Cite as: Göral, K. D.: Implementing shear current theory into the `waves2Foam` toolbox. In Proceedings of CFD with OpenSource Software, 2021, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2021

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementing shear current theory into the `waves2Foam` toolbox

Developed for OpenFOAM-v2012
Requires: `waves2Foam` toolbox

Author:
Koray Deniz Göral
Technical University of Denmark
kdego@mek.dtu.dk

Peer reviewed by:
David R. Fuhrman
Saeed Salehi
Mahmoud Gadalla

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 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 20, 2022

Learning Outcomes

The main requirements of a tutorial in the course are that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- Examining the `waves2Foam` toolbox considering the available wave theories library.
- Showing how to create a new wave theory besides the existing ones in the `waves2Foam` toolbox to lead the implementation of the shear current theory.

The theory of it:

- Describing the structure of available wave theories library in `waves2Foam` toolbox.
- Describing the simple (uniform) current solution in `waves2Foam` toolbox and pointing to the need for a more realistic approach considering the shear currents.
- Explaining the shear current mixing length theory from the van Driest solution for the flow above smooth beds, including the extension of Cebeci & Chang for incorporating effects of bed roughness.

How it is implemented:

- Describing the `waveTheory` class and sub-classes in `waves2Foam` toolbox.
- Describing the `waveProperties.input` file structure in `waves2Foam` toolbox.
- Describing the `makeNewWaveTheory` script in `waves2Foam` toolbox.

How to modify it:

- Showing the steps to implement the shear current theory considering the flow on both smooth and rough walls with all the required equations and other corresponding files into the wave theories library as a new option in `waves2Foam` toolbox.
- Setting up a tutorial case to show the shear current model results considering the wall roughness.

Prerequisites

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

- Previous experience with the OpenFOAM
- Previous experience with the `waves2Foam` toolbox [1, 2]
- The simple (uniform) current and shear current mixing length theory [3]

Contents

List of Figures

Listings

Nomenclature

English symbols

$\overline{u}$	Mean velocity	m/s
A_d	Damping coefficient	-
D	Water depth	m
d	Mean grain diameter of the sandy bed	m
Fr	Froude number	-
g	Gravitational acceleration	m/s^2
k_s	Nikuradse's equivalent sand roughness	m
Re	Reynolds number	-
U	Flow velocity	m/s
U_f	Friction velocity	m/s

Greek symbols

κ	von Karman constant	-
ν	Fluid kinematic viscosity	m^2/s
ρ	Fluid density	kg/m^3

Chapter 1

Introduction

Coastal regions have high importance for human activities such as producing energy from offshore wind farms, trading with harbors, and performing touristic activities on the beach areas. Therefore, the nature of coastal regions is highly prone to change with these human-related activities. These activities require the construction of different purposed coastal structures like breakwaters, harbors, or wind turbines. The design and operation of these coastal structures are a complex study-field that one has to consider the interactions such as wave-wave, wave-structure, wave-bathymetry.

The `waves2Foam` toolbox is one of the powerful numerical toolboxes used for the coastal field problems. This toolbox has been created for estimating the wave characteristics considering different wave theories. The `waves2Foam` toolbox is coupled with the OpenFOAM and gives open access to users. Therefore, the `waves2Foam` toolbox is updated as for the OpenFOAM, considering the latest developments in the coastal research area.

The `waves2Foam` toolbox is very powerful to find the wave parameters propagating on the free surface, but especially the potential current theory in the toolbox is not sufficient to find the different flow characteristics on the entire flow depth (uniform velocity profile is given for entire water depth for the potential current theory) like the flow characteristics close to the wall considering the given wall roughness like the sandy bed. Yet, some of the coastal region applications like the scour of the sandy bed under the pipeline are bound to the flow characteristics close to the wall (sandy bed). Therefore, as a trial to implement new functionality in the `waves2Foam` toolbox, this project focused on the shear current theory which is implemented in the `waves2Foam` toolbox for resolving the flow characteristics for the entire section from wall to the free surface.

This project is structured as follows:

In Chapter 2, "Theoretical Background", the structure and available wave theories in the `waves2Foam` toolbox, simple (uniform) current theory, and shear current mixing length theory from the van Driest solution with the extension of Cebeci & Chang for incorporating effects of bed roughness are explained [3].

In Chapter 3, "Library and Class Structure of `waves2Foam` Toolbox", the `waveTheory` class and sub-classes, the `waveProperties.input` file structure, and the `makeNewWaveTheory` script in the `waves2Foam` toolbox are described.

In Chapter 4, "Modification", the steps to implement the shear current theory considering the flow on both smooth and rough walls inside the `waves2Foam` toolbox is explained.

In Chapter 5, "Tutorial Case", the selected `waveFlume` tutorial case inside the `waves2Foam` toolbox is manipulated for the new `shearCurrent` wave theory library tutorial, and the results with the given bed roughness are shown.

Chapter 2

Theoretical Background

The main contents of `waves2Foam` toolbox are explained in this section, briefly. Then, the potential current solution in the `waves2Foam` is discussed in detail and pointed to the need of implementing the shear current theory as a new `waveTheory` library into the `waves2Foam` toolbox. Also, the shear current theory is explained in detail.

2.1 `waves2Foam` toolbox

The `waves2Foam` toolbox is an open-source plug-in toolbox for the OpenFOAM and was originally developed by Niels Gjøl Jacobsen at the Technical University of Denmark. The libraries in the `waves2Foam` toolbox are used for generating and absorbing free surface water waves applied with the relaxation zone technique (active sponge layers) where the relaxation zones can take arbitrary shapes. The up-to-date instructions on how to download and install the `waves2Foam` toolbox can be found at the OpenFOAMWiki page of `waves2Foam` toolbox [1, 4] .

The `waves2Foam` toolbox is compatible with the three main branches:

- OpenFOAM (Foundation) as distributed through www.openfoam.org.
- OpenFOAM (ESI) as distributed through www.openfoam.com.
- foam-extend (FE) as distributed through the foam-extend community.

In this report, OpenFOAM-v2012 from the ESI distribution is used and the instructions on how to download and install the `waves2Foam` toolbox inside of the selected OpenFOAM-v2012 is given below.

One has to check that the additional third-party packages are installed on the system, before compiling the `waves2Foam` toolbox. The additional third-party packages are:

- GNU Scientific Library (GSL)
- Subversion (SVN)
- git
- gfortran

The `waves2Foam` toolbox is available for download through the OpenFOAM-Extend SourceForge SVN. After obtaining the source code via SVN, one has to move the `waves2Foam` folder inside to the `application/utilities` sub-folder of the `WM_PROJECT_USER_DIR` folder of OpenFOAM-v2012. Then, the `Allwmake` script in the folder `waves2Foam` can be executed for finishing the installation. An example code for the installation of the `waves2Foam` toolbox is given below in Listing 2.1.

Listing 2.1: An example code for the installation

```
// * Source the selected OpenFOAM version * //
svn co http://svn.code.sf.net/p/openfoam-extend/svn/trunk/Breeder_1.6/other/waves2Foam
mkdir -p $WM_PROJECT_USER_DIR/applications/utilities
cp -r waves2Foam $WM_PROJECT_USER_DIR/applications/utilities
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam
./Allwmake
```

One can check the tutorials of `waves2Foam` toolbox to make sure that the compilation procedure of `waves2Foam` toolbox inside to the OpenFOAM-v2012 is successful. If modifications and additions have been made to the `waves2Foam` toolbox like adding a new wave theory, one has to recompile the `waves2Foam` toolbox as a whole with the given changes which are explained in Chapter 4, in detail. The available solvers and the wave theories in the `waves2Foam` toolbox are listed below. More information about the `waves2Foam` toolbox are available in the manual [2].

Solvers:

1. waveFoam
2. porousWaveFoam
3. waveDyMFoam (moving meshes)

Wave Theories:

1. Regular wave theories
   (a) Stokes first order
   (b) Standing Stokes first order
   (c) Stokes second order
   (d) Modulated second-order Stokes wave
   (e) Stokes fifth order
   (f) First-order cnoidal theory
   (g) Stream function wave
2. Bichromatic wave theories
   (a) First-order bichromatic wave
   (b) Second-order bichromatic wave
3. First-order irregular waves
   (a) Spectral shape
      i. JONSWAP
      ii. Pierson-Moskowitz
   (b) Spectral discretization
      i. equidistantFrequencyAxis
      ii. cosineStretchedFrequencyAxis
   (c) Phases
      i. Random phase
      ii. Phase focusing for a certain location, x_0 at a certain time instance t_0

4. Solitary wave theories
   (a) First-order solitary wave
   (b) Chappelear (1962)
5. External wave theories
   (a) Empty external method
   (b) Fast summation of irregular waves
   (c) OceanWave3D
6. Potential current

All of the above-mentioned wave theories can be called by `waveProperties.input` file. Also, there is an additional wave theory class named `combinedWaves`. The wave theories mentioned in the above list can be combined with the help of this `combinedWaves` wave theory class. Also, one can implement the new wave theory with the help of a small script called `makeNewWaveTheory`. `waveProperties.input` file and `makeNewWaveTheory` script are explained in Chapter 3, in detail.

