Wydział Nauk Technicznych

UNIVERSITY OF WARMIA AND MAZURY IN OLSZTYN
The Faculty of Technical Sciences
POLAND, 10-957 Olsztyn, M. Oczapowskiego 11
tel.: (48)(89) 5-23-32-40, fax: (48)(89) 5-23-32-55
URL: http://www.uwm.edu.pl/edu/sobieski/ (in Polish)

# NUMERYCZNE METODY MECHANIKI

## OpenFOAM – część 1

Publikacja została napisana w wyniku odbywania przez autora stażu w Holandii, współfinansowanego przez Unię Europejską w ramach Europejskiego Funduszu Społecznego (Program Operacyjnego Wiedza Edukacja Rozwój), zrealizowanego w projekcie Program Rozwojowy Uniwersytetu Warmińsko-Mazurskiego w Olsztynie (POWR.03.05.00-00-Z310/17).

**Wojciech Sobieski**

Utrecht, 2019

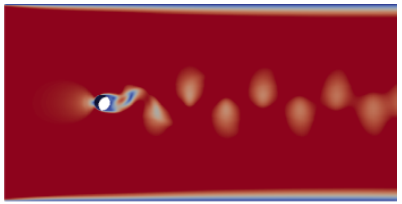
# OpenFOAM

Część 1 – zakres:

- podstawowe informacje o pakiecie OpenFOAM
- dokładna analiza przykładu **cavity**

Część 2 – zakres:

- zmiana geometrii i rodzajów brzegów
- dodanie przeszkody cylindrycznej (bezpośrednio i za pomocą pliku STL)
- lokalne zagęszczenie siatki
- monitorowanie rezyduów, siły unoszenia i siły oporu
- monitorowanie parametrów w wybranym punkcie przepływu
- zmiana solwera (z icoFoam na pisoFoam)



# OpenFOAM

**OpenFOAM** (**Open** **F**ield **O**peration **A**nd **M**anipulation) – pakiet programów napisanych w języku C++, służący do tworzenia modeli numerycznych różnych zagadnień przepływowych, oparty na Metodzie Objętości Skończonych.

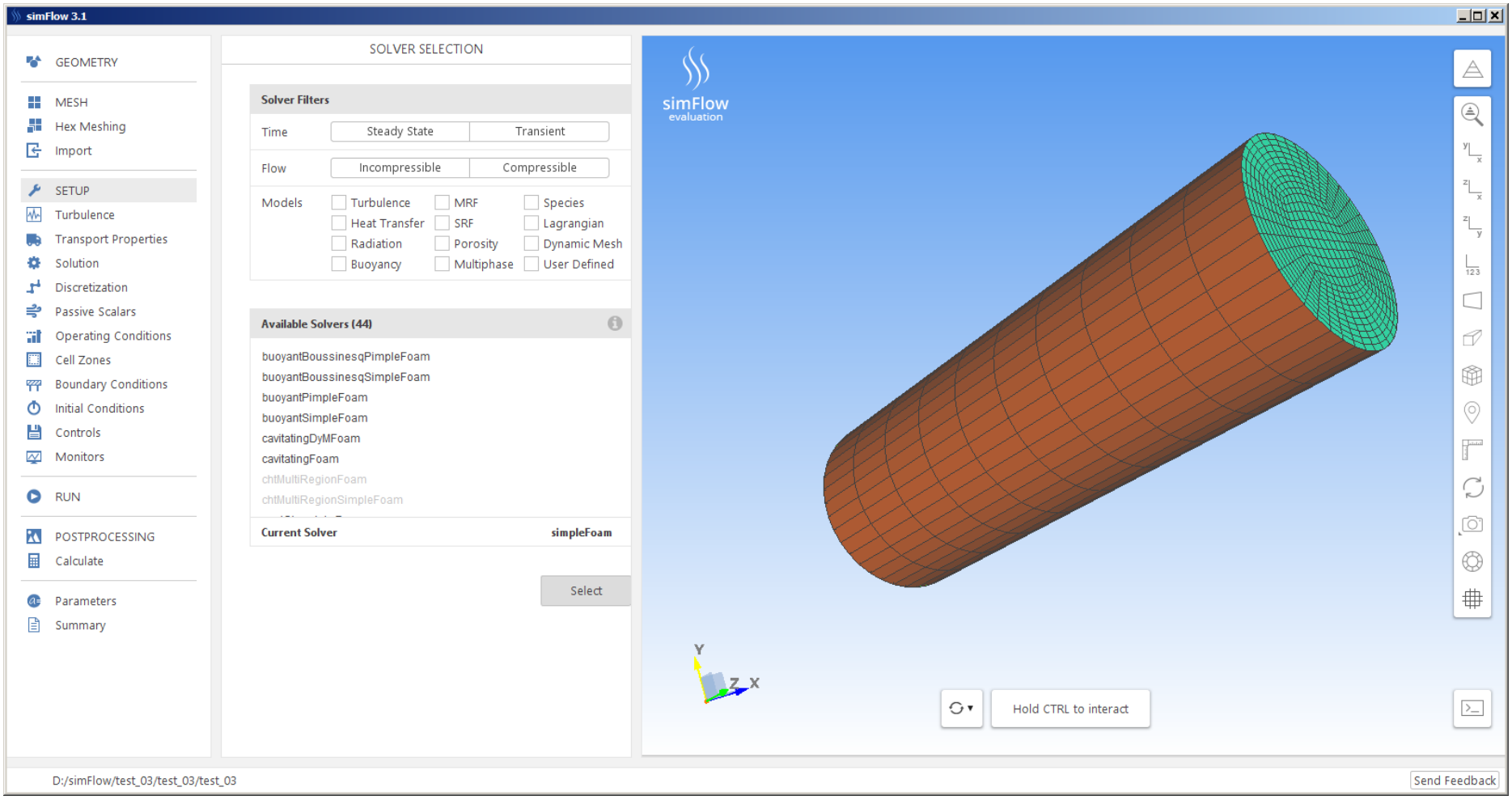
Cechy:

- bardzo popularny
- bezpłatny (licencja GPL)
- otwarty kod
- obliczenia równoległe na CPU
- oryginalnie tworzony dla systemów UNIX/Linux
- brak graficznego interfejsu (praca w konsoli)
- brak dobrej oficjalnej dokumentacji
- dużo dokumentacji tworzonej przez użytkowników

simFlow

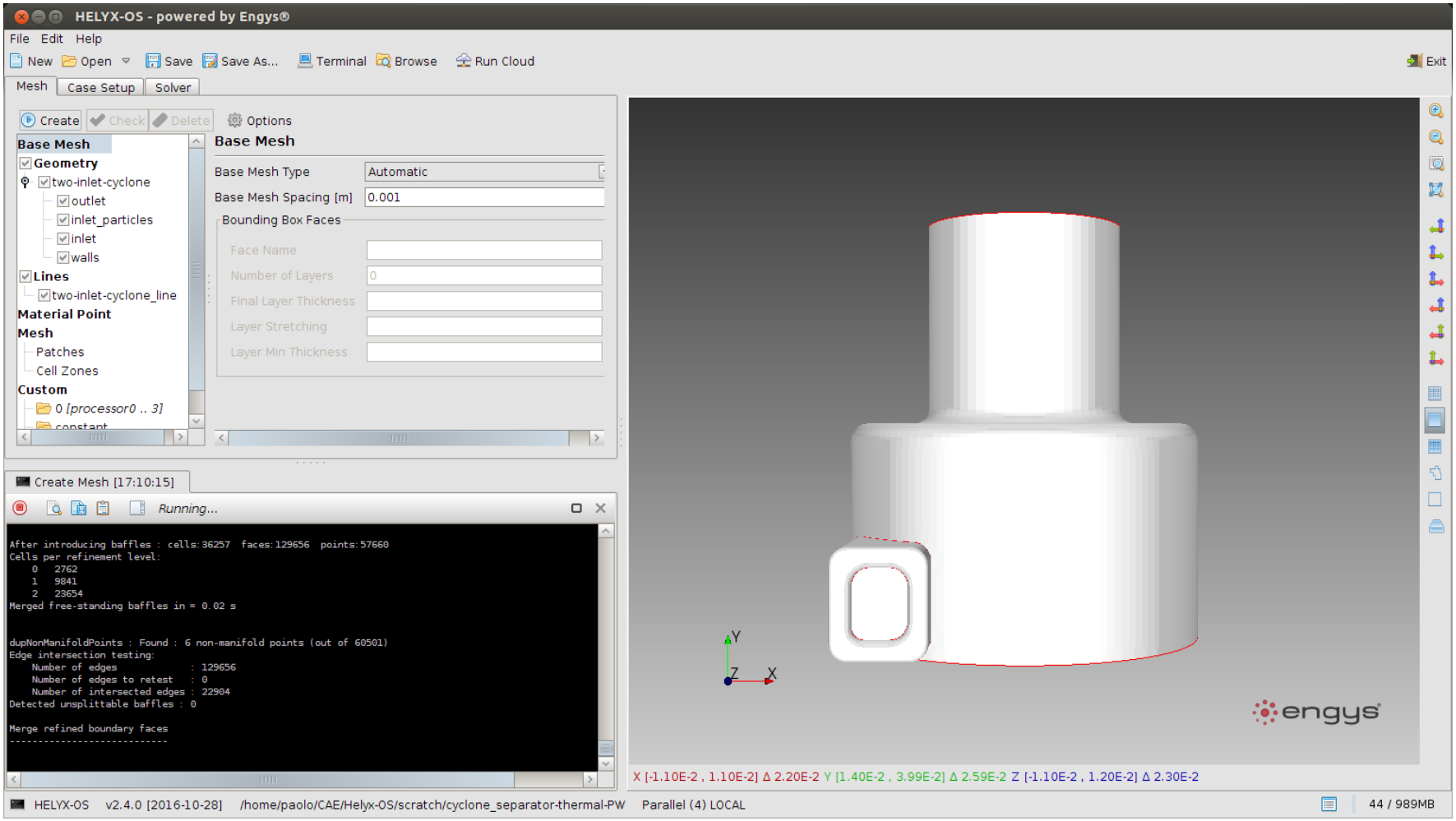
HELYX®os

# OpenFOAM



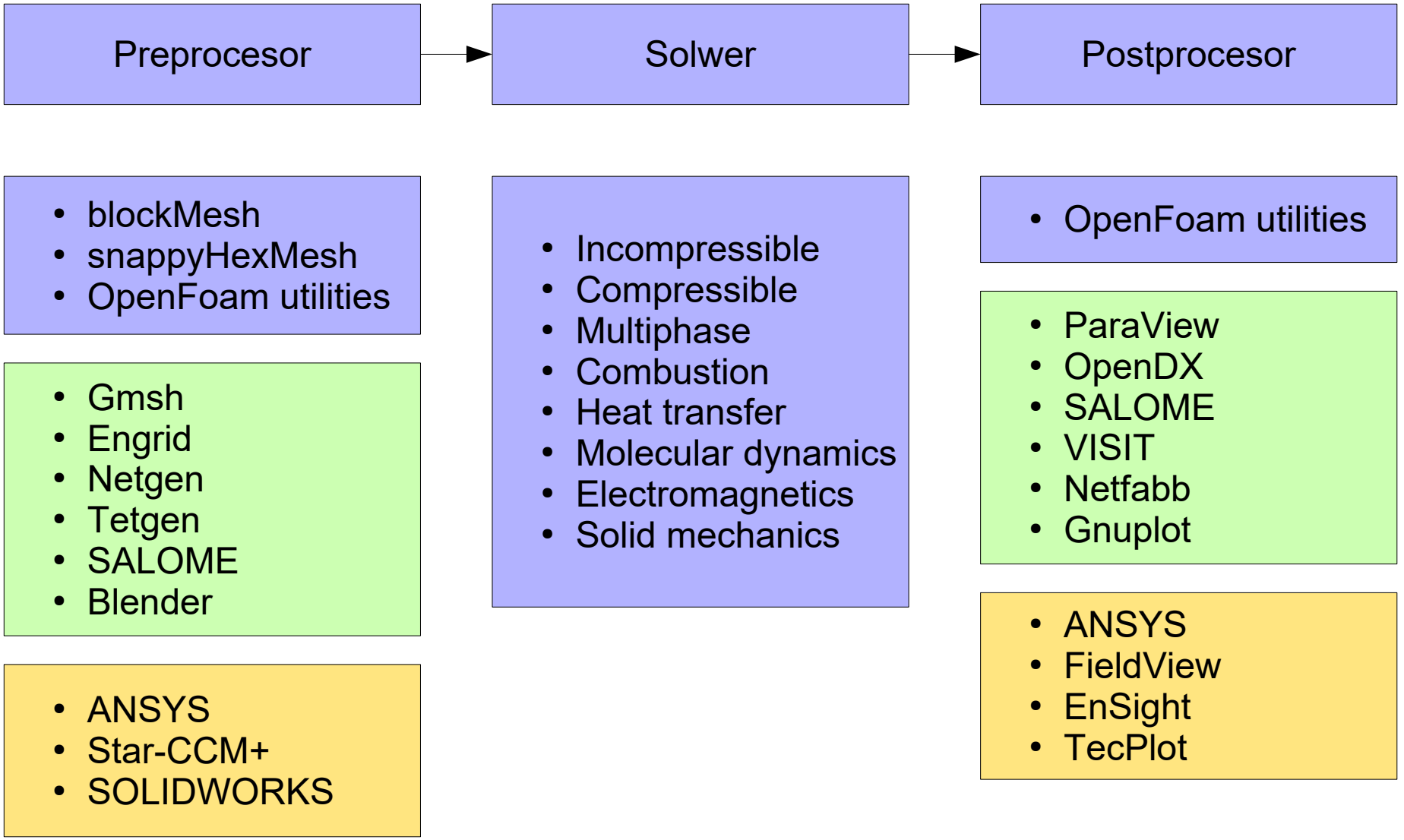

Nakładka graficzna simFlow – produkt komercyjny (Windows, UNIX/Linux).

