Grid2Grid : HOS Wrapper Program

Young-Myung Choi, Maite Gouin, Guillaume Ducrozet,

Benjamin Bouscasse and Pierre Ferrant

26 June, 2019

Version 2.2

Ecole Centrale de Nantes

Release Notes

Version 2.0

New features

- The source code file structure is organized for the recognition of class header and its functionality.
- The source code file structure is organized for the recognization of class header and its functionality.
- HOS results file generated by HDF5 library is added as an input file for Grid2Grid.
- Dictionary type input for Grid2Grid is introduced.
- Input file for `postGrid2Grid` is changed to follow dictionary format.
- Cmake is introduced.

Fixed

- HOS-NWT FFTW3 memory deallocation error (destructor of HOS-NWT class).
- z-directional `surf2vol` grid bug (when `meshGrid` with `meshRatio`$= 1$).
- 3D HOS-Ocean interpolation class initialization error (when n_x and n_y is different).

Version 2.1

- Compiling option with HDF5 library is added when it is not available.

Version 2.2

- CMake options to compile with arguments.
- New reconstruction procedure of HOS field is added(input data from HOS is needed. Functionality in future).

Contents

1 Introduction

Wave generation solvers using Higher Order Spectral Method (HOS) have been validated and developed for several years (Ducrozet et al. (2016), Ducrozet et al. (2007), Bonnefoy et al. (2011) and Ducrozet et al. (2012)). HOS solves nonlinear wave propagation in open-sea (HOS-Ocean) and also in numerical wave tank (HOS-NWT) with low computation time comparing with other nonlinear wave solvers because the theory is based on pseudo-spectral method (HOS (b) and HOS (a)). Those HOS wave solvers are released as open-source codes, which anyone can develop, use and distribute under the terms of GNU General Public Licence (GPLv3).

Nonlinear irregular wave generation in computational fluids dynamic (CFD) solvers becomes important recently in naval fields to better estimate the loads on offshore structure. The conventional linear superposition methods imply a long computational time for wave generation and the simulation is made almost impossible without having enough computational power. And if the method is based on linear wave theory, there is also question on occurrence of nonlinear phenomenon and the interaction between waves as the simulation goes. Therefore dedicated nonlinear wave solvers with high computational speed are needed.

`Grid2Grid` is developed as a wrapper program of HOS to generate wave fields from the results of HOS computation. `Grid2Grid` reconstructs wave fields of HOS with inverse fast Fourier transforms (FFTs) and uses a quick spline module, the nonlinear wave fields can be fastly reconstructed for arbitrary simulation time and space. The nonlinear wave simulation is then possible for a particular position and time where specific non linear phenomenon occur.

`Grid2Grid` compiles a shared library (`libGrid2Grid.so`) which can be used for communication with other programming language using the `ISO_C_BINDING` rule. It compiles also a post processing program of HOS.

2 Program Structure

The file structure of `Grid2Grid` is shown in Fig. 2.1. The directory `config` contains the compiling rule for `makefile` not `cmake`. The directory `doc` constains `Grid2Grid` manual files. The directory `example` contains the examples for interface between other languages and `Grid2Grid`. The directory `src` includes the libraries and source file directory. The library used `Grid2Grid` are `libbspline` and `libFyMc`. The `libbspline` is b-spline interpolation library and `libFyMc` is a general Fortran library. The `libGrid2Grid` is the source code directory of `Grid2Grid` and it is composed of each modules which are sepated in a module directory with its sub-class(type) directories.

```
Grid2Grid
├─ config
├─ doc
├─ example
│  ├─ fortGrid2Grid; Example for the interface between Fortran
│  ├─ ofGrid2Grid; Example 1 for the interface between OpenFOAM
│  └─ ofGrid2GridDict; Example 2 for the interface between OpenFOAM
└─ src
   ├─ libBspline; b-spline module
   ├─ libFyMc; Fortran module for the general purpose
   └─ libGrid2Grid
      ├─ auxiliary
      ├─ fftw3; FFTW3 library interface module
      ├─ HDF5; HDF5 library interface module
      ├─ surf2Vol; HOS wave field reconstruction module
      ├─ vol2Vol; HOS wave field interpolation module
      └─ postGrid2Grid; post Grid2Grid Module
```