In the scope of this project, only the potential current theory class using the `waveFoam` solver is examined and pointed out the need for the shear current mixing length theory implementation.

2.2 Simple (uniform) current

The simple (uniform) current also named potential current introduces a current that is uniform over the entire water depth as stated in Eq. (2.1).

$$\begin{aligned} \overline{u} &= U \\ \overline{u} &= f(U) \end{aligned} \tag{2.1}$$

The potential current approach does not resolve different flow characteristics for the entire water depth as the theory assumes a fixed velocity for the entire flow section. Therefore, the results taken from the potential current theory are beneficial for a basic estimate of the effect of potential current on the given wave field. As an example, if one wants to study the sediment motion which mobilizes from the bed and generally transports in the boundary layer (for the current it is the full water depth), the potential current theory results do not give realistic results. Therefore, the shear current theory should be used for that type of study. The shear current theory is explained in the next section.

2.3 Shear current

The shear current mixing length theory from the van Driest solution for the flow above smooth beds is given in Eq. (2.2) where y is the distance from the wall, U_f is the friction velocity, κ is the von Karman constant and A_d is the damping coefficient [3]. The van Driest velocity distribution derived from the mixing length theory reduces to linear velocity distribution for small values of y^+, whereas for large values of y^+, it reduces to logarithmic velocity distribution.

$$\begin{aligned} &\overline{u} = 2U_f \int_0^{y^+} \frac{dy^+}{1 + \{1 + 4\kappa^2 {y^+}^2 [1 - \exp(-\frac{y^+}{A_d})]^2\}^{1/2}} \\ &\text{where} \quad y^+ = \frac{yU_f}{\nu}, \quad \kappa = 0.4, \quad \text{and} \quad A_d = 25 \end{aligned} \tag{2.2}$$

The effect of bed roughness should be implemented inside the van Driest solution considering the velocity distribution for the flow above both smooth and rough beds. The effect of bed roughness implemented to the van Driest solution with the extension of Cebeci & Chang is shown in Eq. (2.3) where the term Δy is the coordinate displacement (coordinate shift), and k_s is the Nikuradse's equivalent sand roughness [3]. The mean velocity distribution, $\overline{u}$, considering the van Driest solution with the extension of Cebeci & Chang is given in Eq. (2.4).

$$\begin{aligned} &\Delta y^+ = 0.9[\sqrt{k_s^+} - k_s^+ \exp(-\frac{k_s^+}{6})] \qquad \text{if} \qquad 5 < k_s^+ < 2000 \\ &\text{where} \quad k_s^+ = \frac{k_s U_f}{\nu} \end{aligned} \tag{2.3}$$

$$\begin{aligned} &\overline{u} = 2U_f \int_0^{y^+} \frac{dy^+}{1 + \{1 + 4\kappa^2 (y^+ + \Delta y^+)^2 [1 - \exp(-\frac{(y^+ + \Delta y^+)}{A_d})]^2\}^{1/2}} \\ &\overline{u} = f(U_f, y, k_s, \nu) \end{aligned} \tag{2.4}$$

As an example of the possible application, one can use the velocity distribution calculated from the Eq. (2.4) for the sediment transport problems, as the calculated velocity distribution resolves the flow across the entire section such as viscous sublayer, buffer layer, logarithmic layer, and outer region considering the effect of the bed roughness. An example of van Driest velocity profiles for different values of roughness taken from the "Sumer and Fuhrman [3]" is given in Figure 2.1.

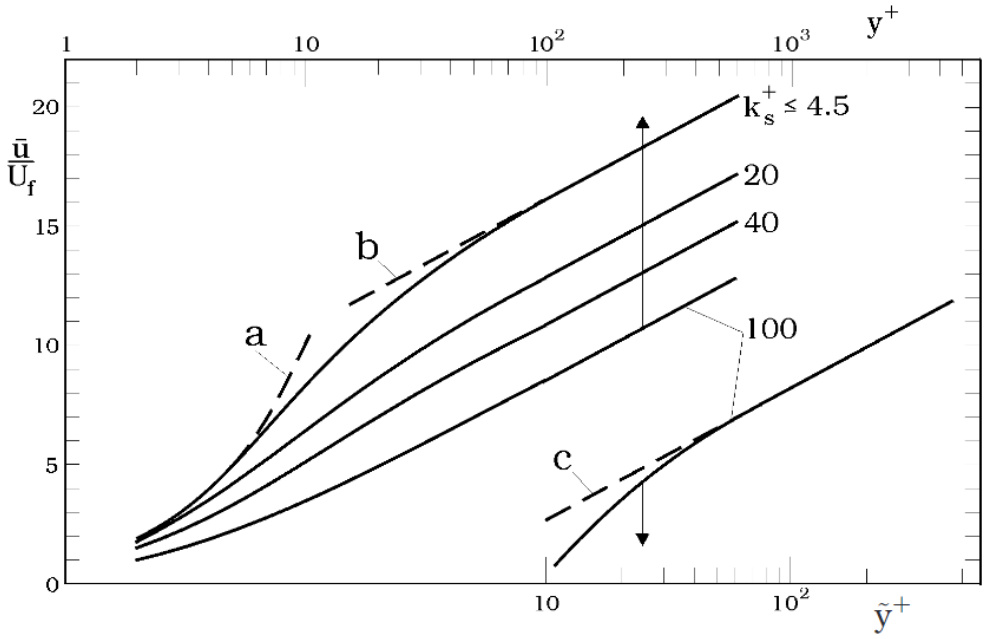

Figure 2.1: Example of van Driest velocity profiles for different values of roughness (Taken from [3], Figure 3.21 at page 101)

Chapter 3

Library and Class Structure of waves2Foam Toolbox

3.1 `waveTheory` class and sub-classes

The `waves2Foam` toolbox has a wide range of wave theories. The `waveTheories` library in the `waves2Foam` toolbox is structured with the names given in the wave theories list in Section 2.1.

As a starter, one should examine the potential current theory library (`potentialCurrent`) in the `waves2Foam` toolbox for understanding the coding structure in `waves2Foam` toolbox. This option is typically used in the `waves2Foam` toolbox applications for outlet relaxation zones, where the velocity vector is set to 0. The `potentialCurrent` library can be found at:

```
$WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2Foam/waveTheories/current/potentialCurrent
```

There are two files in the `potentialCurrent` folder, in which `potentialCurrent.H` is the class declaration and `potentialCurrent.C` is the class definition files. The declaration and definition files of the `potentialCurrent` wave theory library are given in Listing A.1, and Listing A.2, respectively in the Appendix A.

The flow velocity, U (declared as `U_`), T_{soft} (declared as `Tsoft_`), and local sea level (declared as `localSeaLevel_`) are declared in the class declaration part of the `potentialCurrent.H` file shown in Listing 3.1. T_{soft} is used for softening the initial warm-up period of the `waveFoam` solver.

Listing 3.1: Declaration of parameters in `potentialCurrent` library

```
/*---------------------------------------------------------------------------*\
                        Class potentialCurrent Declaration
\*---------------------------------------------------------------------------*/
class potentialCurrent
:
    public waveTheory
{

protected:

    // Protected data
        vector U_;
        scalar Tsoft_;
        scalar localSeaLevel_;
```

In the constructors part of the definition in potentialCurrent.C file, the parameters are read from the related dictionaries. As an example, U, T_{soft}, and local sea level parameters are read from the waveProperties file. These parameters are read in the constructors part of the potentialCurrent.C file shown in Listing 3.2.

Listing 3.2: Reading parameters in potentialCurrent library definition file

```
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * //
potentialCurrent::potentialCurrent
(
    const word& subDictName,
    const fvMesh& mesh_
)
:
    waveTheory(subDictName, mesh_),
    U_(vector(coeffDict_.lookup("U"))),
    Tsoft_(readScalar(coeffDict_.lookup("Tsoft"))),
    localSeaLevel_
    (
        coeffDict_.lookupOrDefault<scalar>("localSeaLevel", seaLevel_)
    )
```

In the member functions part of the potentialCurrent.C file, there is a factor calculation part for softening the calculated parameters within the specified time, T_{soft} for the initial part of the simulation. In the potentialCurrent, this factor is used for relaxing the U values. The related sections of member functions part of the potentialCurrent.C file is shown in Listing 3.3.