# OpenFOAM

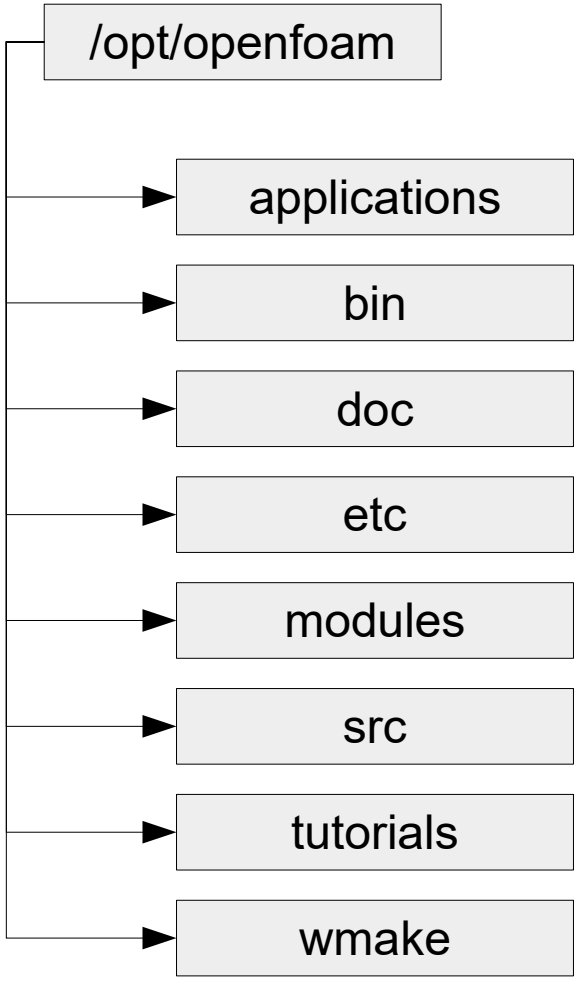



Nakładka graficzna HELYX-OS – na licencji GPL (UNIX/Linux).

# OpenFOAM – narzędzia



# OpenFOAM – foldery pakietu




OpoenFOAM został stworzony dla systemów UNIX/Linux (choć można go również używać w systemach Windows).

Tu znajduje się dokumentacja w postaci plików PDF.

Tu znajduje się kilkadziesiąt pogrupowanych tematycznie przykładów.

W tym katalogu się nie pracuje – trzeba stworzyć katalog roboczy w folderze użytkownika.

# OpenFOAM – solwery

Najprostsze solwery:

- laplacianFoam – równanie Laplace'a
- potentialFoam – przepływy potencjalne

Solwer użyty w przykładzie **cavity**.

Przepływy nieściśliwe:

- **icoFoam – niestacjonarne i nieściśliwe przepływy laminarne cieczy niutonowskich**
- pisoFoam – niestacjonarne i nieściśliwe przepływy laminarne lub turbulentne
- simpleFoam – stacjonarne i nieściśliwe przepływy laminarne lub turbulentne
- …

Przepływy ściśliwe:

- rhoCentralFoam – przepływy ściśliwe
- sonicFoam – przepływy niestacjonarne gazów, trans i supersoniczne, laminarne lub turbulentne
- …

# OpenFOAM – solwery

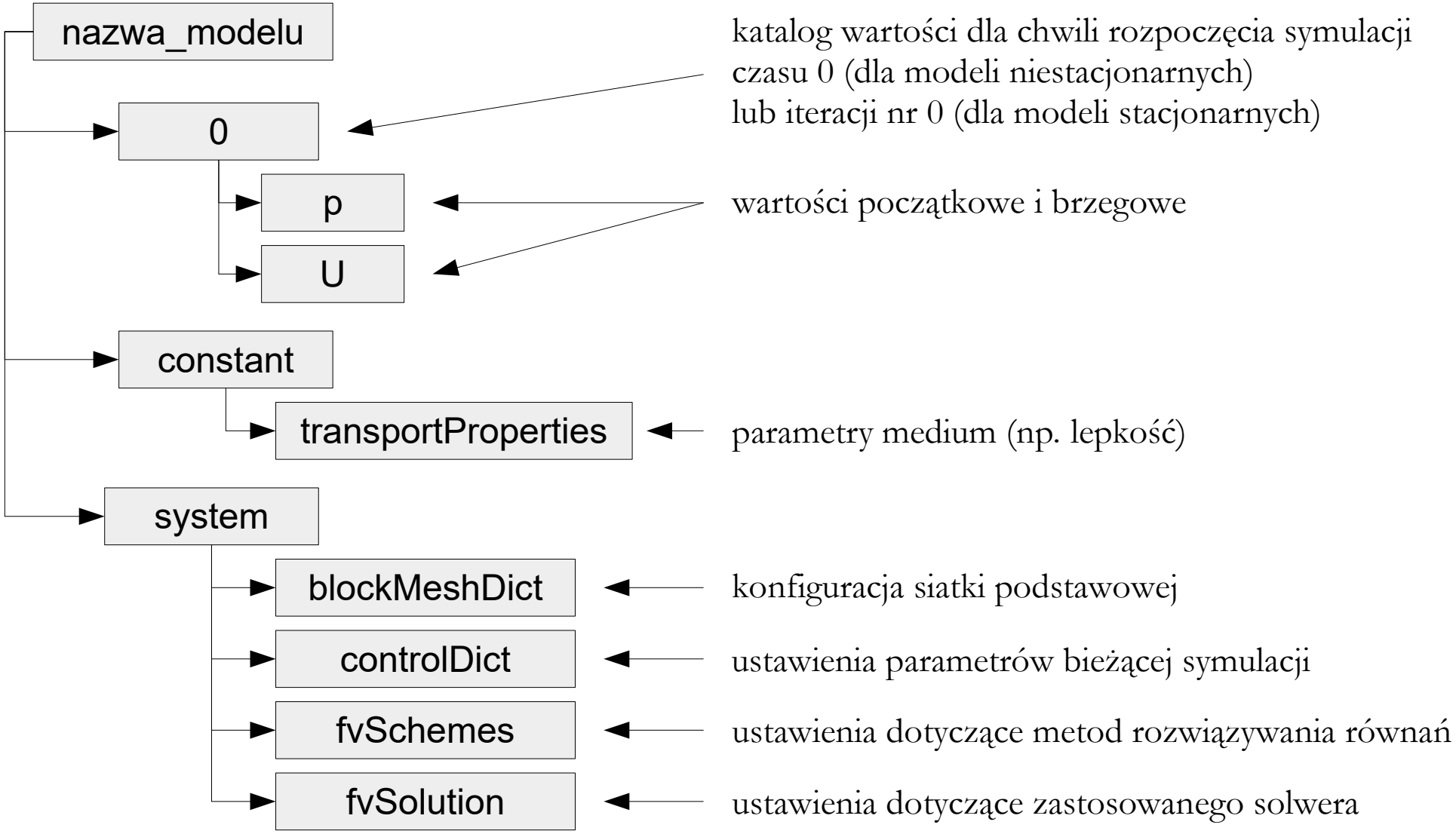
Przepływy wielofazowe:

- interFoam – przepływy nieściśliwe i izotermiczne 2 niemieszających się płynów bazujące na metodzie Volume of Fluid
- interPhaseChangeFoam – przepływy nieściśliwe i izotermiczne 2 niemieszających się płynów, uwzględniające wymianę wielkości bilansowanych
- multiPhaseEulerFoam – przepływy ściśliwe z wymianą ciepła
- …

Inne:

- solidDisplacementFoam – niestacjonarne analizy ciał liniowo-elastycznych o małym odkształceniu
- mdFoam – analizy bazujace na dynamice molekularnej
- buoyantSimpleFoam – niestacjonarne, turbulentne przepływy płynów ściśliwych
- …

# OpenFOAM – struktura katalogów



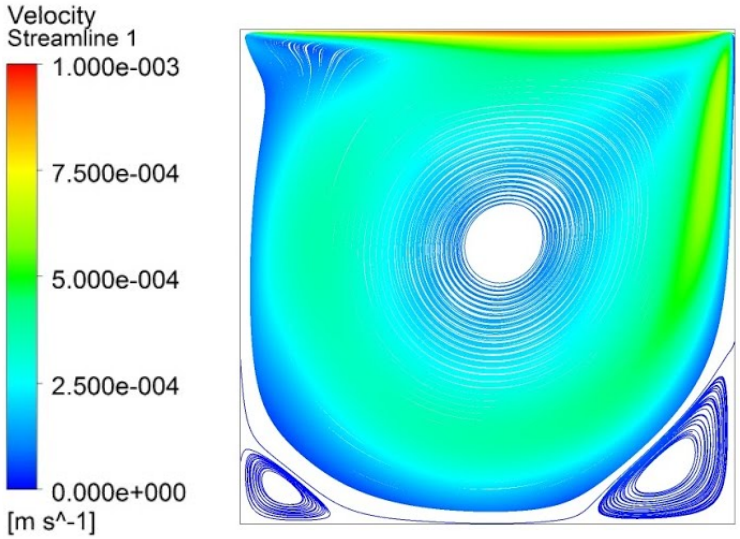
# OpenFOAM – słowniki

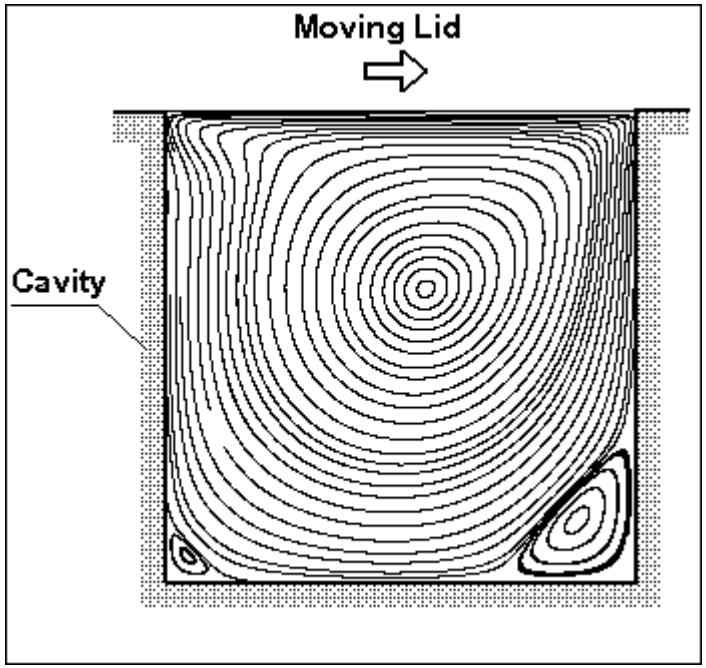
**Słownik** (dictionary, tablica asocjacyjna) – nazwa dla powszechnie stosowanego w informatyce abstrakcyjnego typu danych, który przechowuje pary (unikatowy klucz, wartość) i umożliwia dostęp do wartości poprzez podanie klucza.

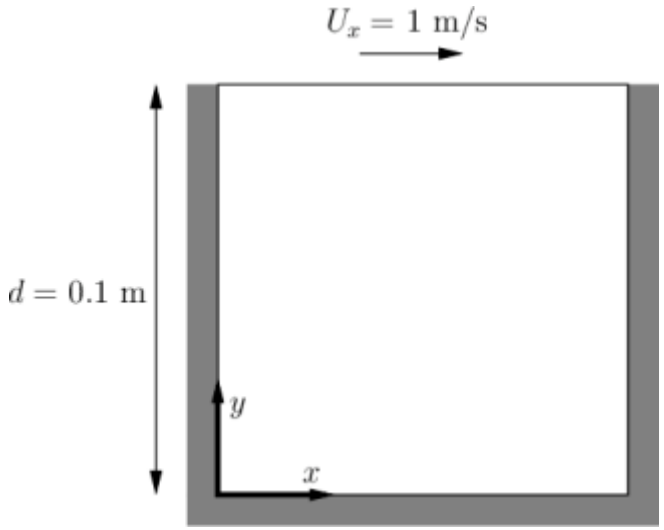
```
klucz: styczeń   wartość:  31
klucz: luty-1    wartość:  28
klucz: luty-2    wartość:  29
klucz: marzec    wartość:  31
```

Każdy model tworzony w OpenFOAM wymaga skonfigurowania odpowiedniego zestawu słowników. Zestaw ten może się różnic dla tego samego solwera w zależności od ustawień modelu (np. zastosowanego modelu turbulencji) lub wykorzystywanych narzędzi.

