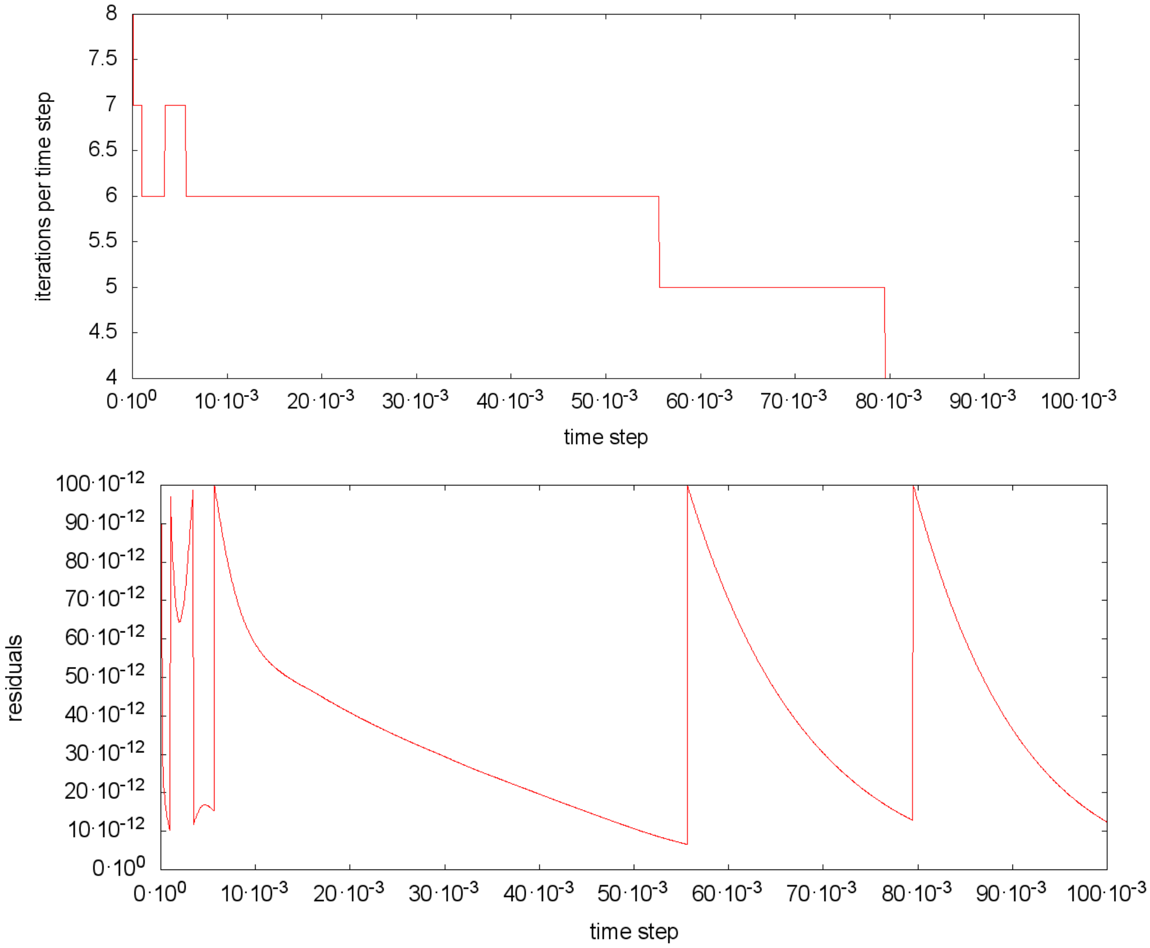


Figure 6: Top: Number of iterations for mass fraction; Bottom: Residuals of mass fractions

2.5.2 Simulation results

The initial problem was defined on page 4: a certain amount of oxygen shall be removed from a gas stream. The inlet of the channel is air with a mass fraction of oxygen of $y = 0.21$. Air flows from left to right in the figures below. At the membrane pure oxygen is removed. The distribution of oxygen mass fraction is shown in Figure 7. The value of 90 % oxygen removed was chosen in order to test the simulation in case that diffusion is reaching the limiting regime. In this case the local mass fraction near the outlet becomes negative ($Y = -0.08$). In the outlet relaxation zone the value of Y increases again slightly.

