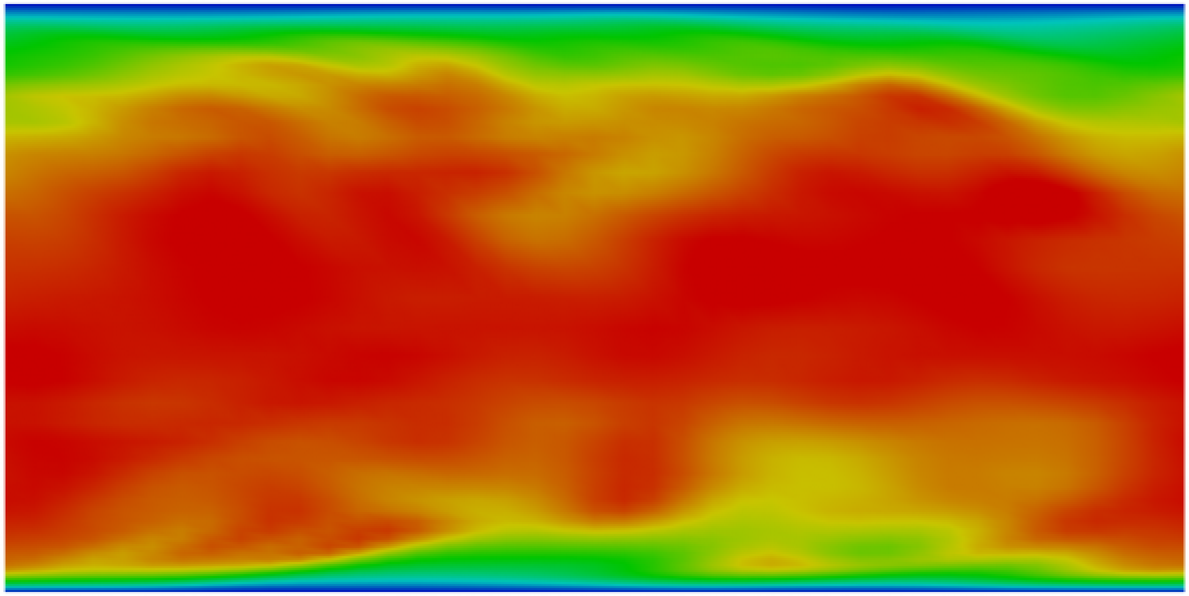
fvOptions meanVelocityForce



An Example Usage of meanVelocityForce

fvOptions file C++

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

momentumSource
{
    type            meanVelocityForce;
    active          yes;

    meanVelocityForceCoeffs
    {
        selectionMode   all;
```

```
        fields          (U);
        Ubar            (0.1335 0 0);
        relaxation      1.0;
    }
}

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

- ***fields***: [**Required**] Name of the velocity field
- ***Ubar***: [**Required**] Desired mean velocity
- ***relaxation***: [**Optional**] Relaxation factor (default = 1.0)
- ***selectionMode***: [**Required**] Domain the source is applied (***all/cellSet/cellZone/points***)

The parameter name ***fields** has been changed from **fieldNames*** in the latest version as described in this page.

The ***meanVelocityForce*** fvOptions calculates a momentum source so that the volume averaged velocity (1) in the whole computational domain (***all***) or a part of domain specified using ***cellSet*** or ***cellZone*** reaches the desired mean velocity ***Ubar***.

$$\frac{\sum_i \left(\frac{\bar{\boldsymbol{u}}}{|\bar{\boldsymbol{u}}|} \cdot \boldsymbol{u}_i \right) V_i}{\sum_i V_i}, \tag{1}$$

where the summation is over the cells that belong to the user-specified domain, $\bar{\boldsymbol{u}}$ is ***Ubar***, $\boldsymbol{u}_i$ is the velocity in the i-th cell and V_i is the volume of the i-th cell.

patchMeanVelocityForce

The ***patchMeanVelocityForce*** fvOptions is also available to specify the desired mean velocity on a patch instead of the volumetric average (1).

```
fvOptions file                                                              C++
momentumSource
{
    type                patchMeanVelocityForce;
    active              yes;

    patchMeanVelocityForceCoeffs
    {
        selectionMode   all;

        fields          (U);
```

```
        Ubar            (0.1335 0 0);
        patch           inout1_half0;
        relaxation      1.0;
    }
}
```

Tutorial

- [incompressible/pimpleFoam/LES/channel395]()