# Cavity – zagadnienie



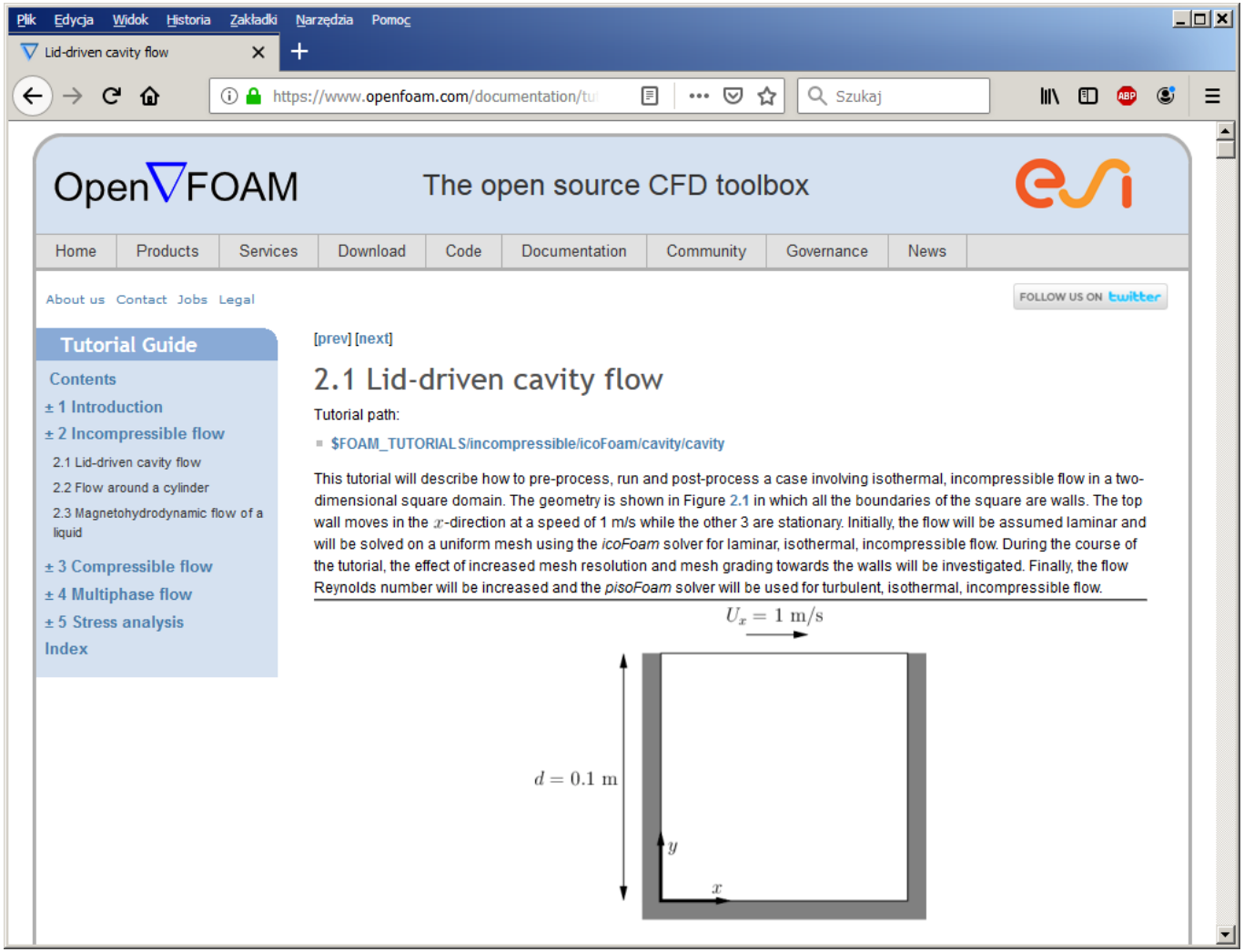




**cavity** – jeden z klasycznych przykładów referencyjnych mechaniki płynów

**cavity** – niestacjonarny, laminarny przepływ cieczy newtonowskiej we wnęce (solwer **icoFoam**)

# Cavity – zagadnienie




**cavity** – pierwszy przykład w oficjalnym tutorialu
(dostępnym na stronie projektu OpenFOAM bezpośrednio lub w dokumentacji PDF)

# Cavity – geometria (blockMeshDict)

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  dev
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

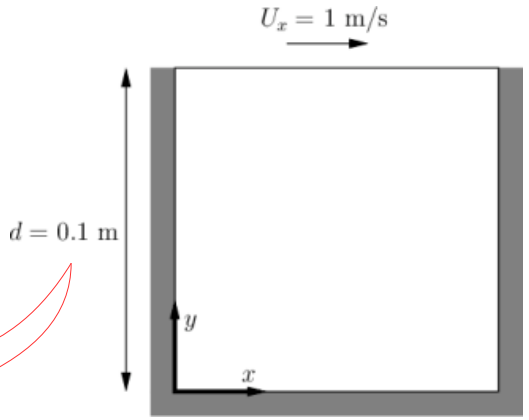
Nagłówek OpenFOAM
(na dalszych slajdach będzie pomijany).

# Cavity – geometria (blockMeshDict)

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  dev
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

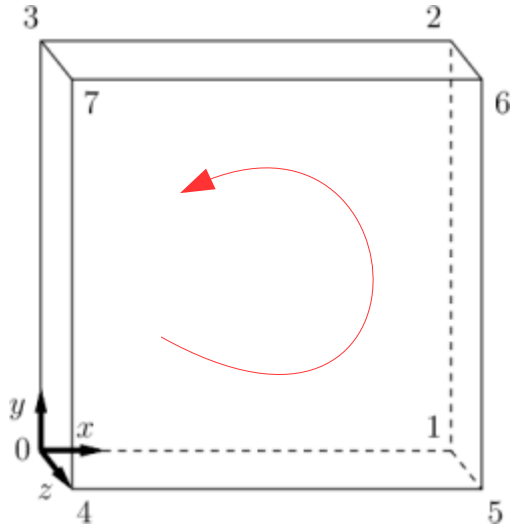



Współczynnik skali: w tym przypadku wartość X
jest zapisana w sekcji **verticles** w zakresie od 0 do 1 [m],
ale ponieważ współczynnik skali wynosi 0.1,
to wymiary siatki na kierunku X będą w zakresie od 0 do 0.1 [m].
Skalowanie dotyczy wszystkich kierunków przestrzeni.

# Cavity – geometria (blockMeshDict)

```
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

zaczynamy od osi x




```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);
```

Definicja współrzędnych na kierunku X, Y oraz Z:
w tym przypadku chcemy uzyskać model 2D,
więc szerokość obszaru obliczeniowego (domeny)
jest mała w stosunku do pozostałych wymiarów.

Punkty numeruje się zaczynając od 0.

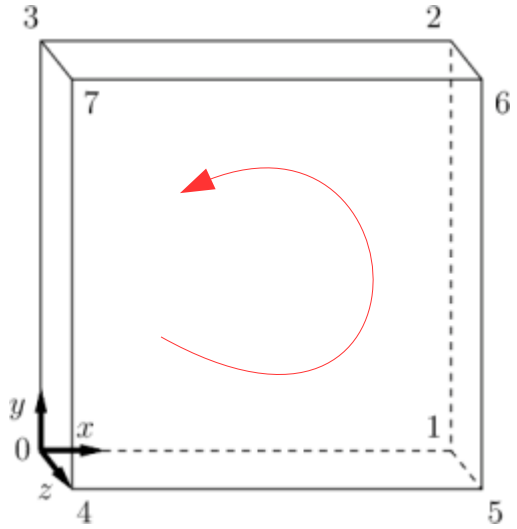
# Cavity – geometria (blockMeshDict)

```
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);
```

zaczynamy od osi x

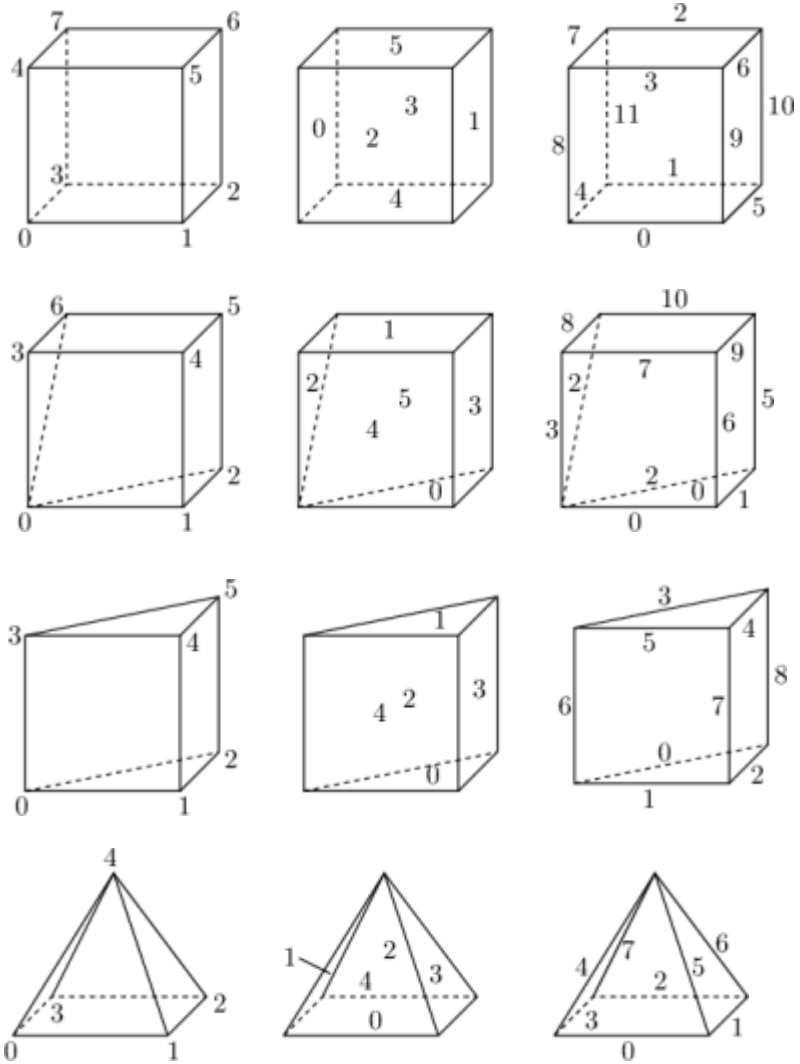


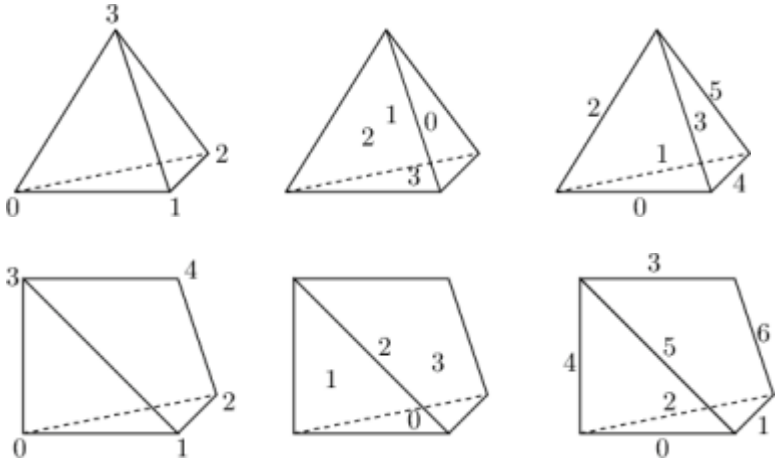

Tworzenie bloków 3D z punktów zdefiniowanych w sekcji **vertices**:
w tym przypadku jest to prostopadłościan (**hex**).

Kolejność numerów punktów tworzących blok jest ważna.

Liczba bloków może być dowolna
(ale trzeba później zdefiniować ich względne ułożenie).

# Cavity – geometria (blockMeshDict)





Typy bloków 3D:

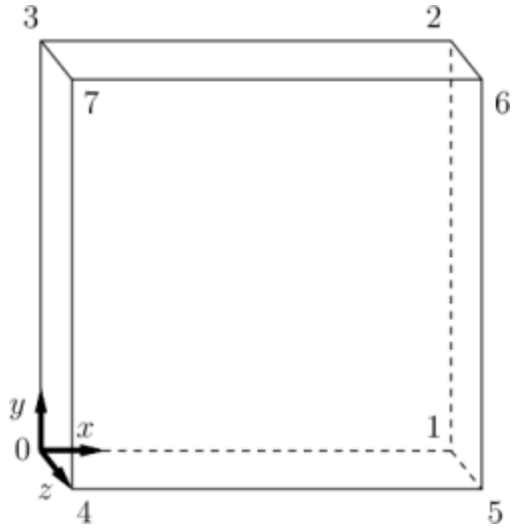
**hex** – prostopadłościan;
**wedge** – prostopadłościan;
ze „zwiniętymi” wierzchołkami;
**prism** – pryzma;
**pyr** – piramida;
**tet** – czworościan;
**tetWedge** – czworościan
z „łamaną” krawędzią.

# Cavity – geometria (blockMeshDict)

```
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);


blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);
```




Definicja liczby komórek siatki na kierunkach X, Y oraz Z:
w tym przypadku symulacja ma być 2D, więc na kierunku Z jest tylko jedna komórka.

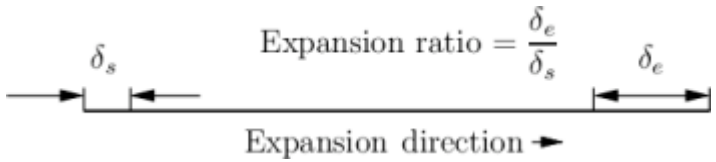
Zdefiniowana siatką jest tzw. siatką strukturalną.

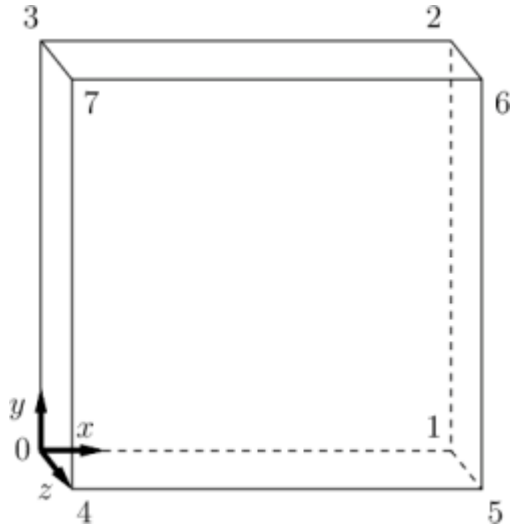
# Cavity – geometria (blockMeshDict)