Listing 3.3: Member functions in potentialCurrent library definition file

```
// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //
scalar potentialCurrent::factor(const scalar& time) const
{
    scalar factor(1);
    if (Tsoft_ > 0.0)
    {
        factor = Foam::sin(PI_/2.0/Tsoft_*Foam::min(Tsoft_, time));
    }

    return factor;
}
.
.
.
vector potentialCurrent::U
(
    const point& x,
    const scalar& time
) const
{
    return (U_*factor(time));
}
```

3.2 waveProperties.input file structure

An example of waveProperties.input file is given in Listing 3.4. waveFoam solver uses relaxation zones for removing the spurious reflections from the numerical simulations. Therefore, one should give a proper relaxation method considering the inlet and outlet regions of the computational domain.

One has to do the preparation before executing waveFoam solver. The waveProperties.input file has to be converted to waveProperties file by executing the setWaveParameters application, where all the necessary wave parameters are transferred based on physical meaningful properties with the given application [2]. Then, the setWaveField application should be executed. The setWaveField application sets the initial conditions that the user defined in the wave theory [2].

Listing 3.4: Example of waveProperties.input file

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
seaLevel    0.00;
// A list of the relaxation zones in the simulation. The parameters are given
// in <name>Coeffs below.
relaxationNames (inlet outlet);
initializationName outlet;
inletCoeffs
{
    // Wave type to be used at boundary "inlet" and in relaxation zone "inlet"
    waveType    stokesFirst;
    // Ramp time of 2 s
    Tsoft       2;
    // Water depth at the boundary and in the relaxation zone
    depth       0.400000;
    // Wave period
    period      2.0;
    // Phase shift in the wave
    phi         0.000000;
    // Wave number vector, k.
    direction  (1.0 0.0 0.0);
    // Wave height
    height      0.1;
    // Specifications on the relaxation zone shape and relaxation scheme
    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;
        relaxType   INLET;
        startX      (0 0.0 -1);
        endX        (5 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};
outletCoeffs
{
    waveType    potentialCurrent;
    U           (0 0 0);
    Tsoft       2;
    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;
        relaxType   OUTLET;
        startX      (13 0.0 -1);
        endX        (18 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};
// ************************************************************************* //
```

3.3 makeNewWaveTheory script

The waves2Foam toolbox has a special coding script that should be followed when creating a new wave theory inside this toolbox. Therefore, the creator of the waves2Foam toolbox has templated a small script named makeNewWaveTheory that creates a new wave theory with the given name. Then, the created folder, which contains the declaration and definition files for the new wave theory, can be transferred to the related wave theory's sub-folder inside the waveTheories dictionary. The makeNewWaveTheory can be called inside the waves2Foam toolbox folder where it is compiled. An example code for creating a new shearCurrent wave theory library for the waves2Foam toolbox is given below in Listing 3.5.

Listing 3.5: Creating shearCurrent library script

```
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/doc/templateWaveTheory
./makeNewWaveTheory shearCurrent
mv shearCurrent $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2Foam/waveTheories/current
```

The created declaration (shearCurrent.H) and definition (shearCurrent.C) files of the new shearCurrent wave theory library after executing the makeNewWaveTheory application are given in Listing B.1, and Listing B.2, respectively in the Appendix B. The implementation of the shear current theory inside of these given blank declaration and definition classes are given in Chapter 4, in detail.

Chapter 4

Modification

4.1 New shearCurrent library

The new `shearCurrent` library is created from scratch. Therefore, after creating the `shearCurrent` scripts with the help of `makeNewWaveTheory` application mentioned in Chapter 3, one should implement all the necessary details to the declaration and definition files of the new library. The directory path of the new `shearCurrent` library created by the `makeNewWaveTheory` application is shown in Listing 4.1.

Listing 4.1: Location of `shearCurrent` library

```
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2Foam/waveTheories/current
```

The declaration file of the `shearCurrent` library is created based on the previously created `shearCurrent.C` script. Friction velocity, U_f (declared as `Uf_`), fluid kinematic viscosity, ν (declared as `nu_`), Nikuradse's equivalent sand roughness, k_s (declared as `ks_`), von Karman constant, κ (declared as `kappa_`), and damping coefficient, A_d (declared as `Ad_`) are added to the class declaration part of the `shearCurrent.H` file as shown in Listing 4.2. T_{soft} (declared as `Tsoft_`), and local sea level (declared as `localSeaLevel_`) are also added to the declaration classes for keeping the functionality of the `wave2Foam` toolbox, as `wave2Foam` toolbox has lots of other wave theories which can be used together with the `combinedWaves` functionality mentioned in Chapter 2. In addition, as the van Driest formulation uses the distances from the wall, these distances of each cell should be stored for the calculation. Therefore, the wall distances of each cell center are stored under the name y (declared as `y_`). The points for each cell in the computational domain are also stored (declared as `x_`) which can be used for indexing in the van Driest cumulative trapezoidal integration part.

Listing 4.2: Declared classes inside the `shearCurrent.H`

```
/*---------------------------------------------------------------------------*\
                        Class shearCurrent Declaration
\*---------------------------------------------------------------------------*/
protected:
    // Protected data
    scalar Tsoft_;
    scalar localSeaLevel_;
    scalar Uf_;
    scalar nu_;
    scalar ks_;
    scalar kappa_;
    scalar Ad_;
    const volScalarField& y_;
    const volVectorField& x_;
```

The member functions part of the declaration file of the shearCurrent library is shown in Listing 4.3. Member functions section in the declaration file is created with the help of the potentialCurrent declaration file as the new shearCurrent library is implemented in the current wave theory folder. For keeping the different functionalities in the wave2Foam toolbox like combinedWaves functionality stated before, the potentialCurrent library is considered as the base for the shearCurrent theory to be able to cover the functionality for the current wave theory library.

Listing 4.3: Declared member functions inside the shearCurrent.H

```
    // Member Functions
        virtual Switch conflictTSoftInitialise() const
        {
            if (Tsoft_ > 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        };

        void printCoeffs();

        virtual scalar eta
        (
            const point&,
            const scalar&
        ) const;

        virtual scalar pExcess
        (
            const point&,
            const scalar&
        ) const;

        bool implementPressure() const
        {
            return true;
        };

        virtual vector U
        (
            const point&,
            const scalar&
        ) const;
};
```

The definition file of the shearCurrent library is created from the previously created shearCurrent.C script. Friction velocity, U_f, fluid kinematic viscosity, ν, Nikuradse's equivalent sand roughness, k_s, von Karman constant, κ, the damping coefficient, A_d, the distance from the wall, and volumetric vector field containing the point coordinates are added to the class definition of the shearCurrent.C file as shown in Listing 4.4. Although von Karman constant, κ, and the damping coefficient, A_d are given with their default values, one can change these values by declaring them in waveProperties.input file. Also, T_{soft}, and local sea level are added to the definition classes for keeping the functionality of the wave2Foam toolbox as discussed before. In addition, as the distances from the wall are calculated with the help of the wallDist function embedded in the OpenFOAM library, #include "wallDist.H" should be added to the #include section which is located on the top of the definition file. At the end, x_ (volVectorField) is constructed with the center coordinates of each cell member in the mesh_ structure (mesh_.C()).

Listing 4.4: Definition of constructors inside the shearCurrent.C

```
\*---------------------------------------------------------------------------*/
#include "wallDist.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
.
.
.
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * //

shearCurrent::shearCurrent
(
    const word& subDictName,
    const fvMesh& mesh_
)
:
    waveTheory(subDictName, mesh_),
    Tsoft_(readScalar(coeffDict_.lookup("Tsoft"))),
    localSeaLevel_
    (
        coeffDict_.lookupOrDefault<scalar>("localSeaLevel", seaLevel_)
    ),
    Uf_(readScalar(coeffDict_.lookup("Uf"))),
    nu_(readScalar(coeffDict_.lookup("nu"))),
    ks_(readScalar(coeffDict_.lookup("ks"))),
    kappa_
    (
        coeffDict_.lookupOrDefault<scalar>
        (
            "kappa",
            0.41
        )
    ),
    Ad_
    (
        coeffDict_.lookupOrDefault<scalar>
        (
            "Ad",
            25.0
        )
    ),
    y_(wallDist::New(mesh_).y()),
    x_(mesh_.C())
```

The steps for implementing the member functions inside the definition file of the shearCurrent library from the blank shearCurrent.C script are itemized below. As the factor member function is kept as it is as in the script, the factor section is not demonstrated.

- Eta definition

  The old definition of the eta member function is changed to its new definition considering the given input of local sea level in the waveProperties.input file. The new definition of the eta member function is shown below.

```
scalar shearCurrent::eta
(
    const point& x,
    const scalar& time
) const
{
    scalar eta = localSeaLevel_;
    return eta;
}
```

- Excess pressure definition

  The old definition of the excess pressure member function is changed to its new definition considering the excess pressure created by the given local sea level. This pressure definition is also used by the potentialCurrent library, therefore, the definition is directly taken from the potentialCurrent.C for keeping the functionality of the wave2Foam toolbox discussed before. The new definition of the excess pressure member function is shown below.