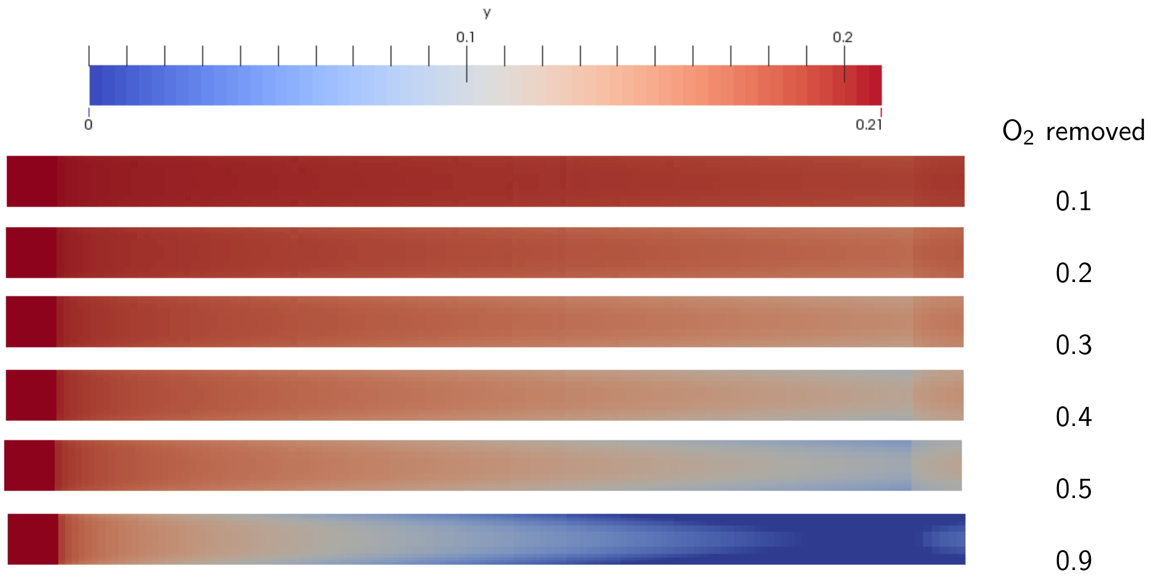


Figure 7: Mass fraction of oxygen at the membrane surface. Air flow is from left to right.

In order to compare the results, the mass fraction distribution at the end of the channel is compared. In Figure 8 the position of the cross section is indicated by the black arrow.

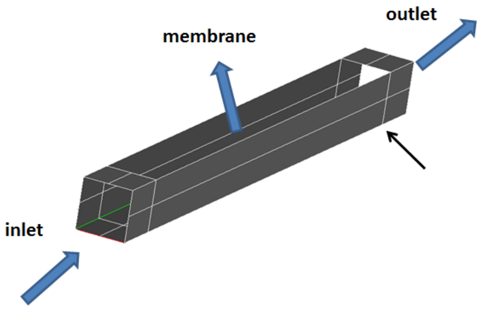


Figure 8: The black arrow shows the position of the cross section (for the following Figure).

The resulting mass fraction distributions are shown in Figure 9. The distribution f Y is more or less homogeneous up to the fraction of 40 % oxygen removed. A significant gradient in Y seems to develop for higher amounts of oxygen removal. The transition towards diffusion limitation can be observed as predicted by theory.