```
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);
```




Definicja stopnia równomierności siatki: w tym przypadku wszystkie komórki na każdym kierunku będą miały taki sam rozmiar.

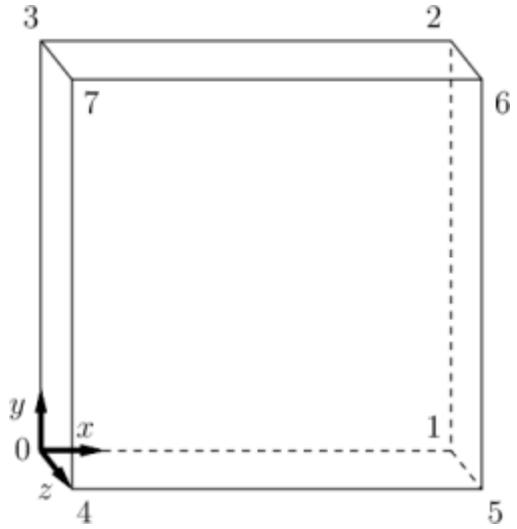
Liczby mogą być mniejsze od jedności (zagęszczanie w kierunku malejących wartości danej osi) lub większe od jedności (zagęszczanie w kierunku rosnących wartości danej osi).

# Cavity – geometria (blockMeshDict)

```
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);


blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);
```

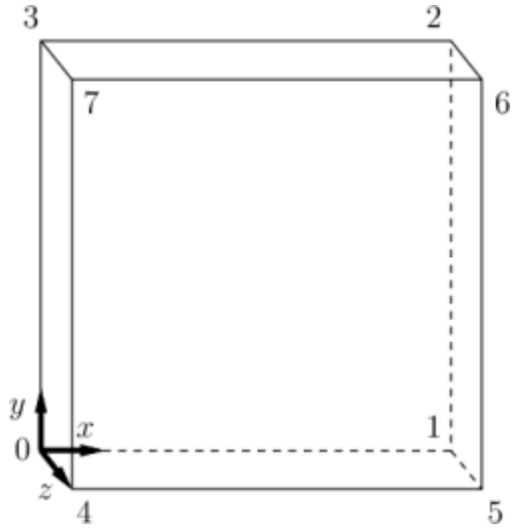



Sekcja umożliwiająca zdefiniowanie krawędzi innych niż prostoliniowe.
Dostępne opcje to: **arc**, **simpleSpline**, **polyLine**, **polySpline**, **line**.

# Cavity – geometria (blockMeshDict)

```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

Nazwy brzegów są bardzo ważne gdyż występują w innych słownikach!

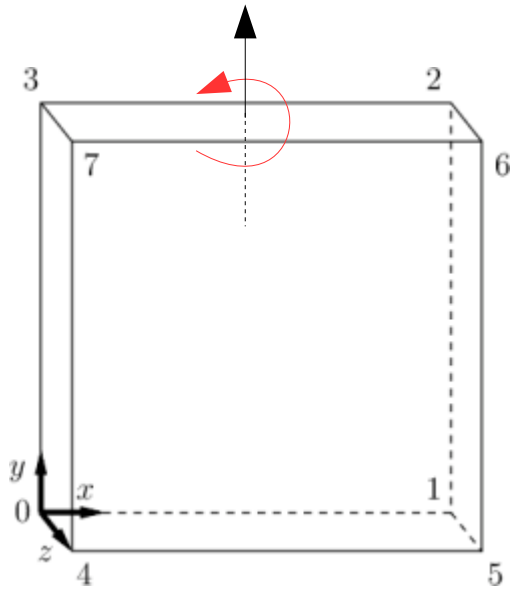



Definiowanie typów brzegów:

| | |
|---|---|
| **patch** | – wlot lub wylot; |
| **symmetryPlane** | – płaszczyzna symetrii; |
| **empty** | – kierunek, dla którego nie rozwiązuje się równań; |
| **wedge** | – płaszczyzny ograniczające dla geometrii osiowo-symetrycznej; |
| **cyclic** | – płaszczyzna określająca warunek periodyczny |
| **wall** | – ściana; |
| **processor** | – powierzchnia podziału sieci dla obliczeń równoległych. |

# Cavity – geometria (blockMeshDict)

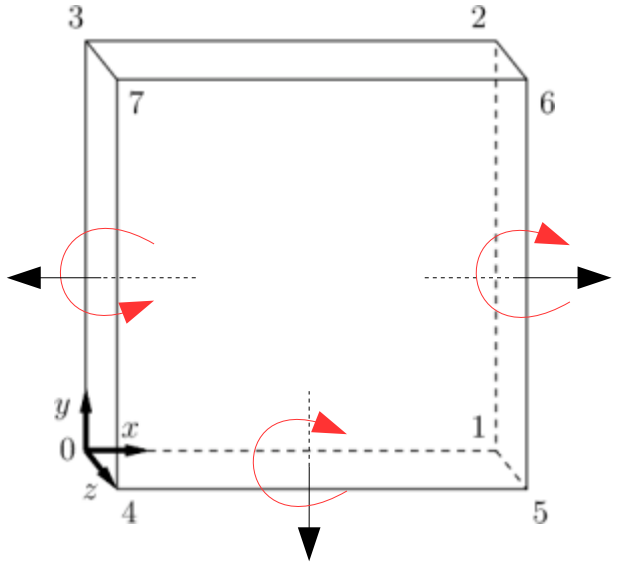
```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```




Kolejność węzłów tworzących poszczególne powierzchnie jest bardzo ważna: patrząc z wnętrza bloku na daną powierzchnię stosujemy regułę prawej ręki.

# Cavity – geometria (blockMeshDict)

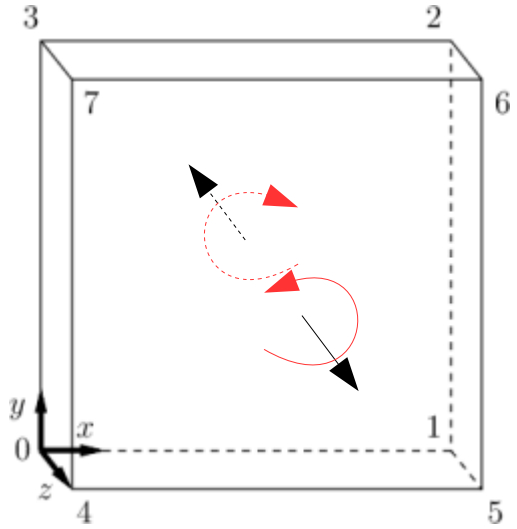
```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```




Kolejność węzłów tworzących poszczególne powierzchnie jest bardzo ważna: patrząc z wnętrza bloku na daną powierzchnię stosujemy regułę prawej ręki.

# Cavity – geometria (blockMeshDict)

```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```




Typ **empty** oznacza, że dla danego kierunku
(muszą być zawsze dwie ściany typu **empty** leżące naprzeciw siebie)
nie będą rozwiązywane równania. W tym przypadku model 3D
staje się modelem dwuwymiarowym.

Można zadać dwie pary ścian typu **empty**,
aby uzyskać model jednowymiarowy.

# Cavity – geometria (blockMeshDict)

```
mergePatchPairs
(
);
```

Definicja połączeń między blokami (muszą być co najmniej dwa). W analizowanym przypadku sekcja jest pusta bo zdefiniowano tylko jeden blok.

Możliwe opcje to:

**face matching** – stosuje się, gdy dwie sąsiadujące powierzchnie są utworzone z tych samych zestawów węzłów;

**face merging** – stosuje się w innych przypadkach niż powyższy.

Omówiliśmy w skrócie wszystkie sekcje.

| Keyword | Description | Example/selection |
|---|---|---|
| `scale` | Scaling factor for the vertex coordinates | `0.001` scales to mm |
| `vertices` | List of vertex coordinates | `(0 0 0)` |
| `edges` | Used to describe `arc` or `spline` edges | `arc 1 4 (0.939 0.342 -0.5)` |
| `block` | Ordered list of vertex labels and mesh size | `hex (0 1 2 3 4 5 6 7)`<br>`(10 10 1)`<br>`simpleGrading (1.0 1.0 1.0)` |
| `patches` | List of patches | `symmetryPlane base`<br>`( (0 1 2 3) )` |
| `mergePatchPairs` | List of patches to be merged | see section 4.3.2 |

Table 4.2: Keywords used in *blockMeshDict*.

# Cavity – warunki początkowe i brzegowe (p)

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

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Definicja jednostki wielkości opisanej w pliku. Poszczególne liczby oznaczają wykładniki następujących jednostek (tu wariant SI):

[kg m s K mol A Cd]

**WAŻNE**: w wielu solwerach OpenFOAM symbol **p** nie oznacza ciśnienia [Pa], ale ciśnienie podzielone przez gęstość!

W tym przypadku jednostką jest [$m^2/s^2$].

$$\frac{Pa}{\frac{kg}{m^3}} = \frac{N\,m^3}{m^2\,kg} = \frac{kg\,m\,m^3}{s^2\,m^2\,kg} = \frac{m^2}{s^2}$$

Nazwa pliku (w tym przypadku **p**) informuje, jakie pole skalarne lub wektorowe jest w nim opisane.

W przykładzie pominięto nagłówek pliku.

# Cavity – warunki początkowe i brzegowe (p)

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

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Określenie jaka ma być wartość danego pola we wnętrzu obszaru obliczeniowego.

Za pomocą dodatkowych narzędzi (**setFields**) można definiować różne wartości w różnych miejscach.

Pola mogą być jednorodne (uniform) lub niejednorodne (nonuniform). W drugim przypadku plik musi zawierać dodatkową tablicę wartości pola w każdej komórce sieci.

W tym przypadku wartość zero oznacza ciśnienie względem ciśnienia referencyjnego.

W solwerach nieściśliwych zasadniczo nie podaje się gęstości – jedynym parametrem materiałowym jest kinematyczny współczynnik lepkości.
Aby obliczyć rzeczywiste ciśnienie statyczne należy pomnożyć ciśnienie wyliczane przez solwer (p*) przez gęstość płynu określoną dla tej samej temperatury, co kinematyczny współczynnik lepkości.
Do wyniku należy dodać ciśnienie odniesienia, np. ciśnienie atmosferyczne.

$$p = p^* \cdot \rho + p_{ref}$$

$$p^* = \frac{p_{statyczne}}{\rho}$$

# Cavity – warunki początkowe i brzegowe (p)

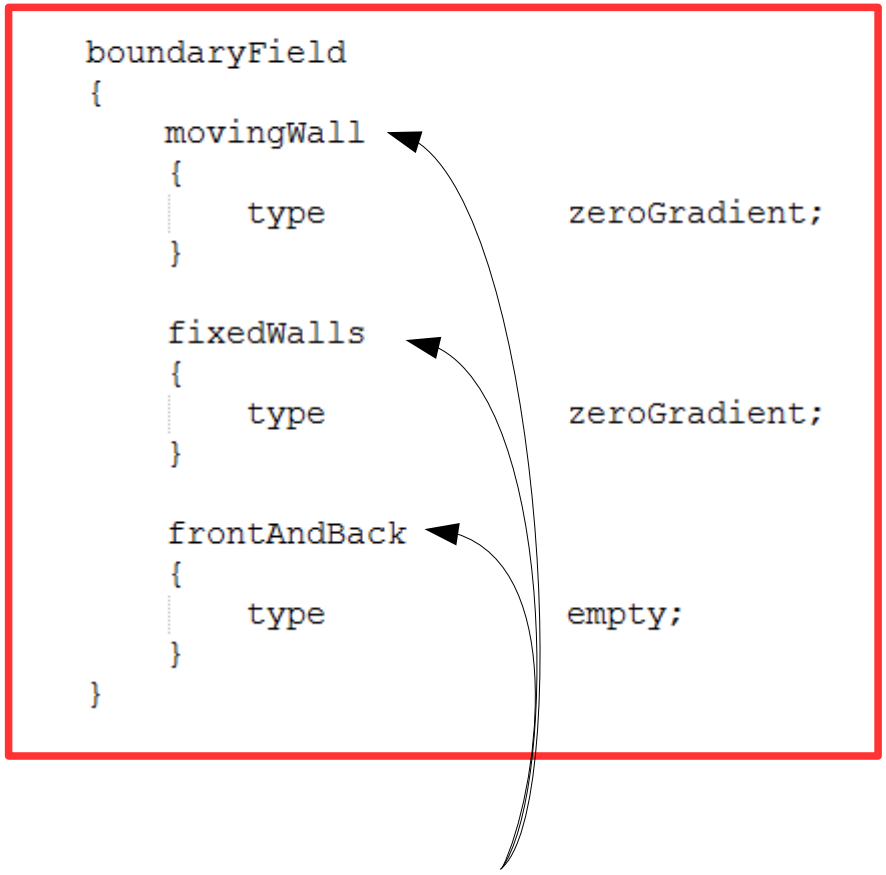
```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}
```



Definicja rodzajów warunków brzegowych.

W OpenFOAM istnieje około 70 rodzajów warunków brzegowych! Można je poznać czytając dokumentację i analizując oficjalne przykłady.

Warunek **zeroGradient** oznacza, że na brzegu nie może istnieć strumień danej wielkości (jest to warunek brzegowy von Neumanna).

Warunek **empty** oznacza, że na danym kierunku nie będą rozwiązywane równania.

Nazwy brzegów muszą być zgodne z nazwami zdefiniowanymi w słowniku **blockMeshDict**!

# Cavity – warunki początkowe i brzegowe (U)

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

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            noSlip;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Definicja jednostki wielkości opisanej w pliku. Poszczególne liczby oznaczają wykładniki następujących jednostek (tu wariant SI):

[kg m s K mol A Cd]

W tym przypadku jednostką jest [m/s].

Nazwa pliku (w tym przypadku **U**) informuje, jakie pole skalarne lub wektorowe jest w nim opisane.

W przykładzie pominięto nagłówek pliku.

# Cavity – warunki początkowe i brzegowe (U)

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

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            noSlip;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Określenie jaka ma być wartość danego pola we wnętrzu obszaru obliczeniowego.

Ponieważ w tym przypadku chodzi o pole wektorowe, należy podać trzy wartości, odpowiednio dla kierunku X, Y oraz Z.

# Cavity – warunki początkowe i brzegowe (U)

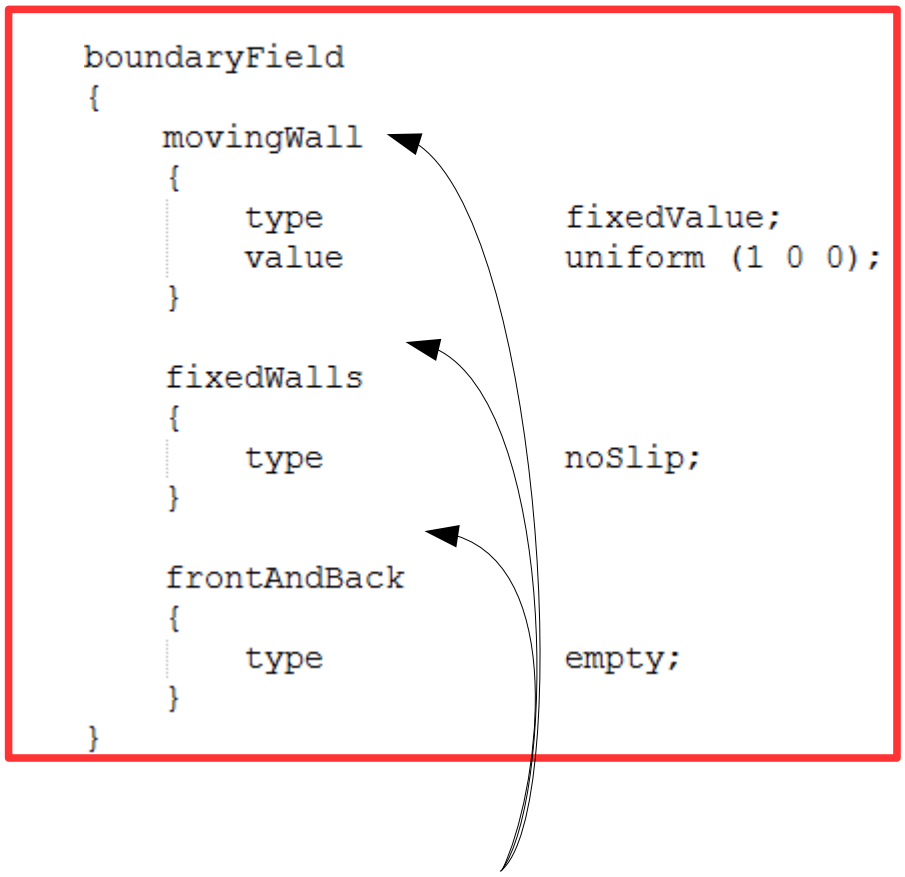
```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            noSlip;
    }