Source Code

In the case of ***pimpleFoam***, we can find three lines related to the ***fvOptions*** as shown in the following code ([*UEqn.H*]()):

applications/solvers/incompressible/pimpleFoam/UEqn.H C++

```cpp
1  // Solve the Momentum equation
2
3  MRF.correctBoundaryVelocity(U);
4
5  tmp<fvVectorMatrix> tUEqn
6  (
7      fvm::ddt(U) + fvm::div(phi, U)
8    + MRF.DDt(U)
9    + turbulence->divDevReff(U)
10  ==
11     fvOptions(U)
12 );
13 fvVectorMatrix& UEqn = tUEqn.ref();
14
15 UEqn.relax();
16
17 fvOptions.constrain(UEqn);
18
19 if (pimple.momentumPredictor())
20 {
21     solve(UEqn == -fvc::grad(p));
22
23     fvOptions.correct(U);
24 }
```

In what follows, we will take a closer look at each of three highlighted lines when the ***meanVelocityForce*** fvOptions is used. The source files of this *class are in [src/fvOptions/sources/derived/meanVelocityForce]().*

- ***l. 11***

*The **addSup** function is called from l.11 in UEqn.H when discretizing the momentum equation and source term Su is added to the equation.*

```
void Foam::fv::meanVelocityForce::addSup
(
    fvMatrix<vector>& eqn,
    const label fieldi
)
{
    DimensionedField<vector, volMesh> Su
    (
        IOobject
        (
            name_ + fieldNames_[fieldi] + "Sup",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        mesh_,
        dimensionedVector("zero", eqn.dimensions()/dimVolume, Zero)
    );

    scalar gradP = gradP0_ + dGradP_;

    UIndirectList<vector>(Su, cells_) = flowDir_*gradP;

    eqn += Su;
}


void Foam::fv::meanVelocityForce::addSup
(
    const volScalarField& rho,
    fvMatrix<vector>& eqn,
    const label fieldi
)
{
    this->addSup(eqn, fieldi);
}
```

- *l. 17*

The ***constrain*** function is called from l.17 in *UEqn.H*. The if loop is true in the first iteration and else loop is executed for the other iterations to initialize and update the *volScalarField* defined as the reciprocal of the diagonal coefficients.

constrain function in meanVelocityForce.C C++

```
void Foam::fv::meanVelocityForce::constrain
(
    fvMatrix<vector>& eqn,
    const label
)
{
    if (rAPtr_.empty())
    {
        rAPtr_.reset
        (
            new volScalarField
            (
                IOobject
                (
                    name_ + ":rA",
                    mesh_.time().timeName(),
                    mesh_,
                    IOobject::NO_READ,
```

```
                    IOobject::NO_WRITE
                ),
                1.0/eqn.A()
            )
        );
    }
    else
    {
        rAPtr_() = 1.0/eqn.A();
    }

    gradP0_ += dGradP_;
    dGradP_ = 0.0;
}
```

- *l. 23*

As the average velocity *magUbarAve* (1) reaches a user-specified value ***Ubar***, the pressure gradient increment *dGradP_* needed to adjust the average velocity converges to 0.

correct function in meanVelocityForce.C C++

```cpp
void Foam::fv::meanVelocityForce::correct(volVectorField& U)
{
    const scalarField& rAU = rAPtr_();

    // Integrate flow variables over cell set
    scalar rAUave = 0.0;
    const scalarField& cv = mesh_.V();
    forAll(cells_, i)
    {
        label celli = cells_[i];
        scalar volCell = cv[celli];
        rAUave += rAU[celli]*volCell;
    }

    // Collect across all processors
    reduce(rAUave, sumOp<scalar>());

    // Volume averages
    rAUave /= V_;

    scalar magUbarAve = this->magUbarAve(U);

    // Calculate the pressure gradient increment needed to adjust the average
    // flow-rate to the desired value
    dGradP_ = relaxation_*(mag(Ubar_) - magUbarAve)/rAUave;

    // Apply correction to velocity field
    forAll(cells_, i)
    {
        label celli = cells_[i];
        U[celli] += flowDir_*rAU[celli]*dGradP_;
    }

    scalar gradP = gradP0_ + dGradP_;

    Info<< "Pressure gradient source: uncorrected Ubar = " << magUbarAve
        << ", pressure gradient = " << gradP << endl;

    writeProps(gradP);
}
```

More from my site

- Temperature calculation from energy variables in OpenFOAM
- National Committee for Fluid Mechanics Films
- Computational Aeroacoustics (CAA) part2
- Textbook on medical application of CFD
- Direct Numerical Simulation (DNS)
- New collateTimes option in EnSight surface writer

Author: fumiya

CFD engineer in Japan View all posts by fumiya

fumiya / June 12, 2016 / OpenFOAM, fvOptions / meanVelocityForce

One thought on “fvOptions meanVelocityForce”

Thanh

January 11, 2019 at 3:36 PM

I saw in the code it only handle pressure incompressible case, so can you guide me to apply it for compressible solver? Thank you!