```
scalar shearCurrent::pExcess
(
    const point& x,
    const scalar& time
) const
{
    return referencePressure(localSeaLevel_);
}
```

- Flow velocity definition

  The old definition of flow velocity member function is changed to its new definition considering the van Driest solution with the extension of Cebeci & Chang by adding the effect of the bed roughness as stated in Chapter 2. The extensions of k_s^+ (defined as ksPlus_), and Δy^+ (defined as deltayPlus_) by Cebeci & Chang are calculated before the main forAll loop. Then, these values are used for finding the effect of the bed roughness on the mean streamwise shear current velocities at each cell center calculated in the forAll loop. Also, temporary volumetric scalar fields temp and temp2 are defined by getting the properties of the y_ (later all the elements inside the temporary object is set to "0") for calculating and storing the streamwise velocities at each cell center in the volumetric field. Later, these temporary objects are also used in the cumulative trapezoidal calculation as shown in the van Driest formulation's integration part.

  In the forAll loop, calculated vanDriest values are stored in both the temp and temp2 volumetric fields. Then, the mean streamwise velocities at each cell are found with the cumulative trapezoidal integration over the water depth with the calculated y^+ values as given in the van Driest solution. After the cumulative trapezoidal integration part, the searched cell for assigning the calculated temp[index] as the mean streamwise velocity is found with the help of the if function where the actual point at that instant is checked with the previously structured x_ volumetric vector field. Then, the calculated summation value of temp[index] is assigned as the mean streamwise shear current (declared as shearU_) at the found cell center with the cell index. Later, in the return part, the assigned scalar shearU_ is converted to vector by multiplying the value to vector(1.0,0,0) as the mean flow is pointing to the streamwise direction. Also, for the T_{soft} functionality, the final vector is multiplied by the factor(time).

```
vector shearCurrent::U
(
    const point& x,
    const scalar& time
) const
{

    volScalarField temp=y_;
    volScalarField temp2=y_;

    temp *=scalar(0.0);
    temp2 *=scalar(0.0);

    scalar yPlus_ = 0.0;
    scalar vanDriest_ = 0.0;

    scalar ksPlus_= ks_*Uf_/nu_;
    scalar deltayPlus_=0.9*(Foam::sqrt(ksPlus_)-ksPlus_*Foam::exp(-ksPlus_/6.0));

    scalar shearU_ = 0.0;

    forAll(temp,index)
    {

        yPlus_=y_[index]*Uf_/nu_;

        vanDriest_=1.0/(1.0+Foam::sqrt(1.0+4.0*Foam::pow(kappa_,2.0)*Foam::pow(yPlus_+deltayPlus_,2.0)*Foam::pow
     (1.0-Foam::exp(-(yPlus_+deltayPlus_)/Ad_),2.0)));

        temp[index] =vanDriest_;
        temp2[index] =vanDriest_;

        if ( index > 0)

        {
            temp[index] = temp[index-1] + 0.5*(temp2[index] + temp2[index-1])*(y_[index]*Uf_/nu_ - y_[index-1]*
     Uf_/nu_);
        }

        if ( x_[index] == x)

        {
            shearU_ = 2.0*Uf_*temp[index];
        }

    }

    return shearU_*vector(1.0,0,0)*factor(time);
}
```

One can find the final declaration (`shearCurrent.H`) and definition (`shearCurrent.C`) files of the `shearCurrent` wave theory library with the all changes mentioned above in Listing B.3, and Listing B.4, respectively in Appendix B.

4.2 Modification of Make folder

The new `shearCurrent` wave theory library is implemented both for `shearCurrent` library and `setWaveParameters` function. While the `Make` folder for the `shearCurrent` library is inside the `waves2Foam` sub-folder, the `setWaveParameters` function for the `shearCurrent` library is located at the `waves2FoamProcessing/preProcessing` sub-folder. Therefore, one should add the new `shearCurrent` wave theory library path into the `files` file inside the `Make` folders and re-compile the `src` folder. These mentioned `Make` folders can be found at:

```
$WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2Foam
$WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2FoamProcessing
```

One can create backups for the `Make` folders in case of any problem. An example code for creating backups for the `Make` folders can be found in Listing 4.5.

Listing 4.5: An example code for the backup of `Make` folders

```
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2Foam
cp -r Make Make_backup
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2FoamProcessing
cp -r Make Make_backup
```

Changes in the `Make` folder both for `shearCurrent` library and `setWaveParameters` function are itemized below.

- Adding `shearCurrent` library path to the `Make/files` for the `waves2Foam` solver

  The `files` file located under the `src/waves2Foam/Make` folder has to be modified considering the new `shearCurrent` library. As the `shearCurrent` library is implemented under the `waveTheories/current` wave theory folder, a directory line pointing the location of the `shearCurrent.C` file should be added after the `potentialCurrent` wave theory line, which is located to the `/*WAVE THEORIES*/ - /*current type*/` part of the `files` file. The updated part of the `files` file is shown below.

```
/* WAVE THEORIES */

/* Current type */
current=current
$(waveTheories)/$(current)/potentialCurrent/potentialCurrent.C
$(waveTheories)/$(current)/shearCurrent/shearCurrent.C
```

- Adding `shearCurrentProperties` path to the `Make/files` for the `setWaveParameters` function

  As explained in Chapter 3, the `waveProperties.input` file has to be converted to `waveProperties` file by executing the `setWaveParameters` application. Therefore, as a starter `shearCurrentProperties` folder should be created under the `setWaveProperties/current` folder. The `potentialCurrentProperties` folder and its declaration and definition files can be used as a template for the `shearCurrentProperties`. An example code for creating `shearCurrentProperties` folder and its declaration and definition files from the `potentialCurrentProperties` is given in Listing 4.6. After copying the `potentialCurrentProperties` folder, one should change the old file names to `shearCurrentProperties` with `mv` (move) command. Then, the contents of the declaration and definition files should be changed with the `sed` command for changing `potentialCurrent` to `shearCurrent`.

Listing 4.6: Creating `shearCurrentProperties` functionality in `setWaveProperties` class

```
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2FoamProcessing/preProcessing/
    setWaveProperties/current
cp -r potentialCurrentProperties shearCurrentProperties
cd shearCurrentProperties
mv potentialCurrentProperties.C shearCurrentProperties.C
mv potentialCurrentProperties.H shearCurrentProperties.H
sed -i s/potentialCurrent/shearCurrent/g shearCurrentProperties.C
sed -i s/potentialCurrent/shearCurrent/g shearCurrentProperties.H
```

After creating the `shearCurrentProperties` declaration and definition files, one should change the member functions sections of the definition file `shearCurrentProperties.C`. As, `shearCurrent` library requires the declarations of friction velocity, U_f, fluid kinematic viscosity, ν, Nikuradse's equivalent sand roughness, k_s, von Karman constant, κ, the damping coefficient, A_d, T_{soft}, and local sea level in the `waveProperties` file, these inputs has to be taken from the `waveProperties.input` file. The changes to cover mentioned inputs are shown below for the member functions section of the `shearCurrentProperties.C` file.

```
// Write the already given parameters
writeGiven( os, "waveType" );
writeGiven( os, "Tsoft" );
writeGiven( os, "Uf");
writeGiven( os, "nu");
writeGiven( os, "ks");

    if (dict_.found( "localSeaLevel" ))
    {
        writeGiven( os, "localSeaLevel");
    }
    if (dict_.found( "kappa" ))
    {
        writeGiven( os, "kappa");
    }
    if (dict_.found( "Ad" ))
    {
        writeGiven( os, "Ad");
    }
```

One can find the final declaration (`shearCurrentProperties.H`) and definition (`shearCurrentProperties.C`) files of the `shearCurrentProperties` with the all changes mentioned above in Listing B.5, and Listing B.6, respectively in Appendix B.

After creating the `potentialCurrentProperties` folder, one should add the directory of the `potentialCurrentProperties.C` file to the `files` file located under the `src/waves2FoamProcessing/Make` folder. A directory line pointing the location of the `potentialCurrentProperties.C` file should be added after the `potentialCurrentProperties.C` line, which is located to the `/*current type*/` part of the `files` file. The updated part of the `files` file is shown below.

```
/* Current type */
current=current
$(waveProp)/$(current)/potentialCurrentProperties/potentialCurrentProperties.C
$(waveProp)/$(current)/shearCurrentProperties/shearCurrentProperties.C
```

After creating all the necessary steps for the `shearCurrent` wave theory mentioned above, one should re-compile the `waves2Foam` and `waves2FoamProcessing/preProcess` wave theory libraries. Before the re-compilation, one should clean the `Make` folders by using `wclean` command. An example code for the `wclean` of the related `Make` folders is shown in Listing 4.7.

Listing 4.7: An example code for the `wclean` of the related `Make` folders

```
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2Foam
wclean
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src/waves2FoamProcessing
wclean
```

After the cleaning process, one can recompile all the contents under the `src` folder by executing the `Allwmake` file. An example recompilation code is shown below.