    frontAndBack
    {
        type            empty;
    }
}
```



Nazwy brzegów muszą być zgodne z nazwami zdefiniowanymi w słowniku **blockMeshDict**!

Definicja rodzajów warunków brzegowych.

Warunek **fixedValue** oznacza niezmienną w czasie wartość liczbową. Wartość ta może być stała dla wszystkich komórek brzegu (uniform) lub zmienna (nonuniform). W tym przypadku składowa X wektora prędkości jest równa 1 [m/s] (pozostałe składowe są równe zeru).

Warunek **noSlip** oznacza, że na brzegu nie ma poślizgu.

Warunek **empty** oznacza, że na danym kierunku nie będą rozwiązywane równania.

Użyty w przykładzie solwer (**icoFoam**) wymaga zdefiniowania tylko dwóch pól (słowników w podkatalogu 0): ciśnienia (**p**) i prędkości (**U**).

# Cavity – rodzaj medium (transportProperties)

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  dev
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

nu              [0 2 -1 0 0 0 0] 0.01;

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

Użyty w przykładzie solwer (**icoFoam**) wymaga podania jedynie kinematycznego współczynnika lepkości modelowanego płynu [$m^2/s$]. Współczynnik ten definiuje się w słowniku **transportProperties** znajdującym się w katalogu **constant**.

# Cavity – parametry symulacji (controlDict)

```
application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1.0;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Nazwa użytego solwera. Nazwa ta jest używana podczas uruchamiania obliczeń z poziomu terminala systemu operacyjnego.

**icoFoam** – niestacjonarne i nieściśliwe przepływy laminarne cieczy niutonowskich.

**UWAGA**: słowo „solwer” może oznaczać dwie rzeczy:

- program, który się uruchamia aby przeprowadzić obliczenia (tu. **icoFoam**);
- metodę rozwiązywania układu równań liniowych.

# Cavity – parametry symulacji (controlDict)

```
application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1.0;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Informacja o czasie rozpoczęcia obliczeń (lub w przypadku solwerów stacjonarnych o numerze iteracji).

W tym przypadku wskazany jest konkretny czas równy 0. Oznacza to, że podczas uruchamiania obliczeń wartości niezbędnych pól skalarnych i wektorowych pobrane zostaną z podkatalogu o nazwie 0.

Możliwe opcje to:

**firstTime** – rozpoczęcie od najwcześniejszego możliwego czasu dostępnego w modelu;

**startTime** – rozpoczęcie od kroku wskazanego zmienną **startTime**;

**latestTime** – kontynuacja obliczeń.

# Cavity – parametry symulacji (controlDict)

```
application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1.0;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Informacja o czasie zakończenia obliczeń (lub w przypadku solwerów stacjonarnych o numerze iteracji).

W tym przypadku modelowana będzie 1 sekunda przepływu (**icoFoam** to solwer niestacjonarny, więc liczby wskazane jako **startTime** oraz **endTime** oznaczają czas).

Możliwe opcje to:

**endTime** – zakończenie po osiągnięciu czasu wskazanego zmienną **endTime**;

**writeNow** – zakończenie bieżącego kroku i zapisanie danych;

**noWriteNow** – zakończenie bieżącego kroku bez zapisu danych;

**nextWrite** – zakończenie symulacji po osiągnięciu najbliższego punktu zapisu danych, określonego zmienną **writeControl**.

# Cavity – parametry symulacji (controlDict)

```
application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1.0;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Krok czasowy symulacji.

Wartość kroku czasowego musi zapewniać spełnienie warunku CFL < 1.

**Warunek Couranta–Friedrichsa–Lewy'ego** (warunek CFL) – warunek zbieżności numerycznych metod rozwiązywania pewnych równań różniczkowych cząstkowych (zwłaszcza równań hiperbolicznych). Pojawia się przy analizie stabilności jawnych metod numerycznych dla zagadnień zależnych od czasu (lub równoważnych im):

$$CFL = v\frac{dt}{dx} < 1$$

v – prędkość
dt – krok czasowy
dx – minimalny krok przestrzenny

Wartość liczby CFL wyświetlana jest podczas obliczeń – jeżeli obliczenia przerwą się, to należy sprawdzić, czy warunek jest spełniony.

# Cavity – parametry symulacji (controlDict)

```
application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1.0;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Definicja częstotliwości zapisywania danych.

W tym przypadku użyto opcji **timeStep**, co oznacza, że dane będą zapisywane co 20 kroków czasowych. Liczbę kroków czasowych pomiędzy kolejnymi zapisami danych określa zmienna **writeInterval**.

Możliwe opcje to:

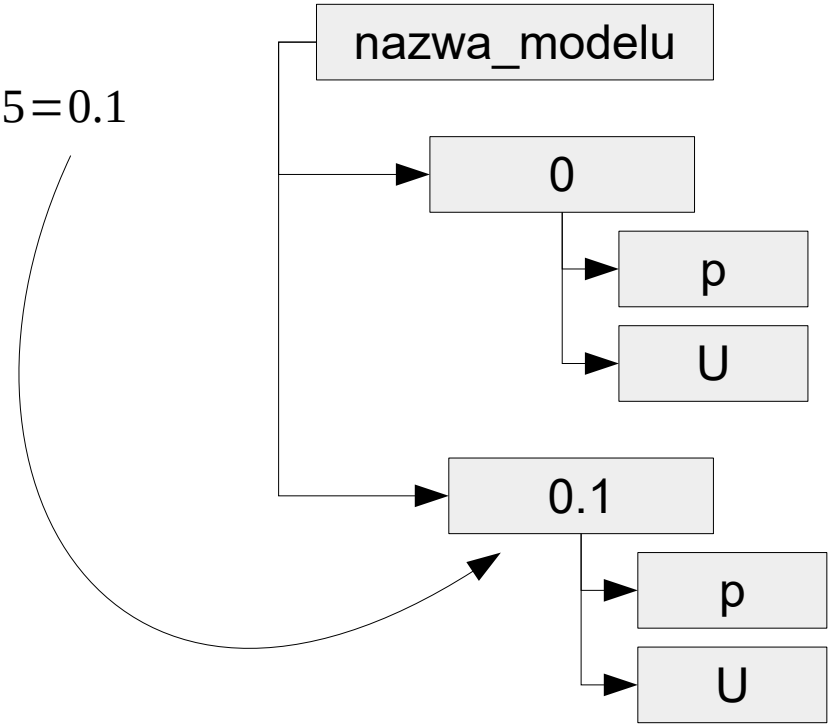
**timeStep** – zapis co n kroków czasowych;
**runTime** – zapis co n sekund;
**adjustableRunTime** – wariant dla obliczeń ze zmiennym krokiem czasowym;
**cpuTime** – zapis zależny od czasu procesora;
**clockTime** – zapis zależny od czasu rzeczywistego.

# Cavity – parametry symulacji (controlDict)

```
application     icoFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         1.0;
deltaT          0.005;
writeControl    timeStep;
writeInterval   20;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
```

Definicja maksymalnej liczby podkatalogów czasowych w modelu. Jeśli symulacja osiągnie zadeklarowaną liczbę katalogów, a obliczenia będzie przebiegać dalej, to w takim przypadku najstarsze katalogi będą na bieżąco kasowane. Wartość 0 oznacza, że nie ma ograniczenia, co do liczby katalogów.