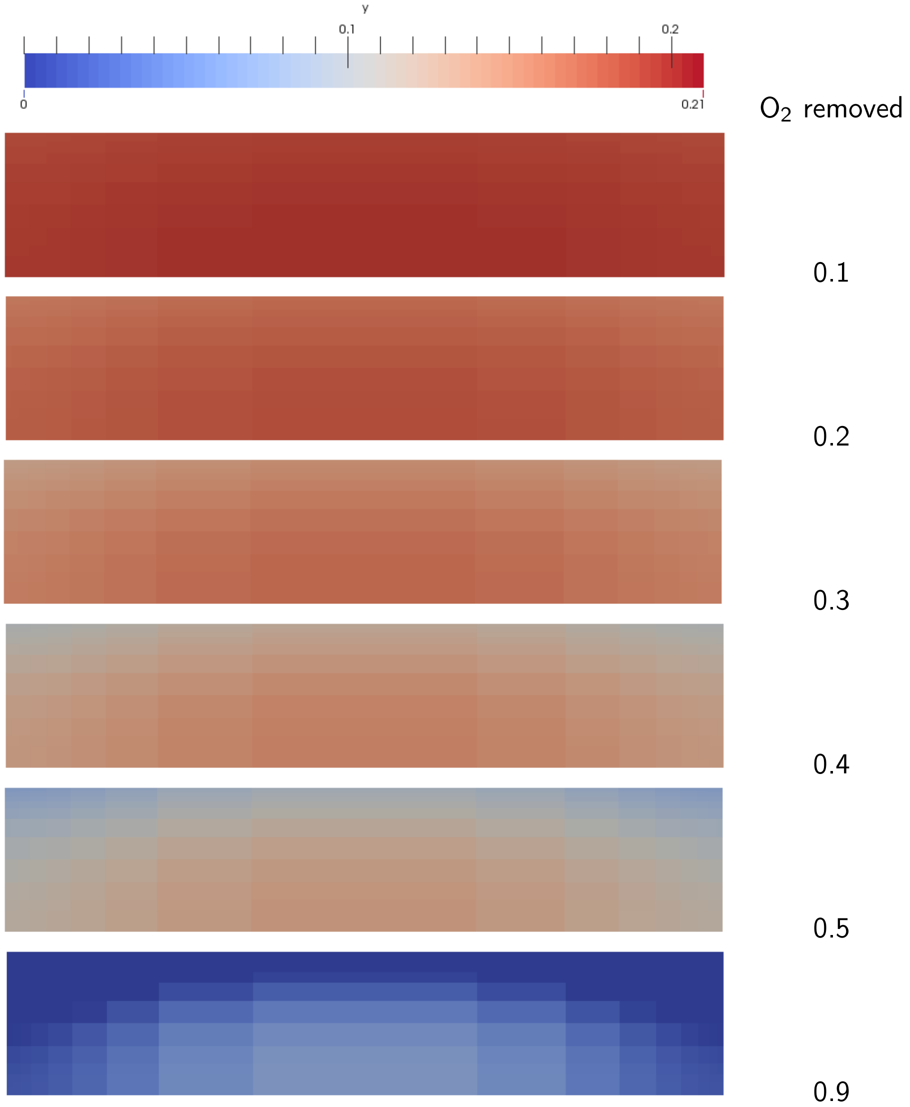


Figure 9: Mass fraction of oxygen at the cross section at the end of the channel.

3 Tutorial case b2: adding a porous zone

3.1 Definition of the problem

In the previous tutorial b1 the second outlet (membrane) was located directly at one side of the channel. Now, the second outlet is separated from the channel by a porous zone, e. g. like a gas diffusion layer (GDL) in a fuel cell. The geometry is shown in Figure 10 and the parameters are given in Table 4. Geometry and mesh are provided by the script 'OneChannel_v3_1.py' for the software SALOME, which is inlcuded in the directory '/mesh'.

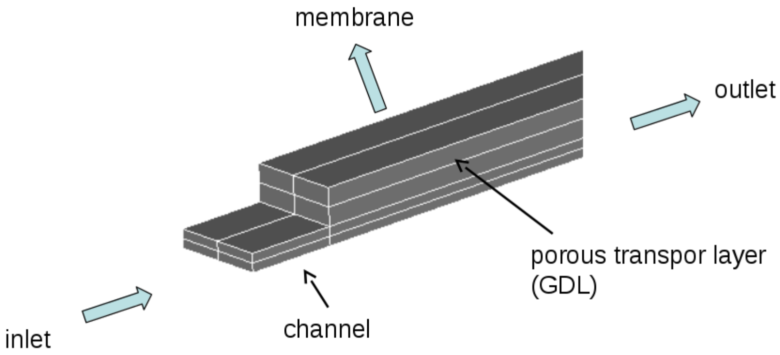


Figure 10: Section of the channel with porous transport layer.

Table 4: Channel dimensions

channel width	5 mm
channel height	1 mm
channel length	85 mm
height porous zone	2 mm
inlet relaxation zone	5 mm
outlet relaxation zone	5 mm

The inlet velocity of air is $u = 1.47\,\mathrm{m/s}$. The velocity at the membrane is identical as in the previous tutorial b1 (see Table 3 on page 4).

3.2 Creating the solver

For this task two options are available.

1. Combine the solver from tutorial case b1 (modified *icoFoam*) with the porous zone from solver *porousSimpleFoam*.
2. Start with the solver *porousSimpleFoam* and modifiy it according to tutorial b1.

