Implement a parabolicVelocity fixedValue boundary condition

(This page is part of the course 'CFD with OpenSource Software', by Håkan Nilsson, see http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/#YEAR_2019)

The implementations of the boundary conditions are located in:

```
$FOAM_SRC/finiteVolume/fields/fvPatchFields
```

The standard procedure when implementing a new boundary condition is to find one that is already implemented and does almost what you want, copy that to your user directory and do the required modifications (change names of folder, files, class, member functions and member data).
We will not do that here, since we will build a boundary condition more or less from scratch.

Initialize and prepare

Open a new terminal window and source the most recent OpenFOAM version.

Start by creating directory and template files. We will first do this as a separate library and later merge it into our own finiteVolume library.

```
mkdir $WM_PROJECT_USER_DIR/src
cd $WM_PROJECT_USER_DIR/src
foamNewBC -f -v parabolicVelocity
```

(check 'foamNewBC -help' for a description and more options)

This creates the following templated structure:

```
parabolicVelocity
├── Make
│   ├── files
│   └── options
├── parabolicVelocityFvPatchVectorField.C
└── parabolicVelocityFvPatchVectorField.H
```

These are just templates, so we need to do some modifications.

Modify the parabolicVelocityFvPatchVectorField.H file (declarations)

Change copyright statement:

```
sed -i s/'AUTHOR,AFFILIATION'/'Hrvoje Jasak, Wikki Ltd.'/g parabolicVelocity/parabolicVelocityFvPatchVectorFi
eld.H
```

This is based on an **implementation by Hrv in FOAM-extend** - if you implement something of your own you give your name and affiliation.

Open the file parabolicVelocity/parabolicVelocityFvPatchVectorField.H using an editor.

Change the Description to:

```
Description
    Boundary condition specifies a parabolic velocity inlet profile
    (fixed value), given maximum velocity value (peak of the parabola),
    flow direction n and direction of the parabolic coordinate y
```

The lines between '// Private data' and 'public:' are just examples of how member data can be constructed. You can thus look at them to learn how construct your member data. Now, simply replace all those lines with:

```
    //- Peak velocity magnitude
    scalar maxValue_;
    //- Flow direction
    vector n_;
    //- Direction of the y-coordinate
    vector y_;
```

This means that we will have three member data objects in our class. Later on we need to make sure that those are initialized in the constructors rather than those that we just deleted from the template.

Under '// Member functions', remove the autoMap and rmap functions (we won't bother about mapping at the moment, since this boundary condition will simply set the values at the boundary according to a mathematical function anyway), and instead add:

```
            //- Return max value
            scalar& maxValue()
            {
                return maxValue_;
            }
            //- Return flow direction
            vector& n()
            {
                return n_;
            }
            //- Return y direction
            vector& y()
            {
                return y_;
            }
```

These functions can be used both to return the values of the member data, and to change the values of the member data.

Done! Save the file!

Modify the parabolicVelocityFvPatchVectorField.C file (definitions)

Change copyright statement:

```
sed -i s/'AUTHOR,AFFILIATION'/'Hrvoje Jasak, Wikki Ltd.'/g parabolicVelocity/parabolicVelocityFvPatchVectorFi
eld.C
```

Open the file parabolicVelocity/parabolicVelocityFvPatchVectorField.C using an editor.

Remove the private member function t().

When we modified the .H file we removed the original member data and added three member data of our own. We now have to go through all the constructors and make sure that those member data are initialized instead of the original ones. Have a look at how it is done for similar types of the original lines.

In the initialization part of the first constructor (after ':' and before '{'), change to:

```
    fixedValueFvPatchVectorField(p, iF),
    maxValue_(0),
    n_(1, 0, 0),
    y_(0, 1, 0)
```

In the initialization part of the second constructor, change to:

```
    fixedValueFvPatchVectorField(p, iF),
    maxValue_(readScalar(dict.lookup("maxValue"))),
    n_(dict.lookup("n")),
    y_(dict.lookup("y"))
```

This particular constructor is used when the boundary condition is set through the dictionary file in the time directory. Our three member data are here given the values looked up in the dictionary file in the time directory.

