# Implement a normalizedHelicity functionObject

This is a simple exercise to implement a functionObject, based on an existing one that does almost what we want.

The functionObjects are implemented in:

```
$FOAM_SRC/functionObjects
```

We will copy the field/vorticity functionObject and create a normalizedHelicity functionObject. The normalizedHelicity is the scalar product between the vorticity vector and the velocity vector, normalized by the product of their absolute values:

$$H_n = \frac{\vec{\omega}\cdot\vec{u}}{|\vec{\omega}||\vec{u}|}$$

The purpose of this measure is to find vortex cores. The magnitude of the normalized helicity tends to unity in the vortex core and its sign indicates the direction of the swirl of the vortex relative to the streamline velocity component. Vortices can then be visualized using iso-surfaces of values close to 1 or -1. Read more in *Y.Levy, D.Degani, and A.Seginer: Graphical visualization of vortical flows by means of helicity. AIAA Journal, 28(8):1347-1352, 1990*. Also in *M.Inoue, M.Furukawa, K.Saiki, and K.Yamada: Physical explanations of tip leakage flow field in an axial compressor rotor. ASME Paper 98-GT-91, 1998*. Note to myself (and others): Perhaps the vorticity should be the absolute vorticity and the velocity should be the relative velocity, if used in a rotating framework (see my licentiate thesis, which was a long time ago). Below we take them both as relative.

First make sure that there is not already a normalizedHelicity functionObject in the installation (if there is we should give our new one another name):

```
find $FOAM_SRC -iname "*normalizedHelicity*"
```

Start by copying the vorticity functionObject to your user directory (make sure that you read and understand each line when copy-pasting):

```
foam
cp -r --parents src/functionObjects/field/vorticity $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/functionObjects/field
mv vorticity normalizedHelicity
cd normalizedHelicity
sed -i 's/vorticity/normalizedHelicity/g' *
```

Rename the files (this command may differ for different OS's):
In Ubuntu:

```
rename 's/vorticity/normalizedHelicity/' *
```

otherwise, try:

```
rename vorticity normalizedHelicity *
```

or, safest but tedious if there are many files:

```
mv vorticity.H normalizedHelicity.H; mv vorticity.C normalizedHelicity.C
```

Base your "Make/{files,options}" files on those in the original installation:

```
cp -r $FOAM_SRC/functionObjects/field/Make $WM_PROJECT_USER_DIR/src/functionObjects/field
```

Change the "Make/files" file to contain only:

```
normalizedHelicity/normalizedHelicity.C
LIB = $(FOAM_USER_LIBBIN)/libmyFieldFunctionObjects
```

Since we copied a part of a library it is very likely that our code needs to include files from the original library. Include-files from the present library are found by default (the local lnInclude directory), but now we have moved away from that library. We also need to link to the original library. Therefore we have to add the following to "Make/options" (at appropriate locations):

```
-I$(LIB_SRC)/functionObjects/field/lnInclude
-lfieldFunctionObjects
```

We should ideally clean up the options file, to only include the necessary, i.e.:

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/functionObjects/field/lnInclude
LIB_LIBS = \
    -lfieldFunctionObjects
```

Compile to make sure that it works:

```
wmake $WM_PROJECT_USER_DIR/src/functionObjects/field
```

At this point it is good to check that it still works as the original implementation. Search the tutorials for a case that uses the vorticity functionObject, change it to normalizedHelicity, and check that it works and gives the same output as before. You will have to do this without instructions.

Now you should have a look at the normalizedHelicity.{H,C} files and try to understand them. You see that there is mainly one function, named calc(). That function is called by the runTime object at the end of each time step (when runTime++ is executed).

We introduce the expression for normalized helicity by changing the entire calc() function in normalizedHelicity.C to:

```
bool Foam::functionObjects::normalizedHelicity::calc()
{
    if (foundObject<volVectorField>(fieldName_))
    {
        dimensionedScalar minV("minV", dimensionSet(0,1,-2,0,0,0,0),1e-12);
        const volVectorField& field = lookupObject<volVectorField>(fieldName_);
        return store
        (
            resultName_,
            ( fvc::curl(field) & field ) /
              max((mag(fvc::curl(field)) * mag(field)), minV)
        );
    }
    return false;
}
```

Compile:

```
wmake $WM_PROJECT_USER_DIR/src/functionObjects/field
```

Test it for "SRFSimpleFoam/mixer":

```
rm -r $FOAM_RUN/normalizedHelicityMixer
cp -r $FOAM_TUTORIALS/incompressible/SRFSimpleFoam/mixer $FOAM_RUN/normalizedHelicityMixer
sed -i s/1000/200/g $FOAM_RUN/normalizedHelicityMixer/system/controlDict
blockMesh -case $FOAM_RUN/normalizedHelicityMixer
echo 'functions
{
    normalizedHelicity1
    {
        type normalizedHelicity;
        libs ("libmyFieldFunctionObjects.so");
        writeControl timeStep;
        writeInterval 100;
        U Urel;
    }
    vorticity1
    {
        type        vorticity;
        libs        ("libfieldFunctionObjects.so");
        writeControl timeStep;
        writeInterval 100;
        U Urel;
    }
    Q
    {
        type        Q;
        libs        ("libfieldFunctionObjects.so");
        writeControl timeStep;
        writeInterval 100;
        U Urel;
    }
};' >> $FOAM_RUN/normalizedHelicityMixer/system/controlDict
SRFSimpleFoam -case $FOAM_RUN/normalizedHelicityMixer
ls $FOAM_RUN/normalizedHelicityMixer/200
```

Have a look in paraFoam at the difference between the vorticity, the Q-criterion and the normalizedHelicity, calculated based on Urel.

# Optional:

You can also implement functionObjects from scratch, using foamNewFunctionObject. You can play with that if you like.

# License