Option 1 did not work for me therefore, option 2 is chosen. The source for the solver is contained in the directory 'source/simplePor' - which already reveals the name of the modified solver.
The flow through porous media is described by the Darcy-Forchheimer-Equation. The property of the porous media is defined in the file '/constant/porosityProperties' where two parameters *d* and *f* are scalars with components in x, y and z direction. The values for d are given in m^{-2}, i.e., $d = 1/K$ with K as the permeability of the porous layer.

File '/constant/porosityProperties'

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      porosityProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

poros1
{
    type            DarcyForchheimer;
    active          yes;
    cellZone        porous;

    DarcyForchheimerCoeffs
    {
        d   (1e7 1e7 1e7);
        f   (0 0 0);

        coordinateSystem
        {
            type    cartesian;
            origin  (0 0 0);
            coordinateRotation
            {
               type    axesRotation;
               e1      (1 0 0 );
               e2      (0 1 0);
            }
        }
    }
}
```

3.3 Attention:

The import of the mesh from the SALOME script is descibed in a separate tutorial. After the import, all boundary faces are assigned the type *patch* automatically. The solver requires *patch* for inlet and outlet, but needs the type *wall* for impermeable boundaries. Therefore, the entry in the file '/constant/polyMesh/boundary' must be changed manually.

File '/constant/polyMesh/boundary' – *before the change*

```
FoamFile
{
    version       2.0;
    format        ascii;
    class         polyBoundaryMesh;
    location      "constant/polyMesh";
    object        boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

5
(
    wall
    {
        type                patch;
        nFaces              2992;
        startFace           89528;
    }
    inlet
    {
        type                patch;
        nFaces              336;
        startFace           92520;
    }
```

File '/constant/polyMesh/boundary' – *after the change*

```
FoamFile
{
    version       2.0;
    format        ascii;
    class         polyBoundaryMesh;
    location      "constant/polyMesh";
    object        boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

