Cite as: Batra, C.: Implementing different drag models in a new OpenFOAM solver (GeN-Foam). In Proceedings of CFD with OpenSource Software, 2020, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2020

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementing different drag models in a new OpenFOAM solver (GeN-Foam)

Developed for OpenFOAM-v2006
Requires: GeN-Foam, Gmsh

Author:
Chirayu BATRA
EPFL
chirayu.batra@epfl.ch

Peer reviewed by:
Ali Yahyaee
Carlo Fiorina
Saeed Salehi

Licensed under CC-BY-NC-SA, https://creativecommons.org/licenses/

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like to learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 22, 2021

Learning outcomes

The reader will learn:

How to use it:

- How the thermal-hydraulics part (single-phase) of the OpenFOAM based Multiphysics solver GeN-Foam is structured and compiled and used for nuclear reactor analysis.
- The thermal-hydraulics part of GeN-Foam is made available to reproduce the tutorial.

The theory of it:

- GeN-Foam uses porous media approach for thermal-hydraulics analysis of advanced nuclear reactors;
- Explain why this approach is used for the analysis of nuclear reactor core, which has a complex geometry;
- Show how a heterogeneous coarse mesh is generated (using gmsh scripting) and used to perform thermal-hydraulics analysis using the subchannel analysis approach;
- Some relevant theory regarding subchannel approach

How it is implemented:

- Explain how a simple case is structured and implemented in GeN-Foam;
- How different drag models can be selected to perform thermal-hydraulic analysis of a fuel assemble of a sodium-cooled fast reactor for single-phase flow
- Explain how the equations are translated to the GeN-Foam code

How to modify it:

- I will show how a simple case is analyzed using the solver
- Modify/add new drag models (not currently available in the solver) to capture different flow conditions in a fast reactor sub-assembly and demonstrate the effects of different models in capturing the flow behaviour.

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- How to run standard document tutorials using `twoPhaseEulerFoam`.
- Simple scripting to understand the usage of Gmsh tool to generate a mesh.
- Fundamentals of Computational Methods for Fluid Dynamics, Book by J. H. Ferziger and M. Peric
- How to customize a solver and do top-level application programming.

Contents

1 Introduction **5**
- 1.1 Background . . . 5
- 1.2 GeN-Foam . . . 6
- 1.3 Structure of the Report . . . 7

2 Generating Mesh **8**
- 2.1 Gmsh . . . 9

3 Modifying Solver **11**
- 3.1 GeN-Foam structure . . . 11
- 3.2 Adding new drag models . . . 11
  - 3.2.1 Novendstern Model . . . 12
  - 3.2.2 Implementation in GeN-Foam . . . 12
    - 3.2.2.1 Understanding the header file . . . 14

4 Case Set-up **17**
- 4.1 Case set-up information . . . 17
  - 4.1.1 Thermal-hydraulics Setup . . . 17
- 4.2 Basic case set-up . . . 18
  - 4.2.1 Creating geometry . . . 19
  - 4.2.2 phaseProperties dictionary . . . 19
  - 4.2.3 Setting up the initial and boundary conditions . . . 20

5 Results **21**
- 5.1 Thermal-Hydraulics results . . . 21

Nomenclature

Acronyms

CFD	Computational Fluid Dynamics
SFR	Sodium cooled fast reactor

English symbols

D	Hydraulic diameter	m
M	Multiplication factor	
p	Pressure	Pa
V_1	Central channel velocity	m/s

Greek symbols

ρ	Fluid density	kg/m^3

Chapter 1

Introduction

1.1 Background

High-fidelity thermal-hydraulic analysis of the innovative or Generation-IV (Gen-IV) Sodium Fast Reactor (SFR) core in normal and transient operation is compellingly challenging owing to high power density and low volume fraction of coolant in the core, presence of subassemblies (SA) of different designs in the core, including axial heterogeneity of SA design, design of each SA as an isolated channel with possible use of windows in the SA wrappers to improve heat transport between neighbouring SAs, bypass flow between SAs (interassembly flow), use of helical wire spacers in fuel pin bundles, features of thermal and physical properties of sodium, etc.

Three kinds of numerical approaches can be applied to analyze the thermal-hydraulic behaviour of the reactor cores and fuel SAs.

1. The **system analysis codes** use the lumped parameter approach
2. The **Computational Fluid Dynamics (CFD)** approach uses fine scaling
3. The **subchannel based numerical approach** or low resolution CFD (coarse mesh or porous media) calculates the average values

OpenFOAM based multi-physics solver GeN-Foam uses the porous media approach for its thermal-hydraulics part of the solver and can be implemented to model the nuclear reactor subassembly using the subchannel approach. This approach requires creation of a specialized structured mesh to properly represent each subchannel.

Figure 1.1 shows the crossectional view of a hexagonal fast reactor subassembly, along with three different types of subchannels associated with the flow. The corresponding structured mesh that is required to solve the physics is shown in Figure 1.2.