In the curly brackets of the second constructor, ADD at the beginning (before the call of the evaluate function):

```
    Info << "Using the parabolicVelocity boundary condition" << endl;
    if (mag(n_) < SMALL || mag(y_) < SMALL)
    {
        FatalErrorIn("parabolicVelocityFvPatchVectorField(dict)")
            << "n or y given with zero size not correct"
            << abort(FatalError);
    }
    n_ /= mag(n_);
    y_ /= mag(y_);
```

These lines will be executed when an object of the class is constructed using that constructor. Watch out for the info-statement in the log file later. The rest of the lines make sure that the values/vectors of the n and y keywords in the boundary condition dictionary are set correctly, and that they are unit vectors.

In the initialization part of the third constructor, change to:

```
    fixedValueFvPatchVectorField(ptf, p, iF, mapper),
    maxValue_(ptf.maxValue_),
    n_(ptf.n_),
    y_(ptf.y_)
```

In the initialization part of the fourth constructor, change to:

```
    fixedValueFvPatchVectorField(ptf),
    maxValue_(ptf.maxValue_),
    n_(ptf.n_),
    y_(ptf.y_)
```

In the initialization part of the fifth constructor, change to:

```
    fixedValueFvPatchVectorField(ptf, iF),
    maxValue_(ptf.maxValue_),
    n_(ptf.n_),
    y_(ptf.y_)
```

Remove the autoMap and rmap member functions, as we also did in the .H file.

Change the entire definition of the updateCoeffs() function to:

```
    if (updated())
    {
        return;
    }

    // Get range and orientation
    boundBox bb(patch().patch().localPoints(), true);

    vector ctr = 0.5*(bb.max() + bb.min());

    const vectorField& c = patch().Cf();

    // Calculate local 1-D coordinate for the parabolic profile
    scalarField coord = 2*((c - ctr) & y_)/((bb.max() - bb.min()) & y_);

    vectorField::operator=(n_*maxValue_*(1.0 - sqr(coord)));
```

The updateCoeffs function is the main function, where we define the mathematical function for the velocity field at the patch. That is done by determining the extent of the patch (boundBox), checking up the center of the patch (ctr), identifying the centers of the faces on the patch (c), calculating a coordinates scalar field along the patch that is later used in the mathematical function (coord), and assigning a new definition of the operator '=' to the requested mathematical function. That operator is then used elsewhere in the code when assigning values to the patch faces. The values are thus not set at the faces right here, but when the operator is used for the object of this class.

Change the entire definition of the write() function to:

```
    fvPatchVectorField::write(os);
    os.writeKeyword("maxValue") << maxValue_ << token::END_STATEMENT << nl;
    os.writeKeyword("n") << n_ << token::END_STATEMENT << nl;
    os.writeKeyword("y") << y_ << token::END_STATEMENT << nl;
    writeEntry("value", os);
```

The write function makes sure that our boundary condition and the current face values are written in the time directories. The original lines can again be used to figure out how to write our own lines.

Done! Save the file!

Compile the library

```
wmake parabolicVelocity
```

Check that the libparabolicVelocity.so file ends up in your user directory.

Set up a case and test

```
run
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily pitzDailyParabolicInlet
```

Change the inlet boundary condition in 0/U to:

```
    inlet
    {
        type parabolicVelocity;
        n (1 0 0);
        y (0 1 0);
        maxValue 1;
    }
```

The contents of this entry must be in accordance with the constructor that reads the dictionary (see above). The vector 'n' is the direction of the flow. The vector 'y' is the coordinate direction of the profile. The scalar 'maxValue' is the centerline velocity.

Add a line at the end of the system/controlDict:

```
libs ("libparabolicVelocity.so");
```

This makes the solve aware of the new library, so that it can be used. I.e., this line must be added for all cases where the library is used, but no re-compilation is needed for any solver. The file libparabolicVelocity.so is found using the LD_LIBRARY_PATH environment variable that is set up by the OpenFOAM environment. Note that the solver itself will not be aware of the new class, so it will not show up using the command ldd `which simpleFoam`. The library is linked to at run-time (using dlopen).