$20 \cdot 0.005 = 0.1$



# Cavity – parametry symulacji (controlDict)

```
application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1.0;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Definicja rodzaju plików wynikowych. Możliwe opcje to **ascii** lub **binary**.

Definicja formatu zapisu wartości pól skalarnych i wektorowych – domyślnie jest 6 pozycji za znakiem separatora.

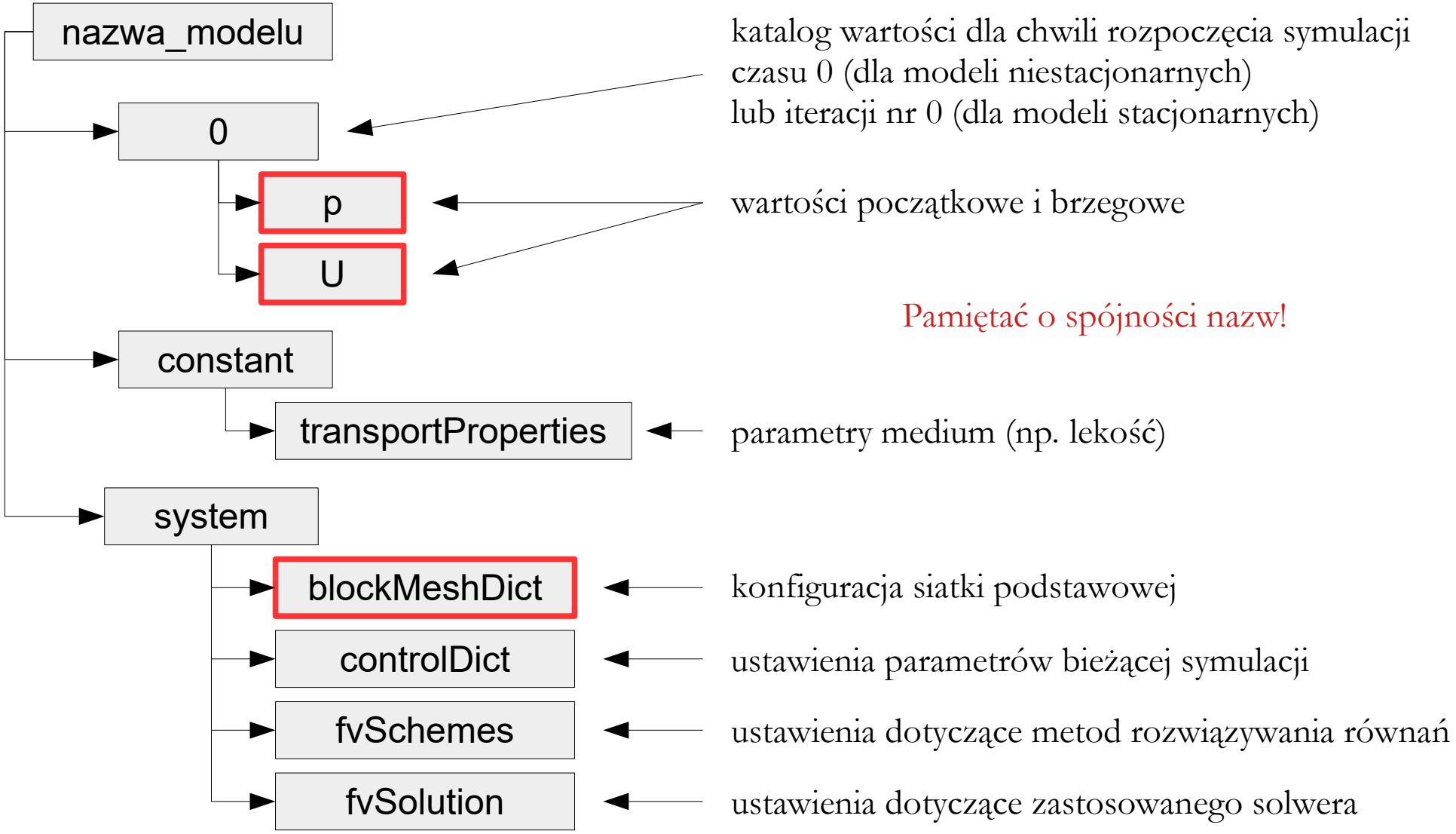
Definicja kompresji danych wynikowych. Możliwe opcje to: **uncompressed** oraz **compressed**. W drugim przypadku zastosowany będzie format gzip.

Definicja formatu zapisu czasu. Możliwe opcje to: **fixed** (postać zwykła), **scientific** (postać wykładnicza) oraz **general** (postać zwykła lub wykładnicza, w zależności od wartości).

Definicja formatu zapisu wartości czasu – domyślnie jest 6 pozycji za znakiem separatora.

Definicja, czy słowniki mają być zawsze wczytywane na początku każdego kroku czasowego. Możliwe opcje to **true** oraz **false**.

# OpenFOAM – struktura katalogów



# Cavity – schematy numeryczne (fvSchemes)

```
ddtSchemes
{
    default         Euler;
}

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

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

laplacianSchemes
{
    default         Gauss linear orthogonal;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;
}
```

Definicja schematów numerycznych odpowiedzialnych za rozwiązywanie poszczególnych członów równań. Ustawienia zależą od solwera użytego w danym przypadku.

**ddtSchemes** – pierwsze pochodne po czasie

**gradSchemes** – gradient $\nabla$

**divSchemes** – dywergencja $\nabla\cdot$

**laplacianSchemes** – laplasjan $\nabla^2$

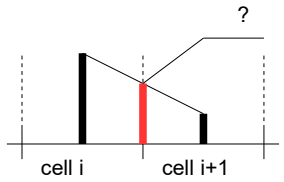
**interpolationSchemes** – interpolacja wartości pola na granicy komórek

**snGradSchemes** – interpolacja wartości normalnej gradientu pola na granicy komórek

# Cavity – schematy numeryczne (fvSchemes)

| Centred schemes | |
|---|---|
| `linear` | Linear interpolation (central differencing) |
| `cubicCorrection` | Cubic scheme |
| `midPoint` | Linear interpolation with symmetric weighting |
| **Upwinded convection schemes** | |
| `upwind` | Upwind differencing |
| `linearUpwind` | Linear upwind differencing |
| `skewLinear` | Linear with skewness correction |
| `filteredLinear2` | Linear with filtering for high-frequency ringing |
| **TVD schemes** | |
| `limitedLinear` | limited linear differencing |
| `vanLeer` | van Leer limiter |
| `MUSCL` | MUSCL limiter |
| `limitedCubic` | Cubic limiter |
| **NVD schemes** | |
| `SFCD` | Self-filtered central differencing |
| `Gamma` $\psi$ | Gamma differencing |

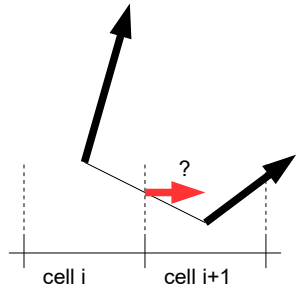
**interpolationSchemes** – schematy służące do obliczania wartości pola na granicy dwóch sąsiednich komórek na podstawie wartości tego pola w ich wnętrzach.




Schematy odnoszące się do zjawisk konwekcyjnych wymagają podania pola, względem którego odbywa się interpolacja. Najczęściej jest to **phi**, czyli strumień prędkości.

# Cavity – schematy numeryczne (fvSchemes)

| Scheme | Description |
|---|---|
| `corrected` | Explicit non-orthogonal correction |
| `uncorrected` | No non-orthogonal correction |
| `limited` $\psi$ | Limited non-orthogonal correction |
| `bounded` | Bounded correction for positive scalars |
| `fourth` | Fourth order |




**snGradSchemes** – schematy służące do obliczania gradientu pola (w kierunku normalnym do powierzchni) na granicy dwóch sąsiednich komórek na podstawie wartości gradientu tego pola w ich wnętrzach.

# Cavity – schematy numeryczne (fvSchemes)

| Discretisation scheme | Description |
|---|---|
| `Gauss <interpolationScheme>` | Second order, Gaussian integration |
| `leastSquares` | Second order, least squares |
| `fourth` | Fourth order, least squares |
| `cellLimited <gradScheme>` | Cell limited version of one of the above schemes |
| `faceLimited <gradScheme>` | Face limited version of one of the above schemes |

**gradSchemes** – schematy dyskretyzacji członów gradientowych.

Najczęściej najlepszym wyborem jest schemat „Gauss linear”, np.:

grad(p) Gauss linear; $\nabla(p)$

# Cavity – schematy numeryczne (fvSchemes)

| Scheme | Numerical behaviour |
|---|---|
| `linear` | Second order, unbounded |
| `skewLinear` | Second order, (more) unbounded, skewness correction |
| `cubicCorrected` | Fourth order, unbounded |
| `upwind` | First order, bounded |
| `linearUpwind` | First/second order, bounded |
| `QUICK` | First/second order, bounded |
| TVD schemes | First/second order, bounded |
| `SFCD` | Second order, bounded |
| NVD schemes | First/second order, bounded |

**divSchemes** – schematy dyskretyzacji członów zawierających operator dywergencji.

W przypadku dywergencji stosuje się schemat Gaussa ze wskazaniem metody interpolacji (dostępne opcje w tabeli powyżej), np.

div(phi,U) Gauss upwind; $\nabla\cdot(\rho U U)$ $\phi=\rho U$

# Cavity – schematy numeryczne (fvSchemes)

| Scheme | Numerical behaviour |
|---|---|
| `corrected` | Unbounded, second order, conservative |
| `uncorrected` | Bounded, first order, non-conservative |
| `limited` $\psi$ | Blend of `corrected` and `uncorrected` |
| `bounded` | First order for bounded scalars |
| `fourth` | Unbounded, fourth order, conservative |

**laplacianSchemes** – schematy dyskretyzacji członów zawierających operator Laplace'a.

W przypadku operatora Laplace'a stosuje się schemat Gaussa ze wskazaniem metody interpolacji oraz metody obliczania gradientu normalnego na granicy komórek (dostępne opcje w tabeli powyżej), np.:

laplacian(nu,U) Gauss linear corrected; $\nabla \cdot (\nu \nabla U)$

# Cavity – schematy numeryczne (fvSchemes)

| Scheme | Description |
|---|---|
| `Euler` | First order, bounded, implicit |
| `localEuler` | Local-time step, first order, bounded, implicit |
| `CrankNicholson` $\psi$ | Second order, bounded, implicit |
| `backward` | Second order, implicit |
| `steadyState` | Does not solve for time derivatives |

**timeSchemes** – schematy dyskretyzacji pierwszych pochodnych po czasie ( $\partial/\partial t$ ).

Współczynnik $\psi$ oznacza wagę między schematem Cranka-Nicholsona ( $\psi=1$ )
a schematem Eulera ( $\psi=0$ ).

Jeżeli schemat CrankNicholson jest niestabilny,
to należy zmniejszać wartość współczynnika wagowego.

Drugie pochodne po czasie mogą być definiowane w słowniku d2dt2Schemes,
ale wówczas i tak dostępny jest jedynie schemat Eulera.

# Cavity – parametry solwera (fvSolution)

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Szczegóły dotyczące solwerów rozwiązujących układy równań liniowych (tu słowo „solwer” nie oznacza nazwy programu, jak było wcześniej): $A x = B$.

Możliwe opcje to:

**PCG** – wstępnie kondycjonowany (bi-) sprzężony gradient dla macierzy symetrycznych;

**PbiCG** – wstępnie kondycjonowany (bi-) sprzężony gradient dla macierzy symetrycznych;

**PbiCGStab** – wstępnie kondycjonowany stabilizowany (bi-) sprzężony gradient dla dowolnych macierzy (zalecany zamiast PCG/PbiCG);

**smoothSolver** – metoda wygładzania funkcji;

**GAMG** – uogólniona wialosiatkowa metoda geometryczno-algebraiczna;

**diagonal** – solwer diagonalny dla metod jawnych.

# Cavity – parametry solwera (fvSolution)

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Wskazanie metody wstępnego kondycjonowania macierzy współczynników rozwiązywanego układu równań.

Możliwe opcje to:

**DIC** – niepełny algorytm Choleskiego (dla macierzy symetrycznych);

**FDIC** – przyspieszony niepełny algorytm Choleskiego (DIC z buforowaniem) (dla macierzy symetrycznych);

**DILU** – diagonalna niepełna metoda LU (dla macierzy niesymetrycznych);

**GAMG** – uogólniona wialosiatkowa metoda geometryczno-algebraiczna;

**diagonal** – solwer diagonalny dla metod jawnych

**none** – bez wstępnego kondycjonowania.

# Cavity – parametry solwera (fvSolution)

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Definicja metody wygładzania funkcji.

Możliwe opcje to:

**GaussSeidel** – metoda Gaussa-Seidela;
**DIC** – niepełny algorytm Choleskiego (dla macierzy symetrycznych);
**DICGaussSeidel** – połączenie niepełnego algorytmu Choleskiego z metodą Gaussa-Seidela.

# Cavity – parametry solwera (fvSolution)

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Parametry zbieżności procesu iteracyjnego.

Obliczenia zostaną zakończone, jeśli będzie spełniony jeden z trzech warunków:

- reszta spadnie poniżej kryterium zbieżności zdefiniowanego zmienną **tolerance** (parametr określa poziom dokładności wyniku);
- stosunek bieżącej wartości reszty i reszty początkowej spadnie poniżej wartość zdefiniowaną zmienną **relTol** (parametr określa względną poprawę rozwiązania obecnego w stosunku do rozwiązania początkowego). Jeśli wartość zmiennej **relTol** wynosi 0, to obowiązuje warunek pierwszy, czyli kryterium zbieżności;
- liczba iteracji osiągnie wartość określoną w zmiennej **maxIter** (tu nie występuje, bo **icoFoam** to solwer niestacjonarny).

# Cavity – parametry solwera (fvSolution)

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Ustawienia dotyczące ostatniej iteracji, w której wyliczane jest pole ciśnień (liczbę dodatkowych przebiegów pętli, służących do poprawienia wartości ciśnień zdefiniowane są w zmiennej **nCorrectors** w sekcji PISO). Ustawienia są takie same jak dla ciśnienia (słownik **p**), ale inna jest wartość parametru **relTol**.

# Cavity – parametry solwera (fvSolution)

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}
```

```
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Ustawienia dotyczące metody PISO:

**nCorrectors** – liczba korekt pola ciśnień (minimalnie 2, maksymalnie 4);

**nonOrthogonalcorrectors** – dodatkowa korekta uwzględniająca nieortogonalność siatki (maksymalnie 20);

**pRefCell** – numer komórki, w której zadana ma być referencyjna wartość ciśnienia;

**pRefValue** – referencyjna wartość ciśnienia.

W solwerach nieściśliwych wyliczane jest ciśnienie względne (a właściwie ciśnienie podzielone przez gęstość!).

Wartość referencyjna używana jest tylko wówczas, gdy na danym brzegu nie zdefiniowano konkretnej wartości ciśnienia (nie zdefiniowano **fixedValue**).

# Cavity – parametry solwera (fvSolution)

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Szczegóły dotyczące zastosowanego solwera podejrzeć można w plikach źródłowych. W przypadku solwera **icoFoam** jest to plik „**icoFoam.C**” (wraz z plikiem nagłówkowym „**createFields.H**”), znajdujący się w podkatalogu **/applications/solvers/incompressible/pisoFoam**.

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
);
```

Układ równań rozwiązywany przez solwer PISO.