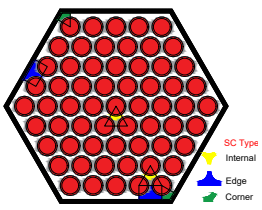


Figure 1.1: Different types of flow channels in a FR Subassembly

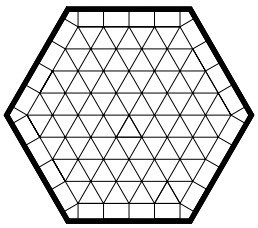


Figure 1.2: Control Volume/Mesh to be used for solving the Physics

1.2 GeN-Foam

The multi-physics solver GeN-Foam [1] contains a thermal-hydraulics sub-solver based on the standard $k-\varepsilon$ turbulence model for compressible or incompressible flows, but extended to coarse-mesh applications through the use of a porous medium approach [2] for user-selected cell zones inside the mesh. The thermal-hydraulic sub-solver is inspired by the available OpenFOAM solvers (`twoPhaseEulerFoam` and `reactingTwoPhaseEulerFoam`). This report focuses on the single-phase part of the thermal-hydraulcis sub-solver of GeN-Foam. Equations for the turbulent single-phase flow of a fluid in a porous medium can be rigorously derived from standard Navier–Stokes equations via time and volume averages. The resulting equations of mass, momentum, and energy conservation are:

$$\frac{\partial \gamma \rho}{\partial t} + \nabla \cdot (\gamma \rho \boldsymbol{u}) = 0 \tag{1.1}$$

$$\frac{\partial \gamma \rho u}{\partial t} + \nabla \cdot (\gamma \rho \boldsymbol{u} \otimes \boldsymbol{u}) = \nabla \cdot (\mu_T \nabla \boldsymbol{u}) - \nabla \gamma p + p_i \nabla \gamma + \gamma \boldsymbol{F}_{\mathrm{g}} + \gamma \boldsymbol{F}_{\mathrm{ss}} \tag{1.2}$$

$$\frac{\partial \gamma \rho e}{\partial t} + \nabla \cdot (\boldsymbol{u} \gamma (\rho e + p)) = \nabla \cdot (\gamma k_T \nabla \mathrm{T}) + \gamma \boldsymbol{F}_{\mathrm{ss}} \cdot \boldsymbol{u} + \gamma \dot{Q}_{\mathrm{ss}} \tag{1.3}$$

Here, the porosity γ is defined as:

$$\gamma = V_\gamma / V$$

The γF_{ss} is in the momentum equation is modeled empirically using the existing pressure drop models, the main focus of this tutorial is to see how new pressure drop models can be added to GeN-Foam solver.

Through the tutorial, we will see how this is modelled in GeN-Foam and then a new pressure drop model will be added from scratch.

If we use the Darcy relation for the single-phase pressure drop, where pressure drop is a function of:

ΔP = func(flow property, flow state, heat condition, channel shape, surface roughness, etc.) we get,

$$\Delta p = f_{\mathrm{D}} \frac{L}{D_{\mathrm{e}}} \frac{\rho V^2}{2} \tag{1.4}$$

where f_{D} is the Darcy friction factor and D_{e} is the hydraulic diameter

The above equation can be written as:

$$\frac{\Delta p}{L} = \frac{\partial p}{\partial z} = F_{\mathrm{ss,z}} = \frac{1}{2} f_D \frac{\rho v^2}{D_e} \tag{1.5}$$

And the surface-surface drag force (F_{ss}) can be semi-implicitly modelled as:

$$F_{\mathrm{ss}} = k(u_{\mathrm{D}}) \cdot u_{\mathrm{D}}$$

where

$$k(u_D) = \frac{1}{2} f_D \frac{\rho u_D}{D_e \gamma^2}$$

In the GeN-Foam solver, looking into the thermal-hydraulics sub-solver under onePhase, the momentum equation implementation could be found as follows:

`UEqn` description in onePhase

```
UEqn =
(
        fvm::div(fluid_.alphaRhoPhi(), U, "div(alphaRhoPhi,U)")
    +   fvm::SuSp(-fluid_.contErr(), U)
    +   MRF_.DDt(alpha*rho, U)
        //- The following is just a re-arrangement of div(nu*grad(U))
    +   (
```

```
            -   fvm::laplacian(alpha*rho*nuEff, U)
            -   fvc::div
                (
                    alpha*rho*nuEff&dev2(T(fvc::grad(U))),
                    "div(alphaRhoPhiNu,U)"
                )
        )
        //- Implicit diagonal contribution of fluid-structure drag,
        //  explicit off-diagonal contribution
    +   fvm::Sp((1.0/3.0)*tr(Kds), U) + (dev(Kds)&U)
    ==
        fvOptions_(alpha, rho, U)
);
```

in the above listing, `Kds` is the $k(u_{\mathrm{D}})$ as described above. It will be shown through the tutorial how this value is calculated using a new drag mode.

More details about the solver can be found in the main paper [1].

1.3 Structure of the Report

Based on brief understanding of what a subchannel is and reading the paper on GeN-Foam (most of the relevant things will be explained in the report), the report is structured in a way to follow the steps to understand and repeat the exercise. Chapter 1 provides a brief introduction and some background information to properly follow the report, Chapter 2 discusses briefly how a structured mesh is generated using Gmsh and explains how the script written for generating mesh through Gmsh can be used for different geometries (3 different geometries are already included). Chapter 3 discusses how to modify the thermal-hydraulics sub-solver to add different drag models. Chapter 4 provides information on how a simple case can be set-up in GeN-Foam. Chapter 5 discusses the results obtained.

Chapter 2

Generating Mesh

To get the desired structured mesh, based on the control volumes defined for the whole subassembly (Figure 2.1), with three types of configurations as shown in Figure 2.2, a separate script written in Gmsh [3] is used.