```
cd $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/src
./Allwmake
```

Chapter 5

Tutorial Case

5.1 Preparation

The tutorial cases can be found in the `waves2Foam_tutorials` folder in the `waves2Foam` toolbox. The `waveFlume` tutorial inside the `waveFoam` dictionary is selected for creating the `shearCurrent` tutorial, which is named as `currentFlume`. The `waveFlume` tutorial is composed of two sections where there are fluid (like water) and gas (like air) sections. The computational domain dimensions of the `waveFlume` tutorial are 18 m in length, 0.6 m in height, and 0.1 m in thickness. While the fluid section is positioned at the bottom with 0.4 m depth, the gas section is positioned at the top with 0.2 m height. One can find detailed information about the `waveFlume` tutorial in `waves2Foam` manual [2]. An example code for copying the `waveFlume` tutorial to the `$WM_PROJECT_USER_DIR/run` folder by changing it's name to `currentFlume` is given below in Listing 5.1.

Listing 5.1: An example code for copying the `waveFlume` tutorial

```
mkdir --parents $WM_PROJECT_USER_DIR/run/currentTutorial/tutorials/waveFoam
cp -r $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/tutorials/waveFoam/waveFlume $WM_PROJECT_USER_DIR/
    run/currentTutorial/tutorials/waveFoam/currentFlume
cp -r $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/tutorials/commonFiles $WM_PROJECT_USER_DIR/run/
    currentTutorial/tutorials
cp -r $WM_PROJECT_USER_DIR/applications/utilities/waves2Foam/bin $WM_PROJECT_USER_DIR/run/currentTutorial
cd $WM_PROJECT_USER_DIR/run/currentTutorial/tutorials/waveFoam/currentFlume
sed -i 's/20/10/g' system/controlDict
```

As a starter, the new `currentFlume` tutorial can be checked by implementing the available `potentialCurrent` wave theory tutorial by changing the `waveProperties.input` file. The old `waveProperties.input` file in the `constant` folder should be changed to Listing 5.2 considering the `potentialCurrent` library where the streamwise current velocity is set to 1 m/s with the Froude number equal to 0.5. Also, the type of the outlet boundary condition of the flow velocity, `U`, located at the `0.org/U.org` file should be changed from `fixedValue` to `zeroGradient` boundary condition. Then, one can run the `potentialCurrent` tutorial by executing the `Allrun` file in the `currentFlume` folder. The graphical result of the `potentialCurrent` tutorial at $t = 10$ s is also shown in Figure 5.1.

Listing 5.2: `waveProperties.input` file of the `potentialCurrent` tutorial

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
seaLevel    0.00;

relaxationNames (inlet outlet);

initializationName outlet;

inletCoeffs
{
    waveType    potentialCurrent;
    U           (1 0 0);
    Tsoft       0;

    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;

        relaxType   INLET;
        startX      (0 0.0 -1);
        endX        (5 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};

outletCoeffs
{
    waveType    potentialCurrent;
    U           (1 0 0);
    Tsoft       0;

    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;

        relaxType   OUTLET;
        startX      (13 0.0 -1);
        endX        (18 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};
// ************************************************************************* //
```

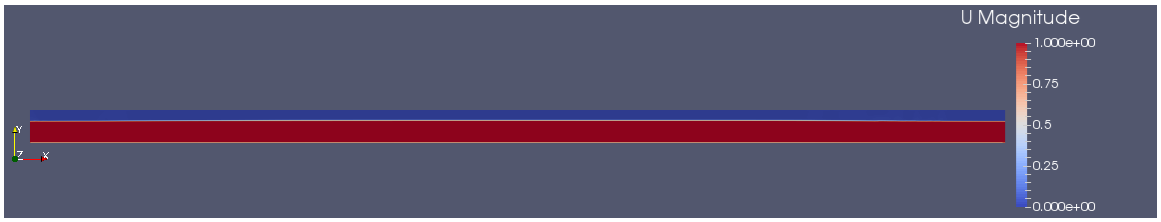

Figure 5.1: Example `potentialCurrent` tutorial result at $t = 10$ s

5.2 shearCurrent tutorial

The currentFlume tutorial mentioned in the preparation section is converted for the shearCurrent tutorial. First, one has to change the context of the waveProperties.input file for the shearCurrent tutorial. The new waveProperties.input file for the shearCurrent wave theory can be seen in Listing 5.3. Additionally, one can adjust the direction (positive or negative) of the shear current by manipulating the sign of the friction velocity, U_f. The depth-averaged velocity calculated from the given $U_f = 0.01$ m/s is calculated as 0.215 m/s by using Eq. (2.4). Also, the k_s value is taken as $1.25 * 10^{-3}$ m. The dimensionless numbers for the shearCurrent tutorial are calculated as in Eq. (5.1), shown below.

$$Re_\tau = \frac{U_f * D}{\nu} = 4000, \quad Fr = \frac{V}{\sqrt{g * D}} = 0.108, \quad \text{and} \quad k_s^+ = \frac{U_f * k_s}{\nu} = 12.5 \tag{5.1}$$

Listing 5.3: waveProperties.input file of the shearCurrent tutorial

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
seaLevel    0.00;
relaxationNames (inlet outlet);
initializationName outlet;
inletCoeffs
{
    waveType    shearCurrent;
    Tsoft       0;
    Uf          0.01;       // Friction velocity (m/s)
    nu          1.0e-06;        // Fluid kinematic viscosity (m^2/s)
    ks          1.25e-03;  // Nikuradse's equivalent sand roughness (m)
    //Optional entries
    //kappa         0.41;   // von Karman constant
    //Ad            25.0;   // Damping coefficient
    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;

        relaxType   INLET;
        startX      (0 0.0 -1);
        endX        (5 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};
outletCoeffs
{
    waveType    shearCurrent;
    Tsoft       0;
    Uf          0.01;       // Friction velocity (m/s)
    nu          1.0e-06;        // Fluid kinematic viscosity (m^2/s)
    ks          1.25e-03;  // Nikuradse's equivalent sand roughness (m)
    //Optional entries
    //kappa         0.41;   // von Karman constant
    //Ad            25.0;   // Damping coefficient
    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;

        relaxType   OUTLET;
        startX      (13 0.0 -1);
        endX        (18 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};
// ************************************************************************* //
```

Calculation of the wall distances of each mesh cell is done by the `wallDist` function as stated in Chapter 4. Therefore, the method for the `wallDist` function should be added at the end of the `fvSchemes` file in the `system` folder. The added lines which specify the method for the `wallDist` function are shown below.

```
wallDist
{
    method          meshWave;
}
```

Additionally, U values of each cell center are sampled for comparing the simulated results with the theoretical van Driest solution with the extension of Cebeci & Chang given in Eq. (2.4). An example code for sampling the U values is given in Listing 5.4. One can put this `sample` file inside the `system` folder. Then after the simulation completed, this `sample` file can be executed with `postProcess -func sample` command.

Listing 5.4: `sample` file of the `shearCurrent` tutorial

```
type            sets;
libs            (sampling);

writeControl    onEnd;

interpolationScheme cellPoint;

setFormat       raw;

sets
(
    x1
    {
        type    face;
        axis    y;
        start   (1 -0.4 0);
        end     (1 0 0);
        nPoints 100;
    }
);

fields          (U);
```

After cleaning the time folders except the `0` and `0.org` and other unnecessary files (log, postprocess etc.) in the `currentFlume` tutorial, one can execute the new `shearCurrent` tutorial. An example code for executing the `currentFlume` tutorial is given in Listing 5.5.

Listing 5.5: An example code for executing the `currentFlume` tutorial