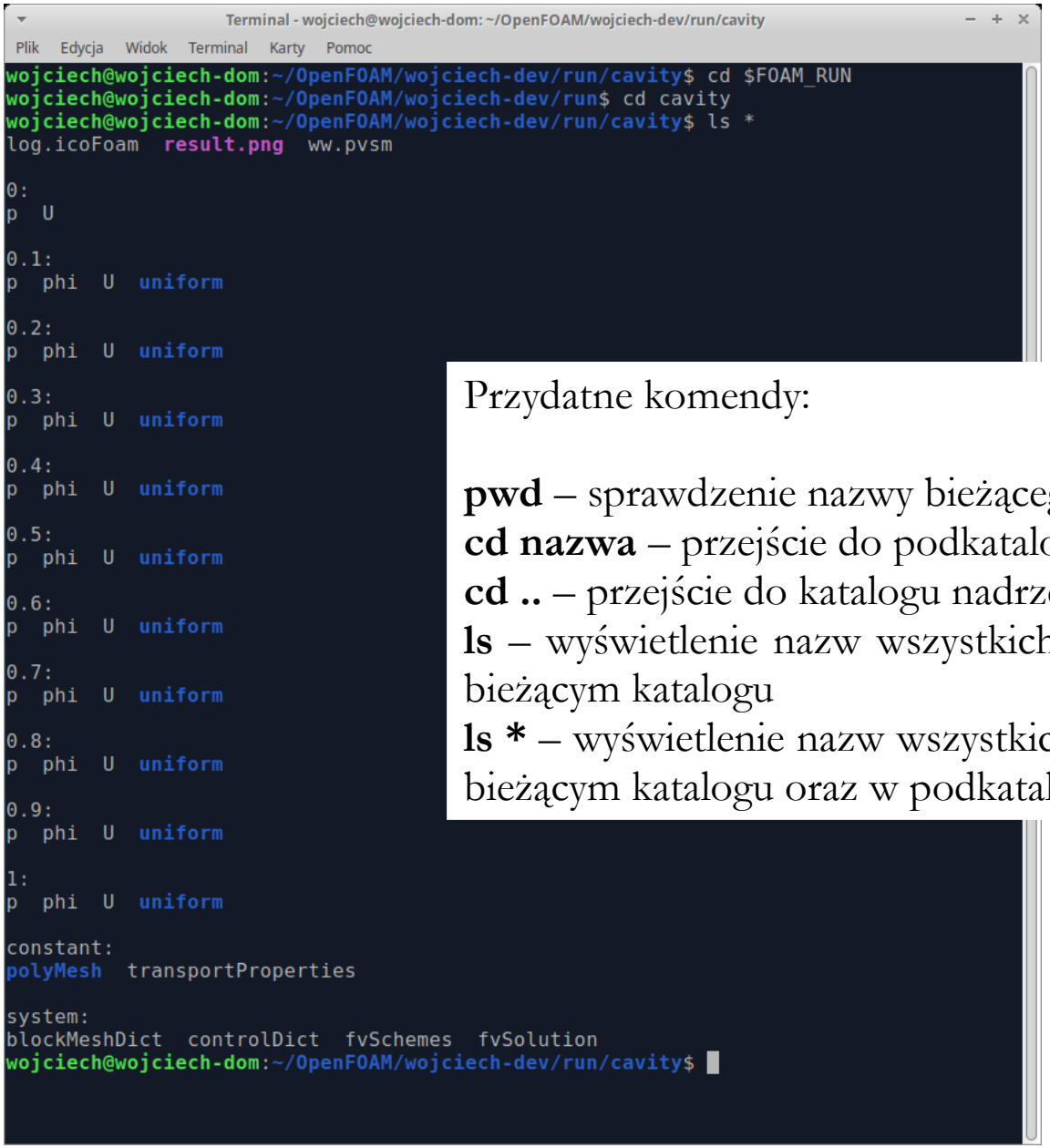
# Cavity – uruchomienie przykładu

**Uruchomienie przykładu**:

- skopiuj katalog **cavity** z podkatalogu **tutorials** (OpenFOAM instaluje się standardowo w katalogu **/opt/openfoam**) do katalogu roboczego (który należy wcześniej stworzyć, zgodnie z instrukcjami dostępnymi w Internecie, np. na stronie podanej na dole slajdu)
- otwórz terminal (np. używając prawego klawisza myszy)
- przejdź do katalogu roboczego: **cd $FOAM_RUN**
- przejdź do katalogu przykładu: **cd cavity**
- wygeneruj siatkę: **blockMesh**
- uruchom program obliczeniowy: **icoFoam**
- po zakończeniu obliczeń uruchom program ParaView: **paraFoam**
- wykonaj wizualizację wyników

Opis instalacji i konfiguracji: https://openfoam.org/download/6-ubuntu/

# Cavity – uruchomienie przykładu

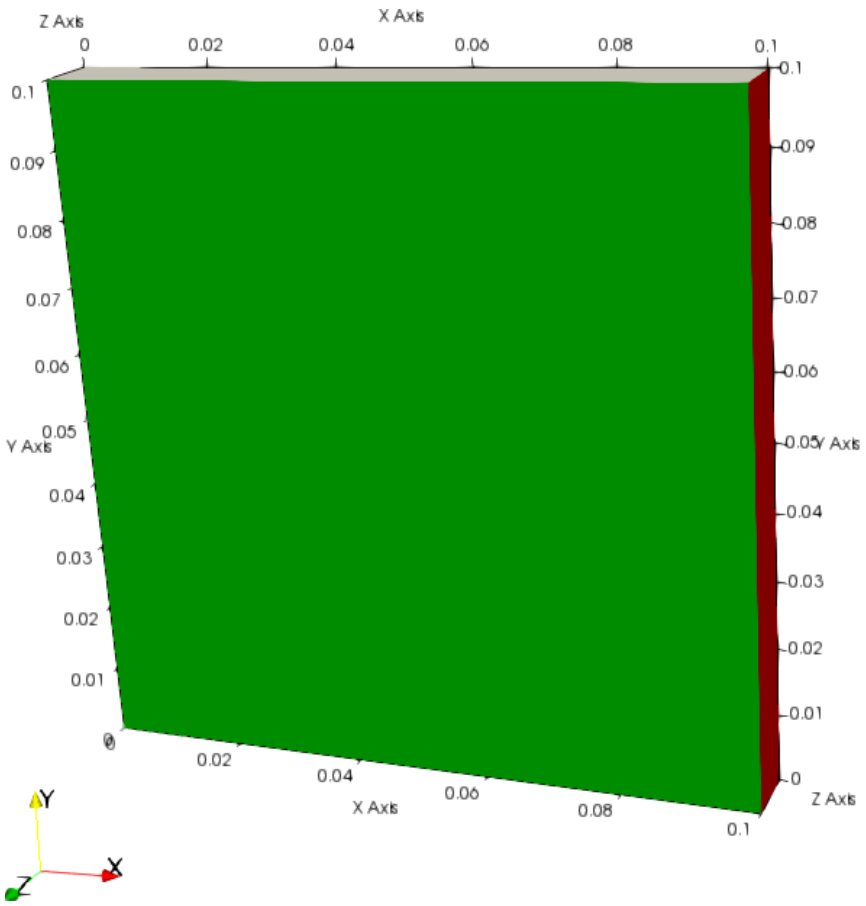



Terminal systemu xUbuntu

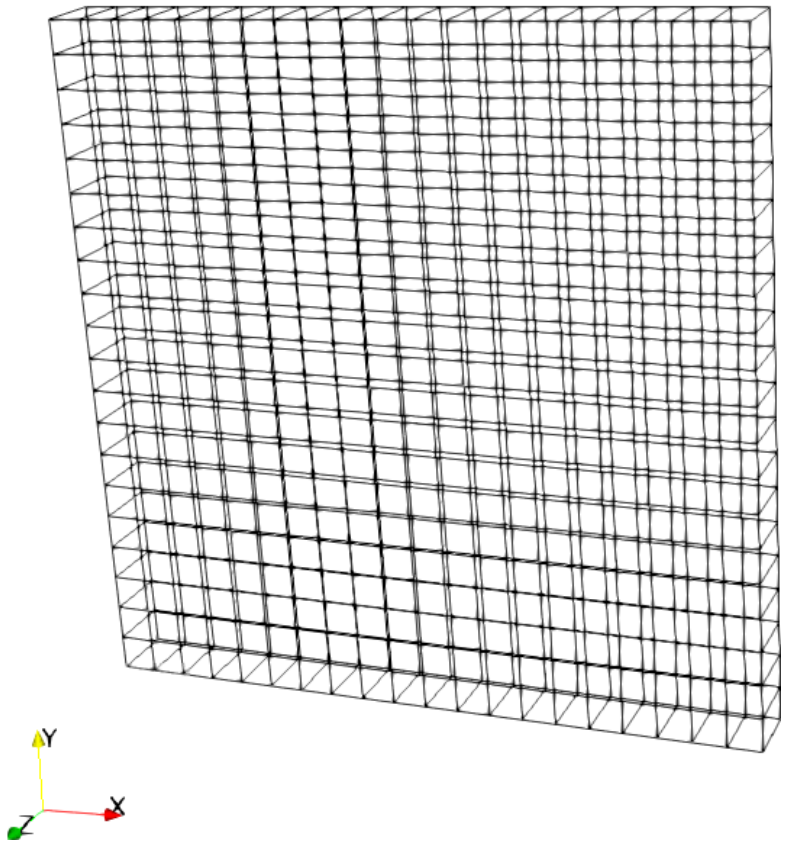
Przydatne komendy:

**pwd** – sprawdzenie nazwy bieżącego katalogu
**cd nazwa** – przejście do podkatalogu o wskazanej nazwie
**cd ..** – przejście do katalogu nadrzędnego
**ls** – wyświetlenie nazw wszystkich plików i katalogów znajdujących się w bieżącym katalogu
**ls *** – wyświetlenie nazw wszystkich plików i katalogów znajdujących się w bieżącym katalogu oraz w podkatalogach

# Cavity – wyniki symulacji

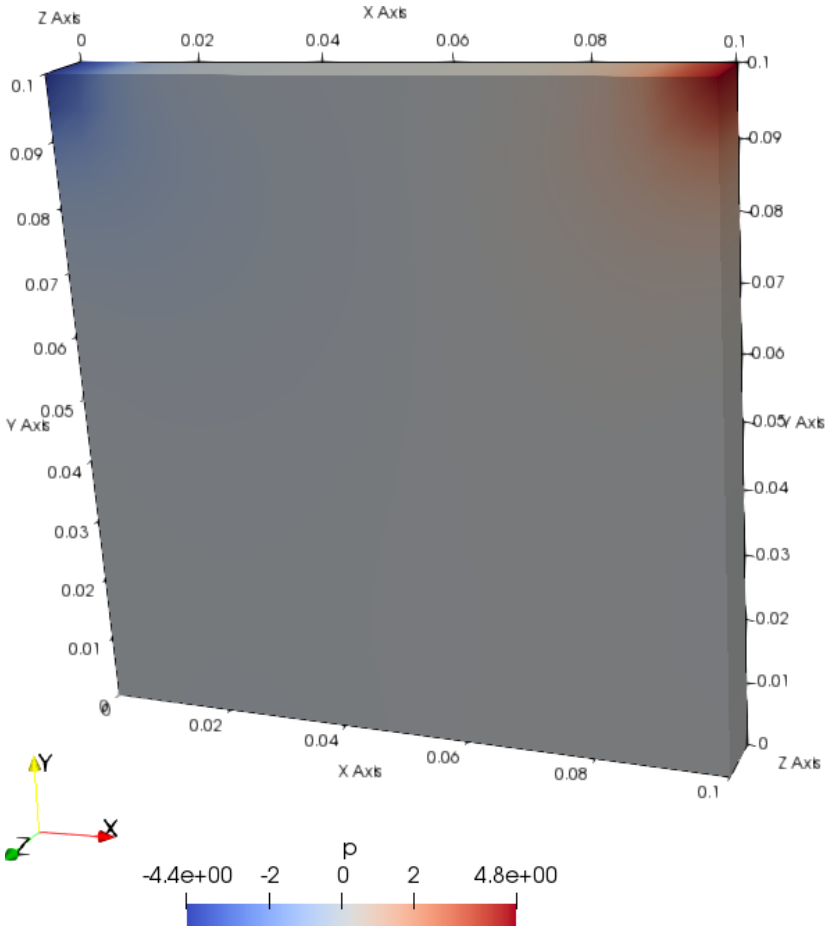



Geometria zagadnienia

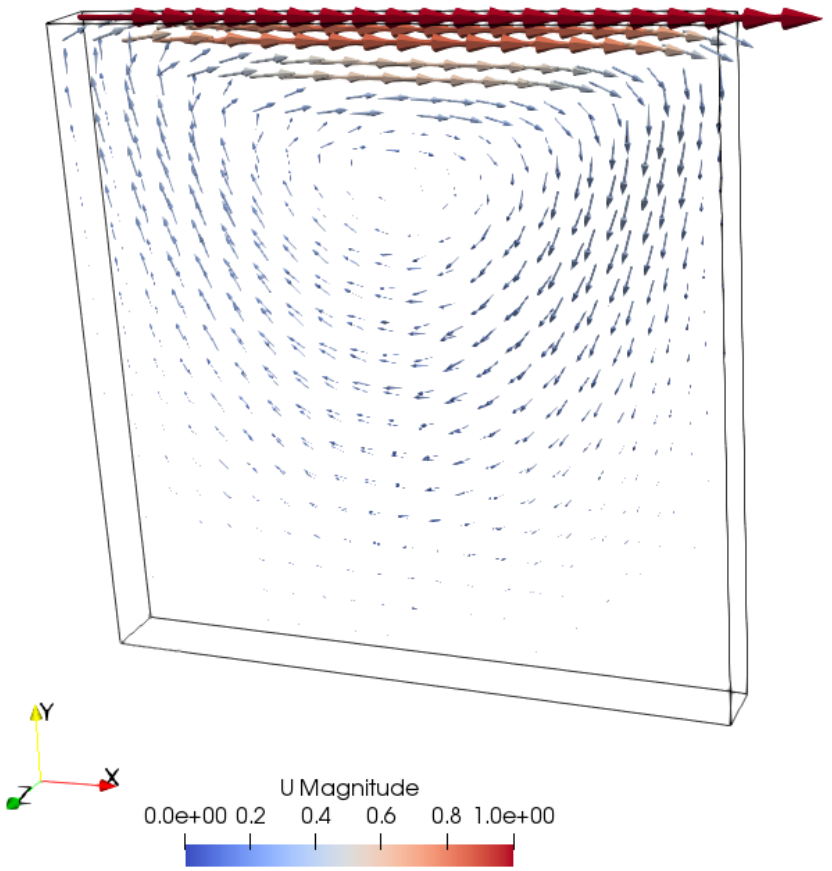



Siatka numeryczna
(efekt działania programu blockMesh)