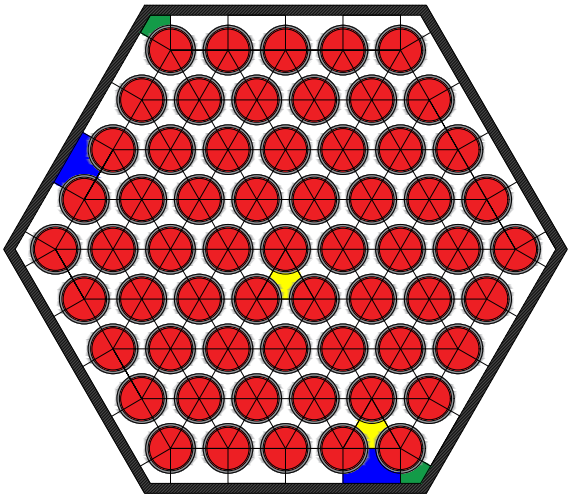
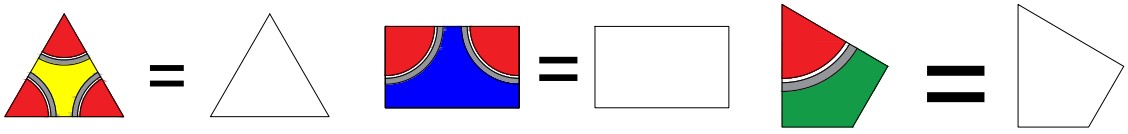


Figure 2.1: Control Volumes defined for the whole Subassembly



(a) (b) (c)

Figure 2.2: Equivalent mesh/control volume for (a) internal (b) edge and (c) corner subchannel

2.1 Gmsh

Gmsh is an open source 3D finite element mesh generator with a built-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities. Gmsh's own scripting language (.geo files) is used to create this mesh.

The below listing can be used to define the geometry. It can be seen from the below listing that there are few geometric parameters that need to be defined, depending upon the type of the reactor assembly. They are also important to define the drag model, that will be explained further in the coming chapters. Based on the reactor subassembly geometry, once these inputs are made, two other macro files are used to create the required mesh. As it is a structured mesh with three different type of cells. The macros could be downloaded from here, and then following command can be run to generate the required mesh. `$ gmsh subChannelMesh.geo`, where the `*.geo` is the name of the geometry file, using Gmsh scripting language.

However, for running the tutorial a .msh file is provided in the case, and it is recommended to use that file. In case some other geometry needs to be built, the macros can then be used.

Gmsh script to define geometry

```
// This is input data to define the geometry

// ---------Geometry 1 CEFR 61 Pin subassembly--------


lc = 1;
edge = 0.32678 ; // length of the side of the hexcan (internal)
FtoF=edge*Sqrt(3); // face to face distance calcualted from edge
thickness=0.0; // thickness of the hexcan
gap = 0.0; // gap between the subassemblies
pitch= 0.0695; // pin pitch
pitchFace=pitch*Sqrt(3); // variable to calculate the face to face distance of inner hexes
rows =5; // no. or rows in the subassembly; center pin is row=1

// example geometry - uncomment to use this geometry; and comment above
// -----------Geometry 2 ORNL 19 Pin subassembly-------------

/*
lc = 1;
edge = 1.9687 ; // length of the side of the hexcan (internal)
FtoF=edge*Sqrt(3); // face to face distance calcualted from edge
thickness=0.0; // thickness of the hexcan
gap = 0.0; // gap between the subassemblies
pitch= 0.726; // pin pitch
pitchFace=pitch*Sqrt(3); // variable to calculate the face to face distance of inner hexes
rows =3; // no. or rowns in the subassembly
*/
```

The output is .msh file. The following command can then be used to convert the mesh to OpenFOAM mesh:

```
gmshToFoam subchannelGeometry.msh // make sure to use the correct file name
```

The .msh file is present in the case folder. Step-by-step guide is provided in the case setup chapter.