```
foamListTimes -rm
yes | rm log.*
rm -rf postProcessing
rm -rf waveGaugesNProbes

setWaveParameters
setWaveField
waveFoam
postProcess -func sample
```

The graphical result of the `shearCurrent` tutorial at $t = 10$ s is shown in Figure 5.2. Also, a zoomed inlet section in Figure 5.2 is shown in Figure 5.3 where the sampled inlet velocity profile is compared with the van Driest profile (Eq. (2.4)) in Figure 5.4.

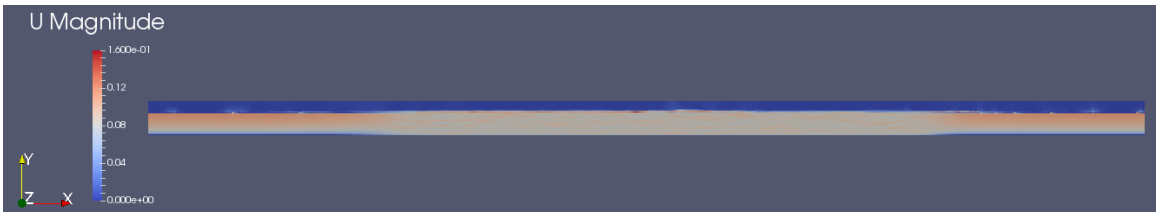

Figure 5.2: Example `shearCurrent` tutorial result at $t = 10$ s

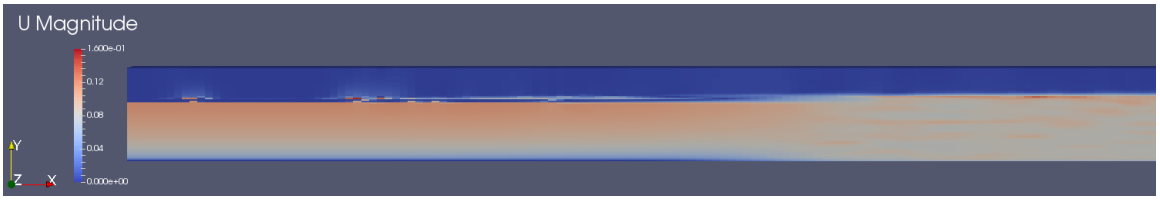

Figure 5.3: Closer look at the example `shearCurrent` tutorial result at $t = 10$ s

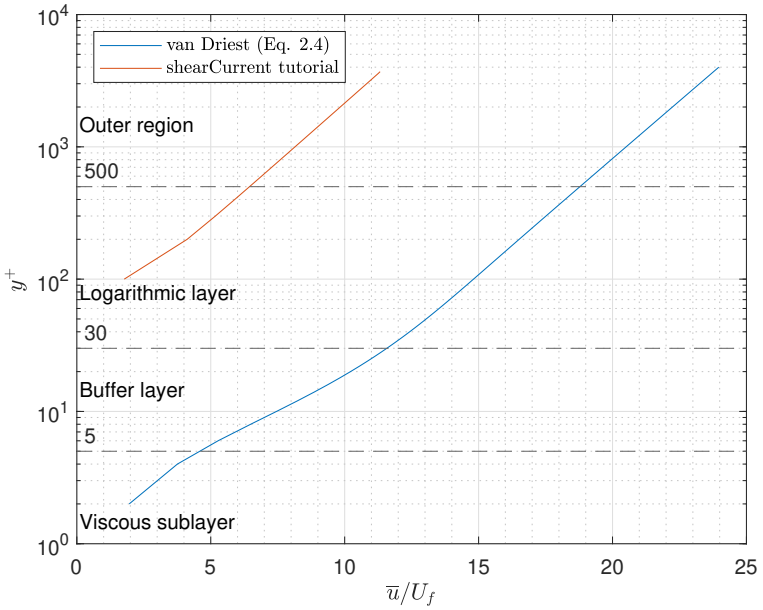

Figure 5.4: Comparison of `shearCurrent` tutorial sampled (inlet) result at $t = 10$ s with van Driest profile (Eq. (2.4))

As seen from Figure 5.3, the mean velocity profile is increasing from the wall to the free surface which is expected from the van Driest velocity profile. Although it seems that the `shearCurrent` library is giving reasonable results, one can notice problems in Figure 5.2. In Figure 5.2, it can be seen that the fluid section of the computational domain is not homogeneous considering the van Driest velocity profile, where the velocities of the entire section should be 0 at the wall, then

increasing up to the free surface like the inlet section in Figure 5.3. Additionally, it can be seen from Figure 5.4 that although the slopes of the simulated and theoretical velocity profiles are nearly the same for logarithmic layer and outer region sections, they are shifted from each other. A possible reason for this problem is the mesh density of the tutorial which is not dense enough to resolve the section close to the wall. As van Driest's solution uses cumulative integration, the streamwise velocity values at that unresolved section could not be added to the computation, therefore due to the lack of these velocity values contribution to the van Driest velocity profile, there is a shift between the computed and theoretical results. Although this may be a possible problem, due to the computational time, the mesh density has not been increased as the simulation has not finished within a reasonable time.

One can notice that there are some nonphysical small velocity pockets inside the fluid section. The reason for this problem can be attributed to a bug inside of the newly implemented `shearCurrent` wave theory library or as `waves2Foam` toolbox is also manipulating the flow velocities at the background considering the relaxation zones etc., there can be problems regarding the `shearCurrent` library which is not fully adapted considering the internal functionalities in the `waves2Foam` toolbox. Additionally, one can notice that there is a significant computational time increase from the `potentialCurrent` tutorial to `shearCurrent` tutorial. This increase may be the effect of the `forAll` loop used for the calculation of the cumulative integration part.

As a different approach to see the reaction of `waves2Foam` toolbox to try to alter possible mentioned problems, the relaxation zone for the inlet and outlet area is increased where both relaxation zones are structured as fully covering the entire fluid region (from `startX`=0 to `endX`=18 m). The new trial result for the `shearCurrent` tutorial at $t = 10$ s is shown Figure 5.5. It can be seen from Figure 5.5 that the fluid domain is showing expected results considering the velocity increase from the wall to the free surface for the entire fluid region, homogeneously. Although this shows that the `shearCurrent` theory is working, as the relaxation zone is set for the entire domain, the `shearCurrent` theory is not projected across the domain as an initial condition. Also, one can see unexpected pockets of high-velocity regions on some parts of the free surface. Therefore, one should consider the possible problems for the `shearCurrent` library and tutorial.

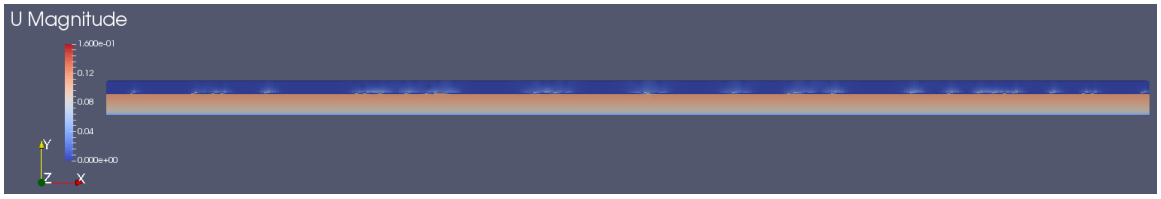

Figure 5.5: Example trial with full relaxation zone for `shearCurrent` tutorial result at $t = 10$ s

Bibliography

[1] N. G. Jacobsen, D. R. Fuhrman, and J. Fredsøe, "A wave generation toolbox for the open-source CFD library: OpenFoam®," *International Journal for Numerical Methods in Fluids*, vol. 70, no. 9, pp. 1073–1088, 2012.

[2] N. G. Jacobsen, "waves2foam manual," 2017.

[3] B. M. Sumer and D. R. Fuhrman, *Turbulence in Coastal and Civil Engineering*. World Scientific, 2020.

[4] "Waves2Foam." https://www.openfoamwiki.net/index.php/Contrib/waves2Foam. Accessed: 2021-12-17.

Study questions

1. Can `waves2Foam` be used without OpenFOAM?
2. What are the main differences between the potential current and shear current considering the velocity distributions?
3. What is the main parts and functionalities of the `waves2Foam` toolbox used in this project?
4. How is the integration in the van Driest solution handled in the newly implemented `shearCurrent` wave theory library?
5. What are the problems seen in the `shearCurrent` tutorial in Chapter 5?

Appendix A

potentialCurrent

A.1 potentialCurrent declaration file

Listing A.1: potentialCurrent.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Class
    Foam::waveTheories::potentialCurrent

Description
    Implementation of a potential current velocity profile, e.g. uniform
    velocity distribution over the vertical. In the special case of
    U = vector::zero it can be used as outlet sponge layer wave type.

    A description of the general wave framework is given in

    @article { jacobsenFuhrmanFredsoe2011,
        Author = {Jacobsen, N G and Fuhrman, D R and Freds\o{}e, J},
        title = {{A Wave Generation Toolbox for the Open-Source CFD Library: OpenFoam\textregistered
     {}}},
        Journal = {{Int. J. for Numer. Meth. Fluids}},
        Year = {2012},
        Volume = {70},
        Number = {9},
        Pages = {1073-1088},
        DOI = {{10.1002/fld.2726}},
```

```
    }

SourceFiles
    potentialCurrent.C

Author
    Niels Gjoel Jacobsen, Technical University of Denmark.  All rights reserved.

\*---------------------------------------------------------------------------*/

#ifndef potentialCurrent_H
#define potentialCurrent_H

#include "waveTheory.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace waveTheories
{

/*---------------------------------------------------------------------------*\
                          Class potentialCurrent Declaration
\*---------------------------------------------------------------------------*/

class potentialCurrent
:
    public waveTheory
{

protected:

    // Protected data
        vector U_;
        scalar Tsoft_;
        scalar localSeaLevel_;

    // Protected member functions

        scalar factor(const scalar&) const;
public:

    //- Runtime type information
    TypeName("potentialCurrent");

    // Constructors

        //- from components
        potentialCurrent
        (
            const word&,
            const fvMesh& mesh_
        );


    // Destructor

        ~potentialCurrent()
        {}


    // Member Functions
        virtual Switch conflictTSoftInitialise() const
        {
            if (Foam::mag(U_) > SMALL && Tsoft_ > 0)
            {
                return true;
```

```
112             }
113             else
114             {
115                 return false;
116             }
117         };
118
119         void printCoeffs();
120
121         virtual scalar eta
122         (
123             const point&,
124             const scalar&
125         ) const;
126
127 //        virtual scalar ddxPd
128 //        (
129 //            const point&,
130 //            const scalar&,
131 //            const vector&
132 //        ) const;
133
134         virtual scalar pExcess
135         (
136             const point&,
137             const scalar&
138         ) const;
139
140         bool implementPressure() const
141         {
142             return true;
143         };
144
145         virtual vector U
146         (
147             const point&,
148             const scalar&
149         ) const;
150
151         inline virtual vector currentSpeed()
152         {
153             return U_;
154         };
155 };
156
157
158 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
159
160 } // End namespace waveTheories
161 } // End namespace Foam
162
163 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
164
165 #endif
166
167 // ************************************************************************* //
```