5
(
    wall
    {
        type                wall;
        nFaces              2992;
        startFace           89528;
    }
    inlet
    {
        type                patch;
        nFaces              336;
        startFace           92520;
    }
```

3.4 Simulation results

Figure 11 shows the distribution of mass fraction of oxygen for a cross section of the channel. The diffusion limit seems to be reached as approximately 50 % of oxygen is removed. Under these conditions the value of y decreases to zero at the interface of the membrane. The solver also provides solutions for higher rates of removal, but these solutions have no physical meaning because local values of y are negative. The simulations have been performed with the turbulence model switched on and switched off. In both cases identical results have been obtained.

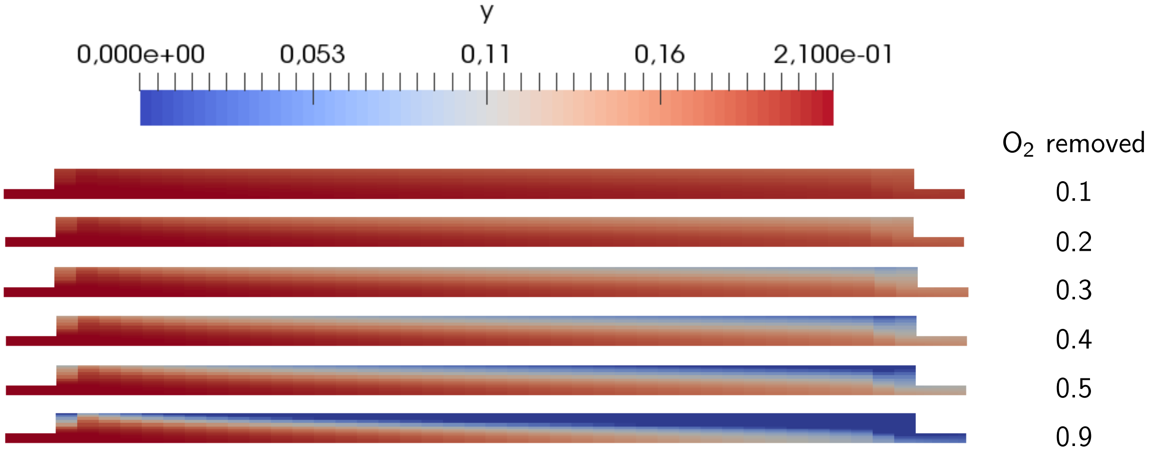


Figure 11: Mass fraction of oxygen at the cross section of the channel (middle of channel). Air flow is from left to right.

3.5 Error analysis

For further analysis the parameters mass flow and mass fraction have been calculated at three positions:

- cross section of the channel at the end of the inlet zone (length = 4 mm),
- cross section of the channel at the beginning of the outlet zone (length = 91 mm),
- cross section of the membrane outlet (height = 3 mm).

The overall mass balance can be obtained from the fluxes at the inlet and both outlets. The absolute error Δm is calculated from the mass fluxes according Equation 4. The resulting error is shown in Figure 12.

$$\Delta m = m(\text{inlet}) - m(\text{membrane}) - m(\text{outlet}) \tag{4}$$

From Figure 12 it can be deduced that the largest error corresponds to 16 % of the mass flux through the membrane. One possible eason for that may be the applied boundary condition. At the membrane the mass flux (velocity) and mass fraction are enforced, which is not the best option for an outlet.

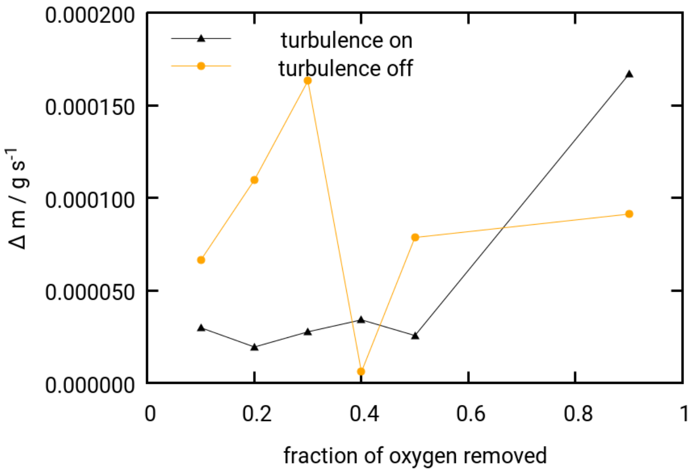


Figure 12: Absolute error of mass flow.

Since the outlet through the membrane is pre-defined, the error of the solution manifests itself at the 'free' outlet of the gas channel. Figure 13 shows the mass fraction at the outlet. It can be observed that the simulation values are slightly higher compared to the theoretical values.

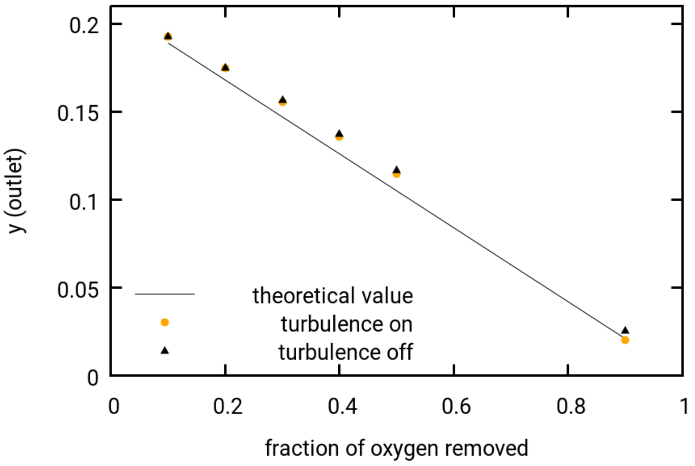


Figure 13: Mass fraction y at the beginning of the outlet region (length = 91 mm).

4 Useful commands

ideasUnvToFoam channel.unv convert the mesh from UNV to OpenFOAM format

checkMesh > checkMesh.log check the quality of the converted mesh

mapFields ../a1 -consistent start a simulation of a fine mesh with results from a coarse mesh in directory 'a1'; see [3]

icoFoam > solver.log & run simulation in background and write output to logfile 'solver.log'

icoFoam » solver.log & run simulation in background and append output to existing logfile 'solver.log'

nohup nice -n 19 icoFoam > solver.log & job runs after logout and has lowest priority on the computer

tail solver.log prints the last 10 lines of file 'solver.log'

foamToVTK -latestTime convert results (last time step) into VTK-files (for PARAVIEW)

foamLog solver.log separates the output of the file 'solver.log' and writes all values in separate files (x-y-data) into the directory 'logs'

---

[3] OpenFOAM: UserGuide.pdf page U-30