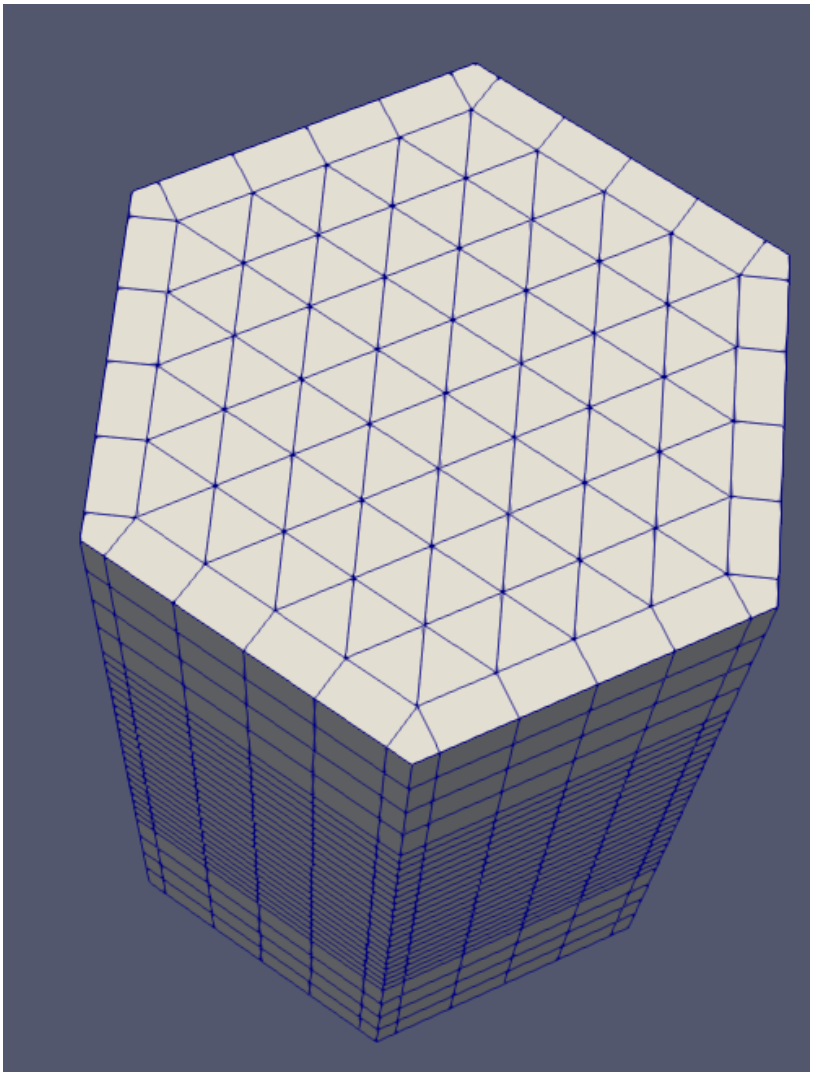


Figure 2.3: Final structured mesh prepared using Gmsh (three types of subchannels can be seen here)

Chapter 3

Modifying Solver

3.1 GeN-Foam structure

As explained in Chapter 1, we will be using a custom solver, GeN-Foam, and modify the drag models within this solver. This approach can be used to also modify the dragmodels in `twoPhaseEulerFoam`. GeN-Foam is a multi-physics solver, where each physics is structured in different classes: "netronics", "thermalHydraulics" and "thermoMechanics" classes to solve the relevant physics and a "multiphysicsControl" class to couple different physics. The `system/controlDict` file in the case folder can be used to activate the required physics, by setting the relevant flags to `true` or `false`. We will focus only on `thermalHydraulics` solver, and in order to add more drag models, we can navigate to the FSModels subdirectory and understand the folder structure.

```
classes
├── multiphysicsControl
├── neutronics
├── thermalHydraulics
│   ├── models
│   │   ├── phaseModels
│   │   ├── physicsModels
│   │   │   └── dragModels
│   │   │       ├── FSModels
│   │   │       ├── FFModels
│   │   │       └── ...
│   │   └── regimeModels
│   ├── solvers
│   └── src
└── thermoMechanics
```

Here you can see some of the drag models already present in the folder. The goal is to add a new drag model based on the experiments done for the wire wrapped fuel assembly of sodium cooled fast reactors.

The next section provides brief theory of the new drag model and then how that can be implemented in GeN-Foam.

3.2 Adding new drag models

This section shows how to add a new drag model to the GeN-Foam solver. The drag models are defined within the thermalhydraulics sub-solver.

To create a new drag model, we can either start from the existing drag model or start from scratch. In order to make the tutorial more complete, we will do this from scratch. We will be implementing the Novendstern model.

3.2.1 Novendstern Model

It is a model developed to predict pressure losses in a wire-wrapped fuel assembly, which determines the flow distribution between the fuel pins theoretically, and multiplies the pressure drop for a smooth pipe, using equivalent diameter techniques, by an empirical correction factor. The empirical correction depends on fuel pin bundle dimensions and flow rate [4]. Using the flow conditions for the central subchannel, the pressure drop is determined by:

$$\Delta p = M f_{\text{smooth}} \frac{L}{D_1} \frac{\rho V_1^2}{2} \tag{3.1}$$

where V_1 and D_1 are the velocity of central channel and its hydraulic diameter respectively.
using Darcy's equation for pressure drop, we can define effective friction factor f_{eff} as:

$$f_{\text{eff}} = M f_{\text{smooth}}$$

The multiplication factor, M, is used to increase the pressure drop above that calculated with the smooth tube friction factor. It primarily accounts for wire lead (H/D) and fuel pin pitch-to-diameter ratio (P/D)

$$M = \left[\frac{1.034}{(P/D)^{0.124}} + \frac{29.7(P/D)^{6.94}\text{Re}^{0.086}}{(H/D)^{2.239}} \right]^{0.885} \tag{3.2}$$

For the calculation of f_{smoooth}, Blasius correlation could be used.

$$f_{\text{smooth}} = 0.316/\text{Re}^{0.25} \tag{3.3}$$

3.2.2 Implementation in GeN-Foam

For implementation in GeN-Foam, we need to get the value of f_{D}, which is equal to f_{eff} here.
It will be represented in this form in GeN-Foam

$$f_{\text{D}} = f_{\text{eff}} = M f_{\text{smooth}} = \left[A_{\text{-}} + B_{\text{-}} \cdot \text{Re}^{0.086}\right]^{0.885} \times \frac{0.316}{\text{Re}^{0.25}} \tag{3.4}$$

$$\therefore f_{\text{D}} = f_{\text{D}}(\text{Re})$$

The drag models are added under: `$WM_PROJECT_USER_DIR/applications/solvers /GeN-Foam/classes/thermalHydraulics/models/physicsModels/dragModels` (assuming that the GeN-Foam solver is placed under the solver directory of user)
Within `dragModels`, there are two folders `FFModels` (fluid-fluid models, for two phase flow) and `FSModels` (fluid-structure models for one phase flow). We will add a new model under the `FSModels`.

In order to better understand the flow through the wire-wrapped subassemblies, the Novendstern model is added to the current solver. To do that, first, create a new folder with the same name "Novendstern". Then we create new source files using the OpenFOAM utility function `foamNewSource`

First, let's create the header file, `Novendstern.H`

```
$ foamNewSource H NovendStern
```

This will create the file `Novendstern.H`.