Figure 2.1. Grid2Grid file structure

The program structure of Grid2Grid is depicted in Fig. 2.2. The program is composed of several modules (.f90) with a suffix mod. HOS wave fields are generated by modSurf2Vol from the HOS result file (modes_HOS_SWENSE.dat or modes_HOS_SWENSE.hdf5). Since the results files only contains modes information, the volumic wave fields is reconstructed on the HOS grid by inverse FFTs (denoted as H_2 Operator). The volumic wave field in Grid2Grid at the certain time step is called as a snapshot of wave field. modVol2Vol constructs an interpolation data structure from several snapshots of wave field by using multidimensional spline module (Williams (2015)). modVol2Vol can directly compiled with other fortran program if the same Fortran compiler is used (gfortran¿4.9.2). modGrid2Grid generates shared library named as libGrid2Grid.so to communicate with other languages. Standard program language rule ISO_C_BINDING is required to communicate with the library Grid2Grid. The examples are given in directory example. modPostGrid2Grid is a post processing module of Grid2Grid generating the HOS wave fields for the 3D visualization(VTK format) and the wave elevation time series.

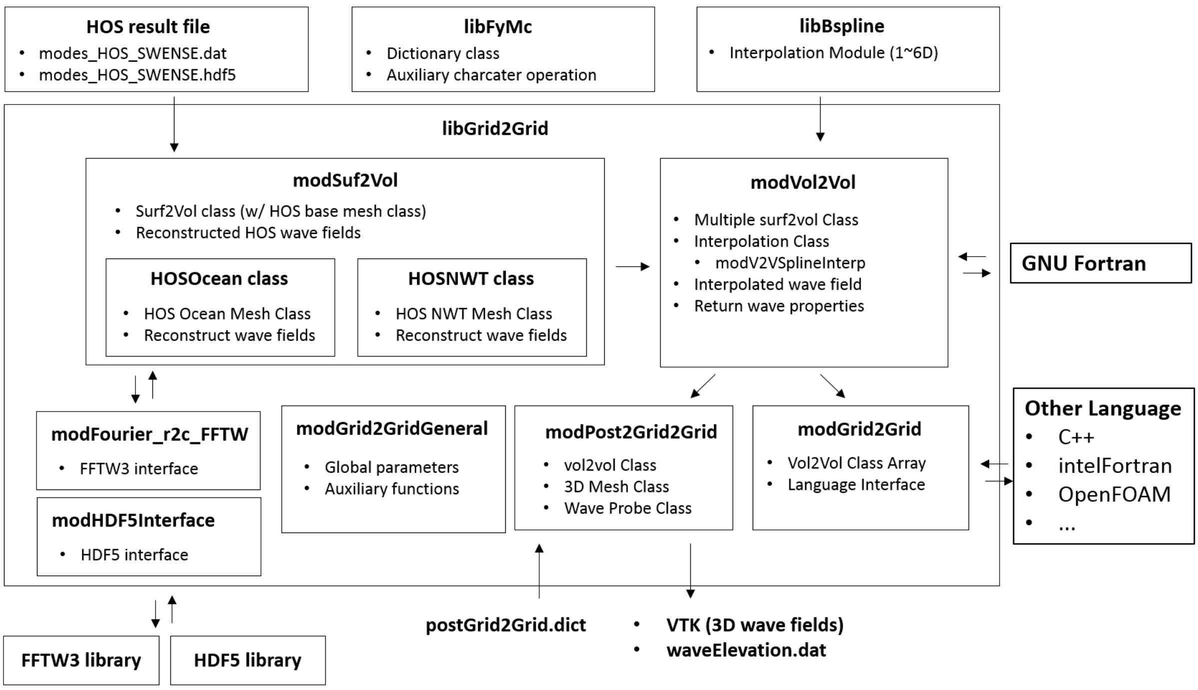


Figure 2.2. Program Structure of Grid2Grid

The main feature of each module is following.

- `modSurf2Vol` : HOS-Ocean and NWT classes and reconstructed HOS wave fields
- `modVol2Vol` : Interpolate reconstructed HOS wave fields
- `modGrid2Grid` : Communication with other language with `ISO_C_BINDING`
- `modPostGrid2Grid` : Post processing of HOS (wave fields and elevation)
- `modFourier_r2c_FFTW` : Interface module of `FFTW` open library
- `modHDF5Interface` : Interface module of `HDF5` open library
- `modGrid2GridGeneral` : Grid2Grid global variables and auxiliary functions
- `libBspline` : Multidimensional b-spline module written in Fortran.
- `libFyMc` : Auxiliary fortran functions (Dictionary, Character operation, ...)