Run the case:

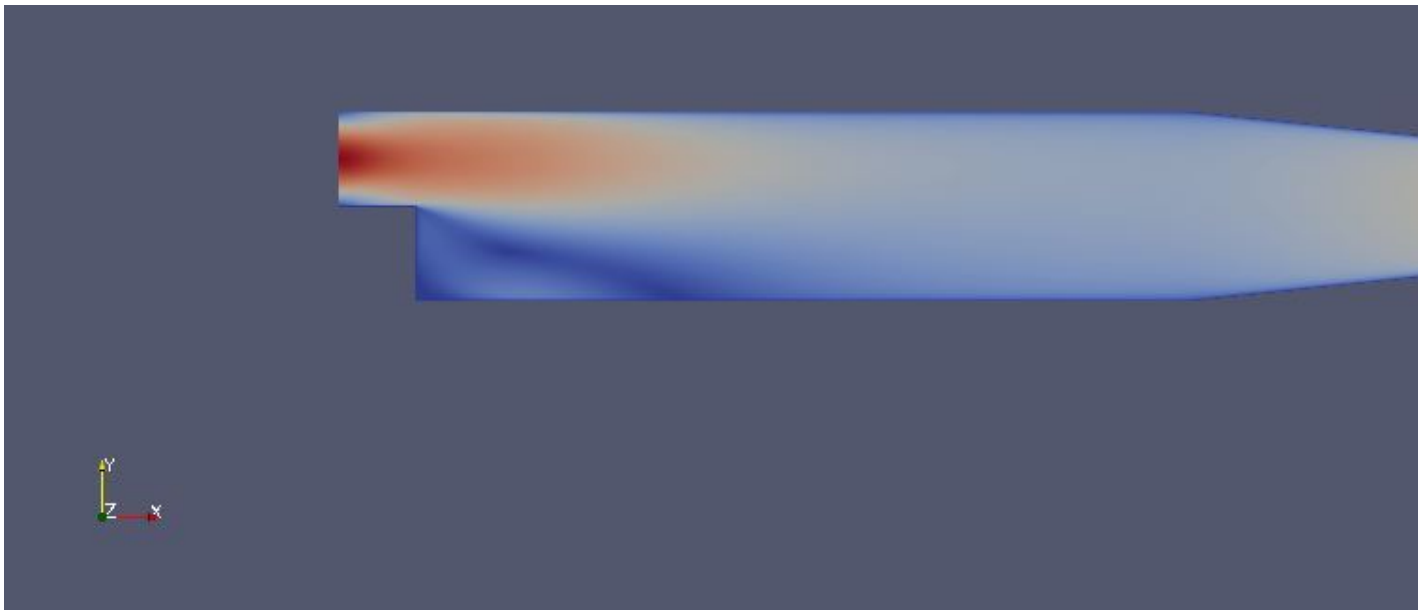
```
blockMesh -case pitzDailyParabolicInlet
simpleFoam -case pitzDailyParabolicInlet >& log&
```

Check for the info statement "Using the parabolicVelocity boundary condition" in the log file, to make sure that the boundary condition constructor is used.