`Novendstern Class Declaration (Novendstern.H)`

```
1 /*---------------------------------------------------------------------------*\
2 Class
3     Foam::dragModels::Novendstern
```

```
4
5  Description
6      New drag model based on Novendstern correlation
7  SourceFiles
8      Novendstern.C
9  \*---------------------------------------------------------------------------*/
10
11 #ifndef Novendstern_H
12 #define Novendstern_H
13
14 #include "dragModel.H"
15
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 namespace Foam
19 {
20
21 namespace dragModels
22 {
23 /*---------------------------------------------------------------------------*\
24                          Class Novendstern Declaration
25 \*---------------------------------------------------------------------------*/
26
27 class Novendstern
28 :
29     public dragModel
30 {
31     // Private Data
32
33         //- Description of data_
34         scalar A_;
35         scalar B_;
36
37 public:
38
39     // Constructors
40
41         //- Construct from a dictionary and FSPair
42         Novendstern
43         (
44             const objectRegistry& objReg,
45             const dictionary& dict,
46             const FSpair& FSpair
47         );
48
49     //- Destructor
50     virtual ~Novendstern() {};
51
52     // Member Functions
53
54         // Drag Coefficient
55         virtual void correctKd(volTensorField& kd) const;
56
57         scalar fd(const scalar& Re) const;
58
59 };
60
61
62 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
63
64 } // End namespace dragModels
65 } // End namespace Foam
66 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
67
68
69 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
70
71 #endif
```

```
72
73 // ************************************************************************* //
```

3.2.2.1 Understanding the header file

All the drag models are within the `namespace dragModels`

defining the namespace

```
18 namespace Foam
19 {
20
21 namespace dragModels
22 {
```

The class inherits from dragModel class and two scalars `A_` and `B_` are declared as private.

declaring the class

```
27 class Novendstern
28 :
29     public dragModel
30 {
31     // Private Data
32
33         //- Description of data_
34         scalar A_;
35         scalar B_;
```

Then, the constructor is defined, which constructs from a dictionary, and fluid-structure pair (another class in the GeN-Foam solver to model fluid structure interaction).

Constructor

```
41         //- Construct from a dictionary and FSPair
42         Novendstern
43         (
44             const objectRegistry& objReg,
45             const dictionary& dict,
46             const FSpair& FSpair
47         );
```

And finally, two member functions are defined, one for f_D and another for K_D, where f_D is the friction factor or drag factor or friction coefficient and K_D is the implicit part of the drag equation.

Member Functions

```
52     // Member Functions
53
54         // Drag Coefficient
55         virtual void correctKd(volTensorField& kd) const;
56
57         scalar fd(const scalar& Re) const;
```

After declaring the header file, we can create the class definition by

```
$ foamNewSource C NovendStern
```

This will create the file `Novendstern.C`.

And the class is implemented as follows:

In order to make it runTime selectable, it needs to be added to the runTime selection table by using the function `addTorunTimeSelectionTable`

`addTorunTimeSelectionTable` is a macro, and defined in `addTorunTimeSelectionTable.H`

addToRunTimeSelection.H macro

```
//- Add to hash-table of functions with typename as the key
#define addToRunTimeSelectionTable\
(baseType,thisType,argNames)                                                \
                                                                            \
    /* Add the thisType constructor function to the table */                \
    baseType::add##argNames##ConstructorToTable<thisType>                   \
        add##thisType##argNames##ConstructorTo##baseType##Table_
```

By using this macro, we can add Novendstern(`thisType`) to the selection table of dragModel(`baseType`), where the name of parameter list for fluid-structure is defined as FSDragModels (`argNames`)

runTimeSelectionTable

```
namespace Foam
{

namespace dragModels
{
    defineTypeNameAndDebug(Novendstern, 0);
    addToRunTimeSelectionTable(dragModel, Novendstern, FSDragModels);
}
}
```

`defineTypeNameAndDebug` is another macro defined in `className.H`. To use this function, we should also add `#include "addToRunTimeSelectionTable.H"` to the source file.

To understand this more properly, we can take a peek into dragModel.H and see how the selection table is define for fluid-structure pair.

runtime construction in dragModel.H

```
//- Declare runtime construction
        declareRunTimeSelectionTable
        (
            autoPtr,
            dragModel,
            FSDragModels,
            (
                const objectRegistry& objReg,
                const dictionary& dict,
                const FSPair& FSPair
            ),
            (objReg, dict, FSPair)
        );
```

Here, we need to add `#include "runTimeSelectionTables.H"` to the header file.

Moving further in the source file, we will now define the constructor and initialize values

constructor definition in Novendstern.C

```
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * //
Foam::Novendstern::Novendstern
(
    const objectRegistry& objReg,
    const dictionary& dict,
    const FSPair& FSPair
)
:
    dragModel
    (
        objReg,
        dict,
        FSPair
    )
{
    scalar Dp(this->get<scalar>("pinDiameter"));
    scalar Dw(this->get<scalar>("wireDiameter"));
```

```
34      scalar Lw(this->get<scalar>("wireLeadLen"));
35
36      scalar Pt(Dp+1.0444*Dw); // Pitch (pinDia+1.0444*wireDia)
37      scalar PbyD(Pt/Dp); // P/D ratio
38
39      A_ = 1.034/pow(PbyD,0.124);
40      B_ = 29.7*(pow(PbyD,6.94)/pow((Lw/Dp),2.239));
41  }
```

Here, the constructor is initialized with the base class (`dragModel.H`) parameters and then some other parameters which are required in the Novendstern equation are initialized. There are three mandatory entries in the dictionary: `pinDiameter`, `wireDiameter`, `wireLeadLen`. These should be present in the `constant/fluidRegion/phaseProperties` dictionary of the case files. These are the diameter of the fuel pin, the diameter of the wrapper wire, and the lead length of the wire, respectively. These values are then stored in local variables `Dp, Dw, Lw`.