A.2 potentialCurrent definition file

Listing A.2: potentialCurrent.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

\*---------------------------------------------------------------------------*/

#include "potentialCurrent.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace waveTheories
{

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

defineTypeNameAndDebug(potentialCurrent, 0);
addToRunTimeSelectionTable(waveTheory, potentialCurrent, dictionary);

// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //


potentialCurrent::potentialCurrent
(
    const word& subDictName,
    const fvMesh& mesh_
)
:
    waveTheory(subDictName, mesh_),
    U_(vector(coeffDict_.lookup("U"))),
    Tsoft_(readScalar(coeffDict_.lookup("Tsoft"))),
    localSeaLevel_
    (
        coeffDict_.lookupOrDefault<scalar>("localSeaLevel", seaLevel_)
    )
{}


void potentialCurrent::printCoeffs()
{
    Info << "Loading wave theory: " << typeName << endl;
```

```
}


// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //


scalar potentialCurrent::factor(const scalar& time) const
{
    scalar factor(1);
    if (Tsoft_ > 0.0)
    {
        factor = Foam::sin(PI_/2.0/Tsoft_*Foam::min(Tsoft_, time));
    }

    return factor;
}


scalar potentialCurrent::eta
(
    const point& x,
    const scalar& time
) const
{
//    scalar eta = seaLevel_;
    scalar eta = localSeaLevel_;
    return eta;
}


//scalar potentialCurrent::ddxPd
//(
//    const point& x,
//    const scalar& time,
//    const vector& unitVector
//) const
//{
//    return 0.0;
//}


scalar potentialCurrent::pExcess
(
    const point& x,
    const scalar& time
) const
{
    return referencePressure(localSeaLevel_);
}


vector potentialCurrent::U
(
    const point& x,
    const scalar& time
) const
{
    return (U_*factor(time));
}


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace waveTheories
} // End namespace Foam

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

Appendix B

shearCurrent

B.1 Blank shearCurrent.H and shearCurrent.C files

Listing B.1: Blank templated shearCurrent.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Class
    Foam::waveTheories::shearCurrent

Description
    TEMPLATE WAVE THEORY

    A description of the general wave framework is given in

    @article { jacobsenFuhrmanFredsoe2011,
        Author = {Jacobsen, N G and Fuhrman, D R and Freds\o{}e, J},
        title = {{A Wave Generation Toolbox for the Open-Source CFD Library: OpenFoam\textregistered
     {}}},
        Journal = {{Int. J. for Numer. Meth. Fluids}},
        Year = {2012},
        Volume = {70},
        Number = {9},
        Pages = {1073-1088},
        DOI = {{10.1002/fld.2726}},
    }

```

```
SourceFiles
    shearCurrent.C

Author
    Niels Gjoel Jacobsen, Technical University of Denmark.  All rights reserved.


\*---------------------------------------------------------------------------*/

#ifndef shearCurrent_H
#define shearCurrent_H

#include "waveTheory.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace waveTheories
{

/*---------------------------------------------------------------------------*\
                           Class shearCurrent Declaration
\*---------------------------------------------------------------------------*/

class shearCurrent
:
    public waveTheory
{

protected:

    // Protected data
        //scalar H_;
        //Add the needed the variables
        //scalar Tsoft_;

    // Protected member functions

        scalar factor(const scalar&) const ;
public:

    //- Runtime type information
    TypeName("shearCurrent");

    // Constructors

        //- from components
        shearCurrent
        (
            const word&,
            const fvMesh& mesh_
        );


    // Destructor

        ~shearCurrent()
        {}


    // Member Functions
        Switch conflictTSoftInitialise() const
        {
            if (Tsoft_ > 0)
            {
                return true;
            }
```

```
            else
            {
                return false;
            }
        };

        void printCoeffs();

        scalar eta
        (
            const point&,
            const scalar&
        ) const;

        scalar ddxPd
        (
            const point&,
            const scalar&,
            const vector&
        ) const;

        vector U
        (
            const point&,
            const scalar&
        ) const;
};


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace waveTheories
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

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

Listing B.2: Blank templated `shearCurrent.C`

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

```

```
\*---------------------------------------------------------------------------*/

#include "shearCurrent.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace waveTheories
{

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

defineTypeNameAndDebug(shearCurrent, 0);
addToRunTimeSelectionTable(waveTheory, shearCurrent, dictionary);

// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

shearCurrent::shearCurrent
(
    const word& subDictName,
    const fvMesh& mesh_
)
:
    waveTheory(subDictName, mesh_)//,
    //H_(readScalar(coeffDict_.lookup("height"))),
    //Add the other variables
    //Tsoft_(coeffDict_.lookupOrDefault<scalar>("Tsoft",period_))
{}


void shearCurrent::printCoeffs()
{
    Info << "Loading wave theory: " << typeName << endl;
}

// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //


scalar shearCurrent::factor(const scalar& time) const
{
    scalar factor(1.0);
    if (Tsoft_ > 0.0)
    {
        factor = Foam::sin(2*PI_/(4.0*Tsoft_)*Foam::min(Tsoft_, time));
    }

    return factor;
}

scalar shearCurrent::eta
(
    const point& x,
    const scalar& time
) const
{
    scalar eta = 0.0;

    // Insert expression for eta

    return eta;
}

scalar shearCurrent::ddxPd
(
    const point& x,
    const scalar& time,
```

```
    const vector& unitVector
) const
{
    // This gives the z-coordinate relative to seaLevel
    scalar Z(returnZ(x));

    scalar ddxPd(0);
    // Most theories return 0, as the issue with oblique waves has not
    // yet been derived.

    return ddxPd;
}

vector shearCurrent::U
(
    const point& x,
    const scalar& time
) const
{
    // This gives the z-coordinate relative to seaLevel
    scalar Z(returnZ(x));

    scalar Uhorz(0.0), Uvert(0);

    // Insert expression for Uvert and Uhorz as a function of z-coordinate

    // Scale by ramp-up time
    Uhorz *= factor(time);
    Uvert *= factor(time);

    // Cast into a directed vector. Look into any of the existing waves
    // theories to test how this is done correctly. Without the definitions
    // of the variables, it looks something like:
    //  return Uhorz*k_/K_ - Uvert*direction_;
    // Note "-" because of "g" working in the opposite direction

    return vector::zero;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace waveTheories
} // End namespace Foam

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

B.2 shearCurrent declaration file (shearCurrent.H)

Listing B.3: shearCurrent.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Class
    Foam::waveTheories::shearCurrent

Description
    TEMPLATE WAVE THEORY

    A description of the general wave framework is given in

    @article { jacobsenFuhrmanFredsoe2011,
        Author = {Jacobsen, N G and Fuhrman, D R and Freds\o{}e, J},
        title = {{A Wave Generation Toolbox for the Open-Source CFD Library: OpenFoam\textregistered
     {}}},
        Journal = {{Int. J. for Numer. Meth. Fluids}},
        Year = {2012},
        Volume = {70},
        Number = {9},
        Pages = {1073-1088},
        DOI = {{10.1002/fld.2726}},
    }

SourceFiles
    shearCurrent.C

Author
    Niels Gjoel Jacobsen, Technical University of Denmark.  All rights reserved.