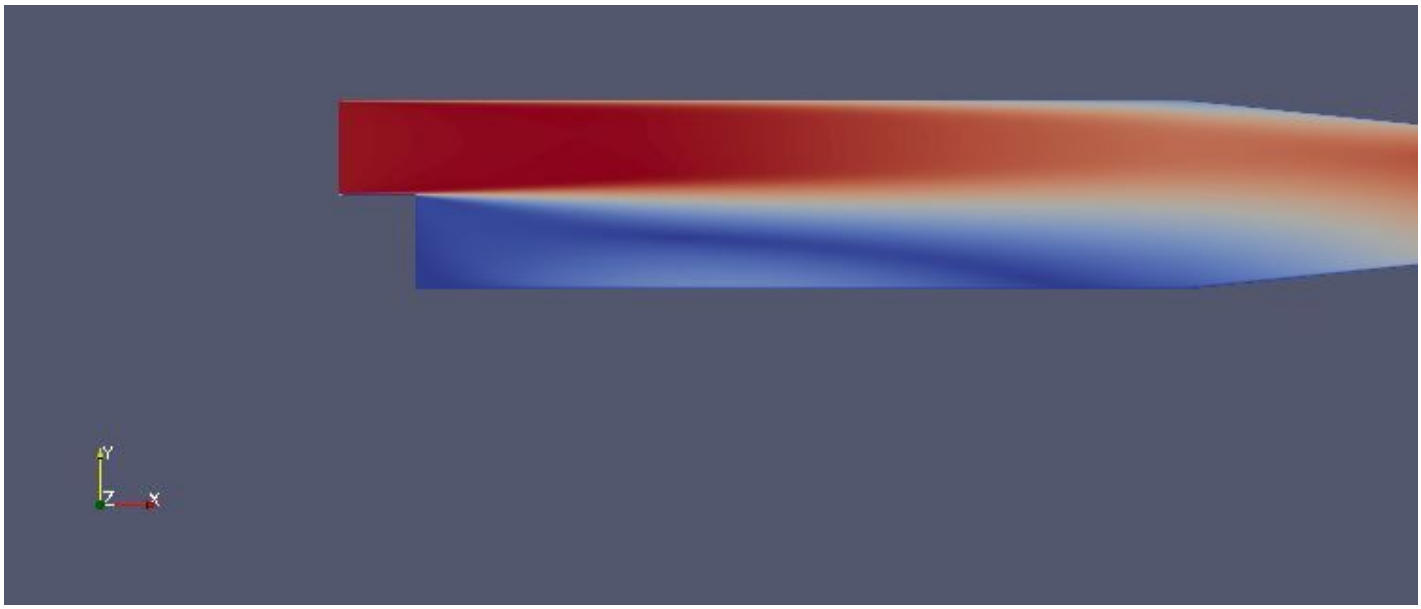
Open the case in paraFoam and go to the final time step and color by velocity. If you now also get the info statement "Using the parabolicVelocity boundary condition" in the terminal window, it means that also paraFoam recognizes the new boundary condition. Then you can also see the velocity distribution at the inlet if you color by interpolated values of the velocity, already at time zero. That is not the case if you did not compile ParaView and the OpenFOAM readers, or if you are running paraFoam with the -builtin flag. There is in fact a risk that paraFoam will crash. If so, add a dummy value entry to the inlet boundary condition in the zero folder, or rename the zero folder so that it is not recognized by paraFoam as a time step. For the OpenFOAM reader in OpenFOAM-v1706+ (and later) the zero directory is by default not

read, which eliminates this problem. In previous versions of ParaView you could also tick Skip Zero Time before clicking on Apply.