Then the coefficient `A_` and `B_` are calculated, which will be used in the member function `scalar fd(const scalar& Re) const` defined in the next line:

calculating fd and Kd

```
42  // * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //
43
44  Foam::scalar Foam::dragModels::Novendstern::fd(const scalar& Re) const
45  {
46          return (pow((A_+B_*pow(Re, 0.086)),0.885)*0.316/pow(Re, 0.25));
47  }
48
49  void Foam::dragModels::Novendstern::correctKd(volTensorField& kd) const
50  {
51      #include "calcKdFromFd.H"
52  }
53
54  // ************************************************************************* //
```

After creating the header and source file for the Novendstern model, we need to make changes in the `Make/files` and `Make/options` of the library to include this in the compilation. The `model` library of GeN-Foam has its own `Make` folder, which eventually compiles to a library named `libFFSModels2`. Though we can also compile this separately, however, it is better to modify the combined `models` library. Therefore, in the `Make/files` add the following line:

```
physicsModels/dragModels/FSModels/Novendstern/Novendstern.C
```

and, in the `Make/options`, add:

```
-I./physicsModels/dragModels/FSModels/Novendstern \
```

along with the other dragModels that are included in the file.

After that, we can either run `wmake` or use the `Allwmake` script at the root of GeN-Foam solver. It is recommended to us the `Allwmake` script to compile. In order to do that, get to the root folder of the main GeN-Foam solver and execute the command

```
$ ./Allwmake
```

Also, note that for GeN-Foam, these libraries are included in the main solver, and therefore we need not add them in the `system/controlDict` of the case. In case you compile the library separately, then there will be a need to add this in the `system/controlDict` file.

```
EXE_LIBS = \
    -L$(FOAM_USER_LIBBIN) -lFFSModels2 \
    -L$(FOAM_USER_LIBBIN) -lFFSSolvers2 \
    -L$(FOAM_USER_LIBBIN) -lFFSSrc2 \
```

After successful compilation, we can go and prepare the case for the run.

Chapter 4

Case Set-up

4.1 Case set-up information

The multi-region approach

GeN-Foam employs a traditional OpenFOAM multi-region approach to model different physics using different meshes. This implies that the `0`, `constant` and `system` folders of each case contains multiple folders, one for each physics. In particular, the regions `fluidRegion`, `neutroRegion` and `thermoMechanicalRegion` are employed in GeN-Foam for thermal-hydraulics, neutronics, and thermal-mechanics, respectively. Please notice that a dummy mesh must always be present for each physics, even if not solved for. One may use the dummy meshes and initial/boundary conditions provided in the EMPTY tutorial case.

The multi-zone approach

In order to assign different properties (for instance, different porous medium properties or different cross sections) to different zones in a mesh, GeN-Foam employs the OpenFOAM concept of `cellZone`. Each mesh should then be divided in different cellZones. Each `cellZone` is associated to a name which is used in `*constant/fluidRegion/phaseProperties`, `*constant/neutroRegion/nuclearData`, `*constant/themoMechanicalRegion/themoMechanicalProperties` to associate each cellZone with a set of properties. The creation of cellZones is normally allowed by all meshers, though different names are normally used (for instance, physical entities in Gmsh and groups in Salome). In some cases, conversion of the mesh into an OpenFOAM format creates cellSet instead of cellZones. In these cases, one can use the topoSet utility to convert cellSets into cellZones.

4.1.1 Thermal-hydraulics Setup

Both single- and two-phase simulations can be performed using GeN-Foam. The selection is made by changing the keyword `thermalHydraulicType` in `constant/fluidRegion/phaseProperties`. Both solvers were developed for coarse-mesh porous-medium applications. This means that, for instance, a reactor core is modeled without resolving each pin. As a matter of fact, for consistency, the finest radial mesh chosen by a user should not finer than one cell per pin cell. *Pressure drops are modeled based on user-selectable correlations.* Correlations are also used to evaluate the heat exchange with the fuel. The fuel can be modeled using a 1-D sub-scale model that will calculate temperatures in the cladding and pellet for the pins that are contained in a cell. In addition to fuel, one can model simpler structures. In this case, one single temperature (and not a radial profile) is evaluated for each cell. This can be used to model structures like the assembly wrappers, the reflectors, or for a simplified modelling of a heat exchanger. The dictionary `constant/fluidRegion/phaseProperties` must include all information about pressure drop and heat transfer correlations, as well indication about the sub-scale models to be used for each mesh zone. As regards to the boundary conditions,

OpenFOAM provides most of the boundary conditions one may need for thermal-hydraulics models. In addition, a few boundary conditions have been included in GeN-Foam and can be found in `GeN-Foam/classes/thermalHydraulics/src/boundaryConditions`. In particular, information on the use of each boundary condition can be found in the header files (`.H`) [5].

The set-up for **neutronics** and **thermal-mechanics** is not explained in this document as it is not relevant to be discussed here.