2.1 typSurf2Vol

2.1.1 Description

`Surf2Vol` is a wrapper class to access HOS Ocean and NWT class. HOS-Ocean and NWT classes have the similar class structures but only reconstruction schemes and definition of meshes are slightly different. The reconstructed wave fields by two HOS classes are saved in each HOS mesh class which is derived class from base HOS mesh class. By using this mesh pointer, `Surf2Vol` class can access both HOS reconstructed wave fields. It is reminded that the direct access to the HOS wave fields is possible on HOS simulation time and grid by only HOS mesh pointer to prevent misuse of HOS wave data.

`Surf2Vol` class structure is depicted in Fig. 2.3. `Surf2Vol` class contains `HOSOceanSurf2Vol` and `HOSNWTSurf2Vol` classes. When the function is called, it calls the sub-class function distiguished by HOS type.

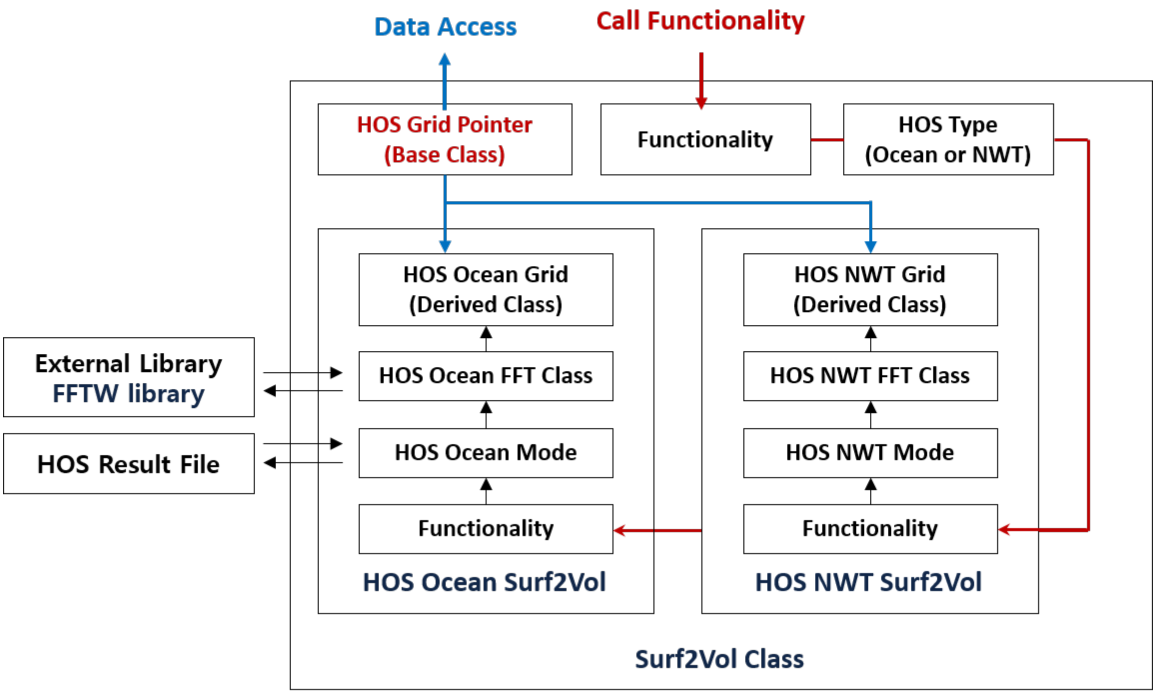


Figure 2.3. `Surf2Vol` class structure

`hosNWTSurf2Vol` and `hosOceanSurf2Vol` classes reconstruct the HOS wave fields from the result file of HOS by using the H_2 operator (inverse FFTs). Wave reconstruction and HOS wave theory is well explained in Ducrozet et al. (2007) and Ducrozet et al. (2012). Each class reconstruct the wave fields on the derived HOS mesh class.