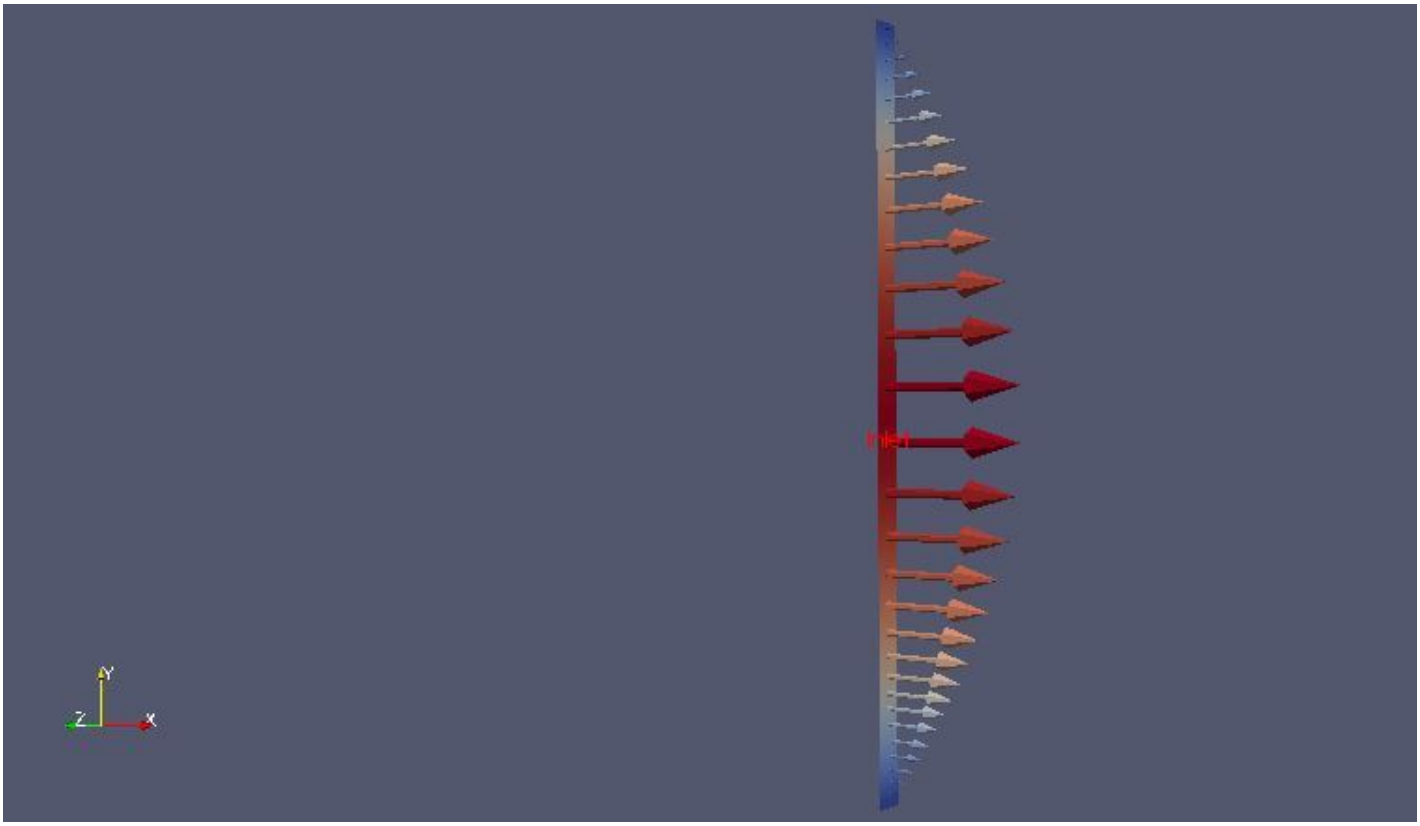
Velocity field of the pitzDailyParabolicInlet case (parabolic inlet profile with maximum value of 1m/s):



Original velocity field of the pitzDaily case (constant 10m/s at the inlet):



Choose only the Inlet patch under Mesh Parts. Use the cellCenters filter and the glyphs filter to show the vectors at the patch:



Move the boundary condition into libmyFiniteVolume.so

When you are happy with your implementation you may want to add it to your own version of a finiteVolume library, with the same organization as the main installation. For that you have to move the parabolicVelocity directory into a path that is the same as in the main installation, but in your working directory.

```
cd $WM_PROJECT_USER_DIR/src
mkdir -p finiteVolume/fields/fvPatchFields/derived
cp -r parabolicVelocity finiteVolume/fields/fvPatchFields/derived
```

If you look in the main installation, $WM_PROJECT_DIR/src/finiteVolume, you find the Make directory at that location. It means that all directories (classes) in that directory will be added to the same library. If we should do the same here we should make sure that we have a Make directory at that location, and we have to make sure that it points out all the files that should be compiled.

If you don't already have a Make directory in your finiteVolume directory, move the one from the directory you just created:

```
mv finiteVolume/fields/fvPatchFields/derived/parabolicVelocity/Make finiteVolume
```

If you do already have a Make directory in your finiteVolume directory, remove the one from the directory you just created:

```
rm -r finiteVolume/fields/fvPatchFields/derived/parabolicVelocity/Make
```

Continue, in both cases, by making sure that you add the following to the finiteVolume/Make/files file:

```
fields/fvPatchFields/derived/parabolicVelocity/parabolicVelocityFvPatchVectorField.C
```

and remove any other line containing parabolicVelocityFvPatchVectorField.C, if any. If you like, you can use a local variable named 'derivedFvPatchFields', as is done in $FOAM_SRC/finiteVolume/Make/files.

Make sure that the library name is libmyFiniteVolume, in finiteVolume/Make/files, and that the library is saved to $FOAM_USER_LIBBIN:

```
LIB = $(FOAM_USER_LIBBIN)/libmyFiniteVolume
```

compile by typing:

```
wmake finiteVolume
```

Since we have a new name for the library we have to change in the controlDict of the case:

```
libs ("libmyFiniteVolume.so");
```

Then run the case and enjoy.

License