GeN-Foam can selectively couple each physics. The selection of the physics to solve is done in system/controlDict while the parameters for the coupling are set in system/fvSolution.

4.2 Basic case set-up

The basic case files are provided, some of the important entries of the dictionaries will be provided below in order to understand the case setup completely. Basic case structure is like following

```
subChannelCase
├── 0
│   ├── fluidRegion
│   │   ├── p
│   │   ├── U
│   │   └── ..
│   ├── neutroRegion
│   │   └── ..
│   └── thermoMechanicalRegion
│       └── ..
├── constant
│   ├── fluidRegion
│   │   ├── g
│   │   ├── phaseProperties
│   │   ├── polyMesh
│   │   └── turbulenceProperties
│   ├── neutroRegion
│   │   ├── neutronicProperties
│   │   ├── polyMesh
│   │   └── ..
│   └── thermoMechanicalRegion
│       ├── thermoMechanicalProperties
│       ├── polyMesh
│       └── ..
└── system
    ├── controlDict
    ├── fluidRegion
    │   ├── fvSchemes
    │   └── fvSolution
    ├── neutroRegion
    │   ├── fvSchemes
    │   └── fvSolution
    ├── thermoMechanicalRegion
    │   ├── fvSchemes
    │   └── fvSolution
    ├── fvSchemes
    └── fvSolution
```

As it can be seen from the structure, each folder (viz. `0`, `constant`, `system`) has its own sub-folders for each physics to be solved. It is required to have all these folder, even with dummy files and values. We will focus mainly on the `fluidRegion`, as it is the relevant physics for our case.

4.2.1 Creating geometry

The geometry file *subChannelScript.msh* is provided within the case folder, execute the following command to generate the required mesh

```
$ gmshToFoam subChannelScript.msh
```

This will generate the `polyMesh` folder in the `constant` folder. This folder should be copied to `constant/fluidRegion` folder. As GeN-Foam is a multi-physics solver, so different meshes can be generated for different physics and then they need to be copied in the respective folder. The other two folders in the `constant` folder - `neutroRegion` and `thermoMechanicalRegion` should also have a dummy mesh

```
constant
├── fluidRegion
│   └── polyMesh ..... copied folder from generated mesh
├── neutroRegion
│   └── polyMesh ..... keep the dummy mesh
└── thermoMechanicalRegion
    └── polyMesh ..... keep the dummy mesh
```

4.2.2 phaseProperties dictionary

The important and mandatory entries of the `phaseProperties` dictionary are as follows:
We will be using `"onePhase"` as `thermalHydraulicType`, as we will be doing simulations for single phase flow. GeN-Foam as two phase flow capabilities and the keyword should be change to `"twoPhase"` in case the simulation is for two phase flow.

phaseProperties

```
// --- THERMAL-HYDRAULIC TYPE ------------------------------------------- //
// ---------------------------------------------------------------------- //

thermalHydraulicType    "onePhase";
```

As GeN-Foam uses porous medium approach, the porosity is defined by defining the `volumeFraction`, and the hydraulic diameter ($D_h = 4A/P$) is also specified here. The mandatory entry is `type`, and the only keyword that is available is `byZone`. The idea here is to have different regions with different porosity, this is done by making use of `cellZones` in OpenFOAM (inspired by `chtMultiRegionFoam`). In this particular case, only one 'zone' is used and named subChannel, with same prosity all across. Ideally, there are three different regions and therefore three different cellZones should be defined with their actual porosity and respective hydraulic diameter. This is done just to keep the tutorial simple and demonstrate mainly the difference due to different drag models. When generating the mesh with Gmsh, it allows to defined different 'Physical Volumes', which convert to cellZones, when converting the file from Gmsh to OpenFOAM readable file, by using the utility `gmshToFoam`.

```
// --- STRUCTURES PROPERTIES -------------------------------------------- //
// ---------------------------------------------------------------------- //

structureProperties
{
    type                "byZone";

    "subChannel"
    {
        volumeFraction    0.386;
        Dh                0.01;
    }
}
```

The `regimeMapModel` is mainly used for two phase flow simulation, the mandatory entry is `type` and here we use the value `none`, as we are not using two phase flow. If we use the value `oneParameter`, the there are several other mandatory keywords and subdictionaries that should be added to `regimeMapModel` subdictionary.

```
// --- REGIME MAP MODEL ------------------------------------------------- //
// ---------------------------------------------------------------------- //

regimeMapModel
{
    type                none;
}
```

To, keep the structure of the code and dictionary similar, the physicsModelByRegime subdictionary is also used even if we are not using any regimeMapModel. The run time selectable drag Models are defined here (aslo by each zone, in this case we have only one zone), in case there are more zones, similar type of subdictionary can be used for other zones and diffferent drag models can be specified for different zones. For the tutorial, we will use the newly constructed Novendstern model, as explained in Chapter 3, the mandatory entries for this drag model are `pinDiameter`, `wireDiameter`, and `wireLeadLen`, all these values are dependent on the geometry and correspond to the ones used in Chapter 2. In case we want to use other drag model, the 2nd dragModel should be uncommented and first one should be commented. There are no heat transfer models being used and therefore this subdictionary is left empty, however, it is mandatory to have this subdictionary.

```
// --- REGIME PHYSICS FOR EACH REGIME ----------------------------------- //
// ---------------------------------------------------------------------- //

physicsModelsByRegime
{
    dragModels
    {
        "subChannel"
            {
                type              Novendstern;
                pinDiameter         0.005842;
                wireDiameter        0.001445;
                wireLeadLen         0.302;
                principalAxis       localZ;
                transverseDragModel GunterShaw;

                // 2nd dragModel
                //type            DarcyReynoldsPower;
                //coeff           2;
                //exp             -0.125;

            }
    }

    heatTransferModels
    {   }
```

4.2.3 Setting up the initial and boundary conditions

In order to understand the behaviour of different drag models, a simple pressure driven flow can be established, where in the pressure at the boundaries is defined and velocity is allowed to develop based on the pressure difference. The files to be modified for pressure and velocity initial and boundary conditions can be found under `0/fluidRegion`.