\*---------------------------------------------------------------------------*/

#ifndef shearCurrent_H
#define shearCurrent_H

#include "waveTheory.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace waveTheories
```

```
{

/*---------------------------------------------------------------------------*\
                          Class shearCurrent Declaration
\*---------------------------------------------------------------------------*/

class shearCurrent
:
    public waveTheory
{

protected:

    // Protected data
    scalar Tsoft_;
    scalar localSeaLevel_;
    scalar Uf_;
    scalar nu_;
    scalar ks_;
    scalar kappa_;
    scalar Ad_;

    const volScalarField& y_;
    const volVectorField& x_;

    // Protected member functions

        scalar factor(const scalar&) const ;
public:

    //- Runtime type information
    TypeName("shearCurrent");

    // Constructors

        //- from components
        shearCurrent
        (
            const word&,
            const fvMesh& mesh_
        );


    // Destructor

        ~shearCurrent()
        {}


    // Member Functions
        virtual Switch conflictTSoftInitialise() const
        {
            if (Tsoft_ > 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        };

        void printCoeffs();

        virtual scalar eta
        (
            const point&,
            const scalar&
```

```
        ) const;

        virtual scalar pExcess
        (
            const point&,
            const scalar&
        ) const;

        bool implementPressure() const
        {
            return true;
        };

        virtual vector U
        (
            const point&,
            const scalar&
        ) const;

};


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace waveTheories
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

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

B.3 shearCurrent definition file (shearCurrent.C)

Listing B.4: shearCurrent.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

\*---------------------------------------------------------------------------*/

#include "shearCurrent.H"
#include "addToRunTimeSelectionTable.H"
```

```
29 #include "wallDist.H"
30
31 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
32
33 namespace Foam
34 {
35 namespace waveTheories
36 {
37
38 // * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //
39
40 defineTypeNameAndDebug(shearCurrent, 0);
41 addToRunTimeSelectionTable(waveTheory, shearCurrent, dictionary);
42
43 // * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * //
44
45 shearCurrent::shearCurrent
46 (
47     const word& subDictName,
48     const fvMesh& mesh_
49 )
50 :
51     waveTheory(subDictName, mesh_),
52     Tsoft_(readScalar(coeffDict_.lookup("Tsoft"))),
53     localSeaLevel_
54     (
55         coeffDict_.lookupOrDefault<scalar>("localSeaLevel", seaLevel_)
56     ),
57     Uf_(readScalar(coeffDict_.lookup("Uf"))),
58     nu_(readScalar(coeffDict_.lookup("nu"))),
59     ks_(readScalar(coeffDict_.lookup("ks"))),
60     kappa_
61     (
62         coeffDict_.lookupOrDefault<scalar>
63         (
64             "kappa",
65             0.41
66         )
67     ),
68     Ad_
69     (
70         coeffDict_.lookupOrDefault<scalar>
71         (
72             "Ad",
73             25.0
74         )
75     ),
76     y_(wallDist::New(mesh_).y()),
77     x_(mesh_.C())
78
79 {
80 }
81
82 void shearCurrent::printCoeffs()
83 {
84     Info << "Loading wave theory: " << typeName << endl;
85 }
86
87 // * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //
88
89 scalar shearCurrent::factor(const scalar& time) const
90 {
91     scalar factor(1.0);
92     if (Tsoft_ > 0.0)
93     {
94         factor = Foam::sin(2*PI_/(4.0*Tsoft_)*Foam::min(Tsoft_, time));
95     }
96
```

```
97      return factor;
98  }
99
100 scalar shearCurrent::eta
101 (
102     const point& x,
103     const scalar& time
104 ) const
105 {
106     scalar eta = localSeaLevel_;
107     return eta;
108 }
109
110 scalar shearCurrent::pExcess
111 (
112     const point& x,
113     const scalar& time
114 ) const
115 {
116     return referencePressure(localSeaLevel_);
117 }
118
119 vector shearCurrent::U
120 (
121     const point& x,
122     const scalar& time
123 ) const
124 {
125
126     volScalarField temp=y_;
127     volScalarField temp2=y_;
128
129     temp *=scalar(0.0);
130     temp2 *=scalar(0.0);
131
132     scalar yPlus_ = 0.0;
133     scalar vanDriest_ = 0.0;
134
135     scalar ksPlus_= ks_*Uf_/nu_;
136     scalar deltayPlus_=0.9*(Foam::sqrt(ksPlus_)-ksPlus_*Foam::exp(-ksPlus_/6.0));
137
138     scalar shearU_ = 0.0;
139
140     forAll(temp,index)
141     {
142
143         yPlus_=y_[index]*Uf_/nu_;
144
145         vanDriest_=1.0/(1.0+Foam::sqrt(1.0+4.0*Foam::pow(kappa_,2.0)*Foam::pow(yPlus_+deltayPlus_,2.0)
     *Foam::pow(1.0-Foam::exp(-(yPlus_+deltayPlus_)/Ad_),2.0)));
146
147         temp[index] =vanDriest_;
148         temp2[index] =vanDriest_;
149
150         if ( index > 0)
151
152         {
153             temp[index] = temp[index-1] + 0.5*(temp2[index] + temp2[index-1])*(y_[index]*Uf_/nu_ - y_[
     index-1]*Uf_/nu_);
154         }
155
156         if ( x_[index] == x)
157
158         {
159             shearU_ = 2.0*Uf_*temp[index];
160         }
161
162     }
```

```
163
164     return shearU_*vector(1.0,0,0)*factor(time);
165 }
166 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
167
168 } // End namespace waveTheories
169 } // End namespace Foam
170
171 // ************************************************************************* //
```

B.4 shearCurrentProperties.H and shearCurrentProperties.C files

Listing B.5: shearCurrentProperties.H

```
1  /*---------------------------------------------------------------------------*\
2    =========                 |
3    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4     \\    /   O peration     |
5      \\  /    A nd           | Copyright held by original author
6       \\/     M anipulation  |
7  -------------------------------------------------------------------------------
8  License
9      This file is part of OpenFOAM.
10
11     OpenFOAM is free software; you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by the
13     Free Software Foundation; either version 2 of the License, or (at your
14     option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM; if not, write to the Free Software Foundation,
23     Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
24
25 Class
26     Foam::shearCurrentProperties
27
28 Description
29     This is a dummy file, where nothing happens, as the input parameters given
30     by the user is sufficient.
31
32     A description of the general wave framework is given in
33
34     @article { jacobsenFuhrmanFredsoe2011,
35         Author = {Jacobsen, N G and Fuhrman, D R and Freds\o{}e, J},
36         title = {{A Wave Generation Toolbox for the Open-Source CFD Library: OpenFoam\textregistered
       {}}},
37         Journal = {{Int. J. for Numer. Meth. Fluids}},
38         Year = {2012},
39         Volume = {70},
40         Number = {9},
41         Pages = {1073-1088},
42         DOI = {{10.1002/fld.2726}},
43     }
44
45 SourceFiles
46     shearCurrentProperties.C
47
48 Author
```

```
    Niels Gjoel Jacobsen, Technical University of Denmark.  All rights reserved.


\*---------------------------------------------------------------------------*/

#ifndef shearCurrentProperties_H
#define shearCurrentProperties_H

#include "setWaveProperties.H"

namespace Foam
{

class shearCurrentProperties
:
    public setWaveProperties
{
private:

public:

        //- Runtime type information
        TypeName("shearCurrentProperties");

        shearCurrentProperties
        (
            const Time&,
            dictionary&,
            vector,
            bool
        );

        // Method

        virtual void set(Ostream&);
};

}

#endif
```

Listing B.6: `shearCurrentProperties.C`

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
```

```
24
25 \*---------------------------------------------------------------------------*/
26
27 #include "shearCurrentProperties.H"
28 #include "addToRunTimeSelectionTable.H"
29
30 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
31
32 namespace Foam
33 {
34
35 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
36
37 defineTypeNameAndDebug(shearCurrentProperties, 0);
38 addToRunTimeSelectionTable
39 (
40     setWaveProperties,
41     shearCurrentProperties,
42     setWaveProperties
43 );
44
45 // * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * //
46
47
48 shearCurrentProperties::shearCurrentProperties
49 (
50     const Time& rT,
51     dictionary& dict,
52     vector g,
53     bool write
54 )
55 :
56     setWaveProperties(rT, dict, g, write)
57 {
58     Info << "\nConstructing: " << this->type() << endl;
59 }
60
61
62 // * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //
63
64
65 void shearCurrentProperties::set(Ostream& os)
66 {
67     // Write the beginning of the sub-dictionary
68     writeBeginning( os );
69
70     // Write the already given parameters
71     writeGiven( os, "waveType" );
72     writeGiven( os, "Tsoft" );
73     writeGiven( os, "Uf");
74     writeGiven( os, "nu");
75     writeGiven( os, "ks");
76
77     if (dict_.found( "localSeaLevel" ))
78     {
79         writeGiven( os, "localSeaLevel");
80     }
81     if (dict_.found( "kappa" ))
82     {
83         writeGiven( os, "kappa");
84     }
85     if (dict_.found( "Ad" ))
86     {
87         writeGiven( os, "Ad");
88     }
89
90     // This is where type specific data can be written
91     // Nothing to be done for shearCurrent
```

```
92
93      // Write the relaxation zone
94      writeRelaxationZone( os );
95
96      // Write the closing bracket
97      writeEnding( os );
98  }
99
100
101 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
102
103 } // End namespace Foam
104
105 // ************************************************************************* //
```

Index