HOS wave theory is based on the superposition of modes satisfying the Laplace equation, sea bottom and free surface boundary condition basically. The vertical profile of the mode function is usually given as the exponential or hyperbolic functions. The vertical profile could magnify the local properties when the number of modes is increased and the wave steepness is signigicantly high. Those behaviors ususally happens above the $z = 0$. It is resulted from the amplitudes reduction is not as much decreasing with the number of modes. Therefore those mathematical formulation gives the unnatural values. To prevent those things, the criterion to cut the local mode contribution above the $z = 0$ is introduced as (2.1).

$$f_{mn}(z) = \begin{cases} \dfrac{\cosh k_{mn}(z+H)}{\cosh k_{mn} H} & \text{if} \quad f_{mn}(z) < C_{f(z)} \\ C_{f(z)} & \text{if} \quad f_{mn}(z) \geq C_{f(z)} \end{cases} \tag{2.1}$$

where

k_{mn} : HOS mode wave number $\quad z$: HOS z-coordinates $\quad H$: water depth

$C_{f(z)}$: $f_{mn}(z)$ function criterion value (default = 10)

The parameter $C_{f(z)}$ is set to be 10 in `Grid2Grid` as a defualt value. If the local wave velocities generated by `Grid2Grid` are not sufficient, the parameter can be changed. The parameter is defined in `modGrid2GridGeneral.f90` as a `FNZ_VALUE`.

HOS result file only contains the mode amplitudes computed at each HOS simulation time. The volumic grid should be generated and the wave fields are reconstructed on this grid. The generated volumic grid is used as a background grid for the interpolation. To constuct the HOS grid, following information should be given when `surf2vol` class is initialised.

- `zMin`, `zMax`
- `nZMin`, `nZMax`
- `nZMinRatio`, `nZMaxRatio` [Optional]

¡¡¡¡¡¡¡ HEAD where `zMin` and `zMax` is used to set z-directional domain and it should have negative and positive value. `nZMin` and `nZMax` are the number of z-directional grid. It is recommended to have at least 50-100 points for the interpolation. If a sufficient number of grid points is not given, the interpolation scheme could give strange values due to exponetial profile of $f_{mn}(z)$. `nZMinRatio` and `nZMaxRatio` are the ratio of maximum and minumum height of grid ($\Delta z_{max}/\Delta z_{min}$). Minimum grid is located at $z = 0$. Those are optional values set to be 3 as default. The grid can be visualized by using ParaView. The VTK file is located at VTK/Grid2Grid/surf2volMesh.vtk. ======= where `zMin` and `zMax` is used to set z-directional domain length and it should have the negative and positive value. `nZMin` and `nZMax` are the number of z-directional grid. It is recommended to have at least 50-100 for the sufficient interpolation. If the number of grid points is not sufficient, the interpolated values could be strange due to the vertical profile of $f_{mn}(z)$. `nZMinRatio` and `nZMaxRatio` are the ratio of maximum and minumum height of grid ($\Delta z_{max}/\Delta z_{min}$). Minimum grid is located at $z = 0$. Those are optional values set to be 3 as default. The generated HOS grid is automatically saved whe `surf2vol` class is initialised. And it is visualized by using ParaView. The VTK file is located at VTK/Grid2Grid/surf2volMesh.vtk. ¿¿¿¿¿¿¿ master

2.1.2 Class (Type)

Class : `Surf2Vol`

- Data :
  - ◦ `hosNWTSurf2Vol_` : HOS-NWT Surf2Vol Class
  - ◦ `hosOceanSurf2Vol_` : HOS-Ocean Surf2Vol Class
  - ◦ `ptrHOSMesh_` : HOS Mesh Pointer

- Functionality :
  - ◦ `initialize` : Initialise HOS Surf2Vol class
  - ◦ `correct` : Update HOS wave fields
  - ◦ `destroy` : Class desroyer

Class : `HOSOcean` and `HOSNWT`

- Data :
  - ◦ `HOSfile` : HOS result file (`modes_HOS_SWENSE.dat`) or `modes_HOS_SWENSE.hdf5`
  - ◦ `HOSmode` : HOS Ocean or NWT modes
  - ◦ `HOSmesh` : HOS Ocean or NWT mesh
  - ◦ `HOSfftw` : HOS FFT class for H_2 operator

- Public Functionality :
  - ◦ `initialize` : Initialise HOS Ocean or NWT Surf2Vol
  - ◦ `correct` : Update HOS Ocean or NWT wave fields
  - ◦ `destroy` : Class desroyer

- Private Functionality :
  - ◦ `init_read_mod` : Read HOS number of modes and allocate dynamic array
  - ◦ `read_mod` : Read HOS modes at given HOS time index
  - ◦ `buildGlobalMesh` : Build HOS mesh with the given HOS construction parameter
  - ◦ `reconstuctionFFTs` : Reconstruct the wave fields by H_2 operator