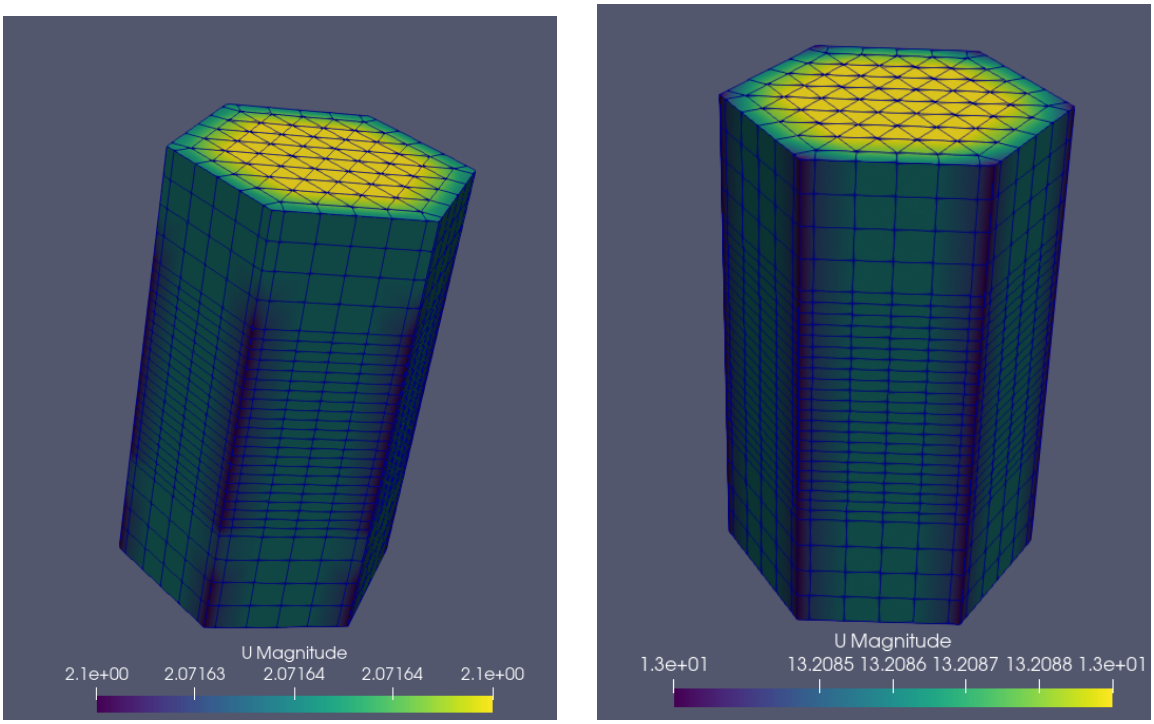
Chapter 5

Results

5.1 Thermal-Hydraulics results

As a simple understanding of a system with fixed pressure on the boundaries, we can expect to see different velocity development depending upon the difference in fluid structure drag. Figure 5.1 shows the difference in final velocity value across the subassembly for `darcyReynoldsPower` (Figure 5.1a) and `Novendstern` (Figure 5.1b) drag models when applied to a same geometry with same porosity. This is due to the difference in friction factors for two drag models, which leads to different pressure drops, and therefore difference in velocity values.

Simple results can be visualized by looking at the velocity



(a) DarcyReynoldsPower drag model

(b) Novendstern drag model

Figure 5.1: Final velocity using two different drag models

Note: no validation is done by comparing to actual results, the main purpose is to show that the new model is added and behaving in an expected way. Actual verification can be done by using

proper geometry as well as boundary conditions.

Bibliography

[1] C. Fiorina, I. Clifford, M. Aufiero, and K. Mikityuk, "GeN-Foam : a novel OpenFOAM ® based multi-physics solver for 2D / 3D transient analysis of nuclear reactors," *Nuclear Engineering and Design*, vol. 294, pp. 24–37, 2015.

[2] K. Vafai, *Handbook of porous media*. Crc Press, 2015.

[3] C. Geuzaine and J. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, pp. 1309–1331, 2009.

[4] E. H. Novendstern, "Turbulent flow pressure drop model for fuel rod assemblies utilizing a helical wire-wrap spacer system," *Nuclear Engineering and Design*, vol. 22, no. 1, pp. 28–42, 1972.

[5] "GeN-Foam wiki written on gitlab." https://gitlab.com/foam-for-nuclear/GeN-Foam/-/wikis/Home. Accessed: 2020-11-04.

Study questions

1. What is the use of adding drag models to the solver?
2. What is the advantage of using GeN-Foam solver, and which physics is already coupled in it?
3. Why is gmsh used in this case for generating mesh, what is the benefit of using such meshing tool?

Index

GeN-Foam, 5–7, 11, 12, 14, 16–19

Gmsh, 2, 7–9, 17, 19

Novendstern, 11, 12, 15, 16, 20

OpenFOAM, 5, 6, 9, 12, 17–19