# Cavity – wyniki symulacji

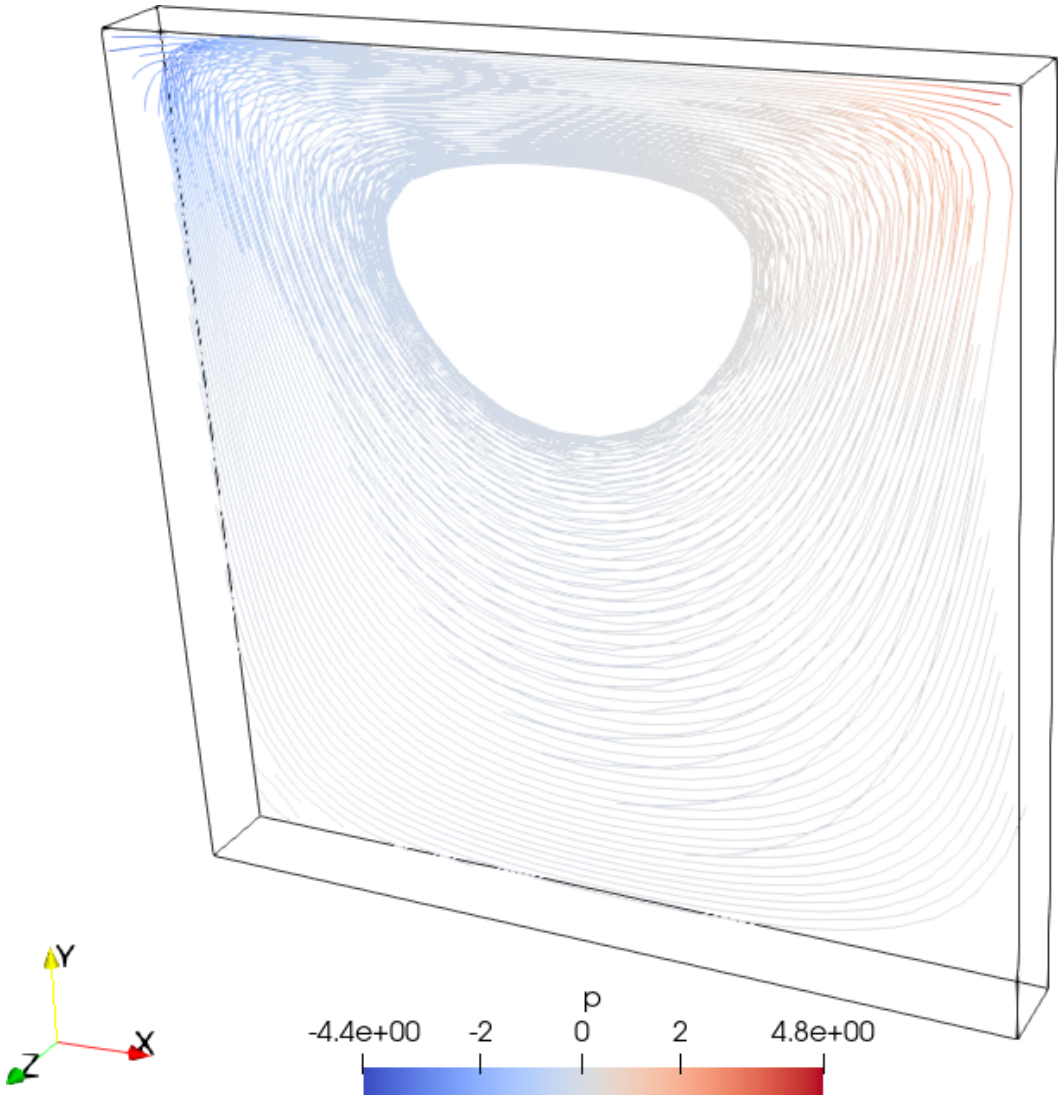


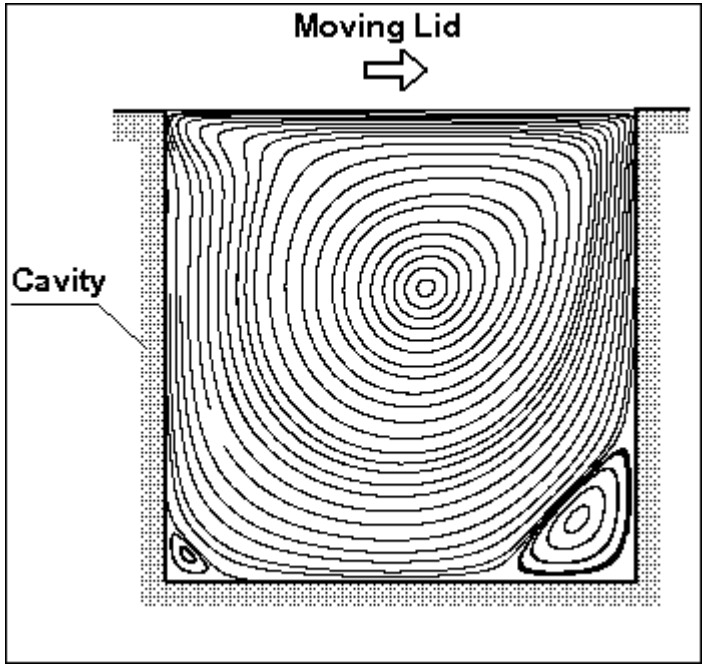

Pole ciśnień




Pole prędkości

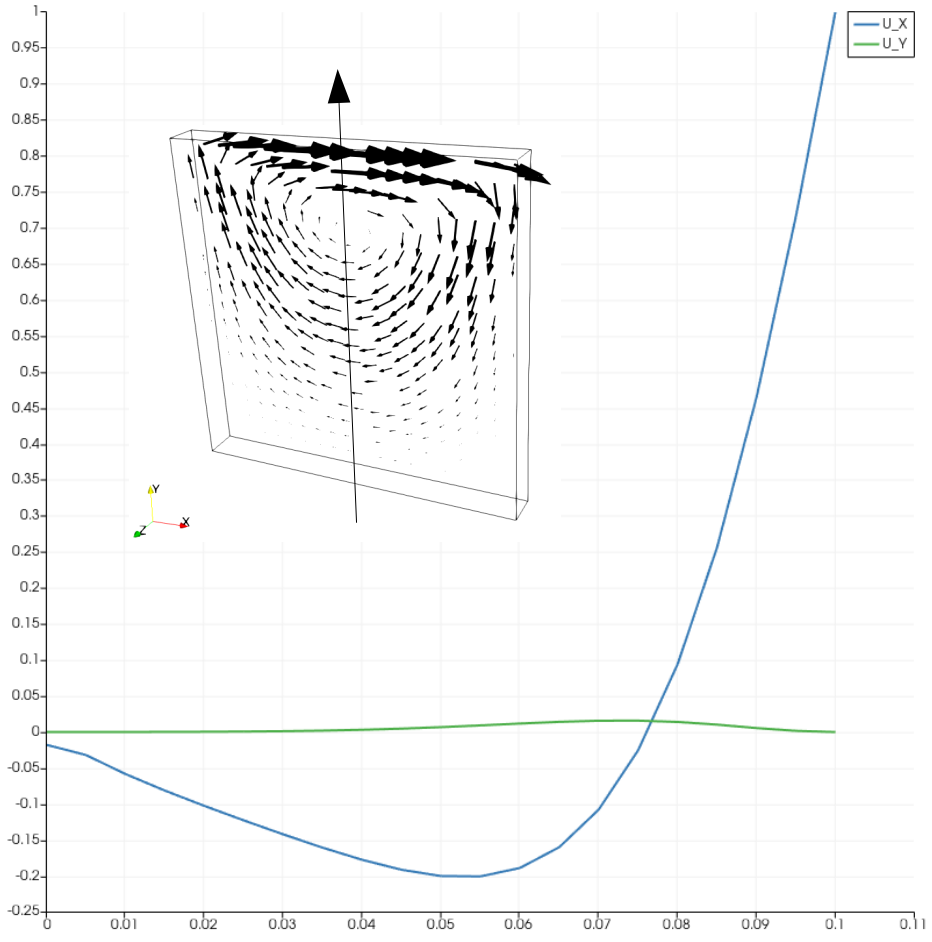
# Cavity – wyniki symulacji



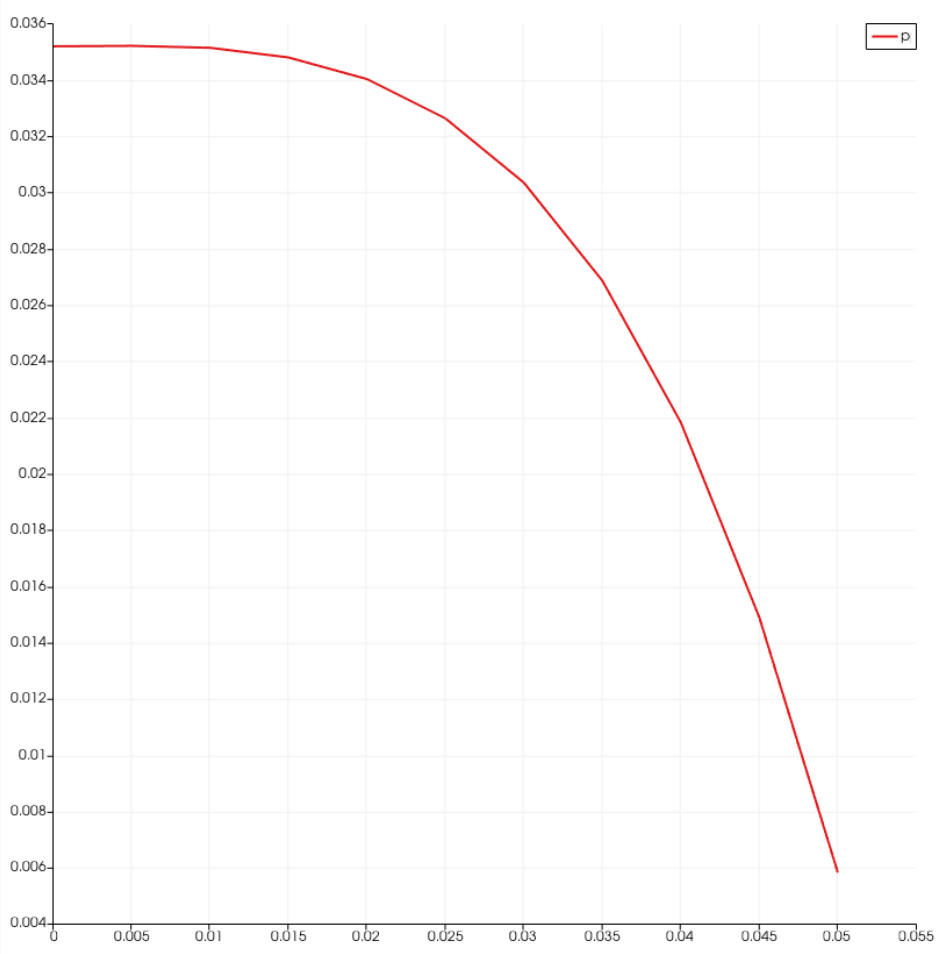




Linie prądu

# Cavity – wyniki symulacji




Wykres prędkości wzdłuż zaznaczonej linii




Wykres ciśnienia wzdłuż zaznaczonej linii

Wydział Nauk Technicznych

UNIVERSITY OF WARMIA AND MAZURY IN OLSZTYN
The Faculty of Technical Sciences
POLAND, 10-957 Olsztyn, M. Oczapowskiego 11
tel.: (48)(89) 5-23-32-40, fax: (48)(89) 5-23-32-55
URL: http://www.uwm.edu.pl/edu/sobieski/ (in Polish)

# Dziękuję za uwagę

Publikacja została napisana w wyniku odbywania przez autora stażu w Holandii, współfinansowanego przez Unię Europejską w ramach Europejskiego Funduszu Społecznego (Program Operacyjnego Wiedza Edukacja Rozwój), zrealizowanego w projekcie Program Rozwojowy Uniwersytetu Warmińsko-Mazurskiego w Olsztynie (POWR.03.05.00-00-Z310/17).

**Wojciech Sobieski**

Utrecht, 2019