2.1.3 How to use

– Initialise `Surf2Vol`

```
Call hosS2V%initialize(hosType, filePath, zMin, zMax, nZmin, nZmax,
   zMinRatio, zMaxRatio)

! hosS2V        : Surf2vol Class (Type)
! hosType       : HOS Type (Ocean or NWT)
! filePath      : HOS result file path (modes_HOS_SWENS.dat)
! zMin, zMax    : HOS grid z-minimum and z-maximul (vertical domain)
! nZmin, nZmax  : Number of z-directional Grid
!
! zMinRatio, zMaxRatio (Optional)
! : Ratio of maximum and minimum height of grid (default=3)

! or

Call hosS2V%initialize(dict)

! hosS2V      : Surf2vol Class (Type)
! dict        : HOS dictionary to initialize HOS surf2vol (Type)
```

– Correct `Surf2Vol`

```
Call hosS2V%correct(hosIndex)

! hosS2V        : Surf2vol Class (Type)
! hosIndex      : HOS time index
```

– Data access on wave field of `Surf2Vol`

◦ Wave elevation

```
eta = hosS2V%ptrHOSMesh_%eta(ix, iy)

! hosS2V        : Surf2vol class (Type)
! ix, iy        : HOS grid index (x, y)
! eta           : Wave elevation
```

◦ Wave velocity

```
u = hosS2V%ptrHOSMesh_%eta(ix, iy, iz)
v = hosS2V%ptrHOSMesh_%eta(ix, iy, iz)
w = hosS2V%ptrHOSMesh_%eta(ix, iy, iz)
```

```
! hosS2V         : Surf2vol class (Type)
! ix, iy, iz     : HOS grid index (x, y, z)
! u, v, w        : Wave velocity (x, y, z)
```

- ◦ Dynamic pressure ($p_d = p - \rho g z$)

```
pd = hosS2V%ptrHOSMesh_%pd(ix, iy, iz)

! hosS2V         : Surf2Vol class (Type)
! ix, iy, iz     : HOS grid index (x, y, z)
! pd             : Dynamic pressure
```

– Destruct of `Surf2Vol`

```
Call hosS2V%destroy()

! hosS2V         : Surf2Vol class (Type)
```

2.2 typVol2Vol

2.2.1 Description

¡¡¡¡¡¡¡ HEAD `Vol2Vol` interpolates the wave fields and return wave properties at given time and position. It has the multiple `Surf2Vol` classes and interpolation class. The `Vol2Vol` class structure is described in Fig 2.4. The reconstructed HOS wave fields (snapshot of wave field) retrieved from `Surf2Vol` are used to construct interpolation data.

HOS result file holds modes amplitudes time series for the whole HOS simulation time. If we construct HOS wave fields and interpolation data structure for the whole HOS simulation time, not only the computation time is long but also a huge memory is demanded. `Grid2Grid` aims for construction of demanding wave fields for relatively short period and domain, it is not necessary to construct the whole HOS wave fields and interpolation data structure. Therefore revolving algorithm is applied just to update HOS wave fields adjucent to simulation time and constuct small interpolation data structure for efficient computation time and memory. ======= `Vol2Vol` interpolates the wave fields and return wave properties at given time and position. It has the multiple `Surf2Vol` classes and interpolation class. The `Vol2Vol` class structure is described in Fig 2.4. The reconstructed HOS wave fields (snapshot of wave field) retrived from `Surf2Vol` are used to construct interpolation data.

The HOS result file contains the whole simulation time information. If the HOS wave fields and interpolation is applied for the whole HOS simulatio time, the pre-computation time becomes too long and it needs hugh computation memory. `Grid2Grid` is designed for the efficient wave-field reconstruction demanded by CFD solvers for a relatively short period and small domain. Therefore it is not necessary to reconstruct whole HOS wave fields. For the efficient memory control, the revolving algorithm is introduced to holds multiple HOS wave fields and to update HOS wave field if it is demanded. ¿¿¿¿¿¿¿ master

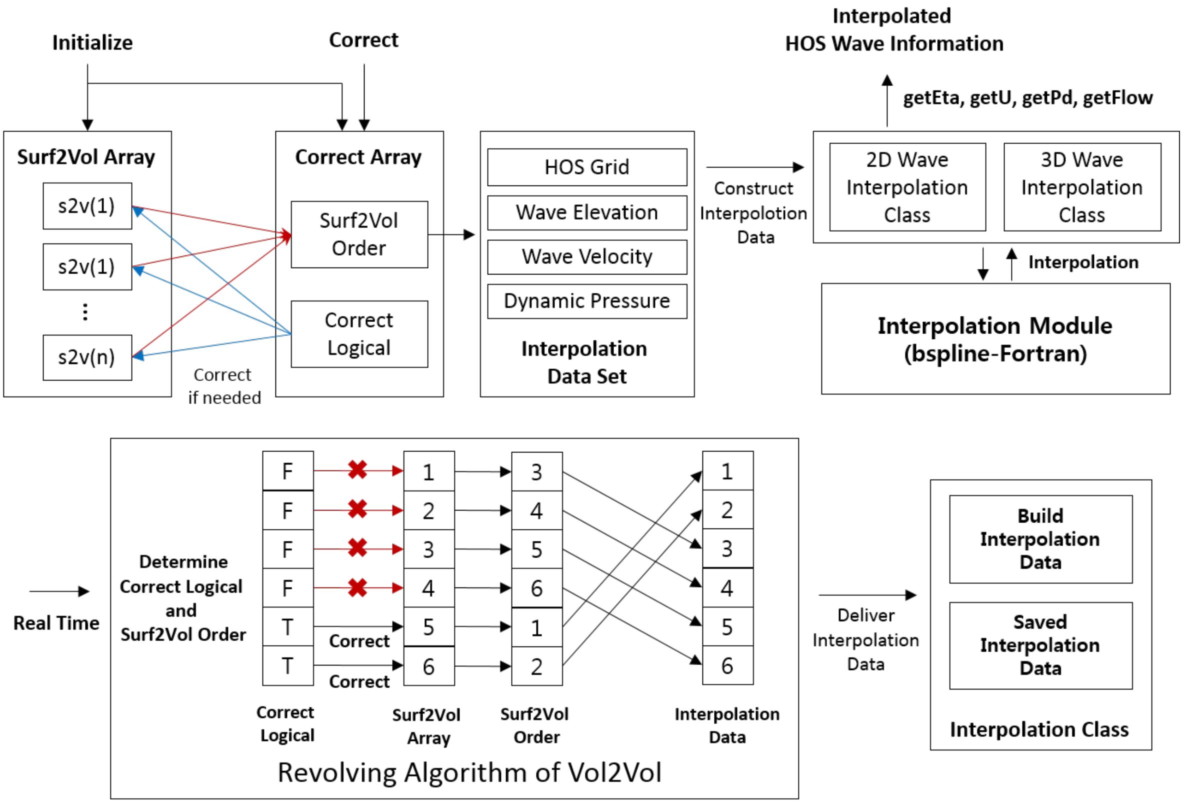


Figure 2.4. `Vol2Vol` class structure

¡¡¡¡¡¡¡ HEAD When Vol2Vol initialised, the Surf2Vol class array is allocated and initialise multiple Surf2Vol and connect those Surf2Vol class array with the revolving algorithm. The revolving algorithm determines which Surf2Vol should be updated and the order of Surf2Vol array for the efficient generation of interpolation data.

The subroutine correct with input t first determine HOS Surf2Vol correction index and Surf2Vol order based on input t and previous HOS time index. By using correction index and order, only necessary HOS Surf2Vol is updated and re-ordered to constructed for interpolation data. After interpolation data is constructed, it is delivered to the interpolation class. The interpolation class communicates with the bspline-Fortran module and constructs the interpolation data structure. ======= When Vol2Vol initialised, the Surf2Vol class array is allocated and initialise multiple Surf2Vol and connect those Surf2Vol class array with the revolving algorithm. The revolving algorithm determine which Surf2Vol should be updated and the order of Surf2Vol array for the efficient generation of interpolation data.

The subroutine correct with an input t first determine HOS Surf2Vol correction indexand Surf2Vol order by revolving algorithm. Only necessary HOS Surf2Vol is updated and re-ordered to constructed for interpolation data. The interpolation data is constructed from the multiple wave field snapshot. The interpolation class holds the intepolation classes specific to HOS and it communicates with the bspline-Fortran module. ¿¿¿¿¿¿¿ master

The subroutine getEta, getU, getPd and getFlow return the interpolated wave properties for the given time and space.

2.2.2 Class (Type)

Class : Vol2Vol

– Data :

- nInterp_ : Interpolation order (2 : Linear, 3 : Quadratic, 4 : Cubic, ...)
- nSaveT_ : Number of Surf2Vol wave fields (nSaveT_ > nInterp_)
- HOSs2v_(:) : Array of Surf2Vol class
- itp2D_ : Interpolation class for 2D waves
- itp3D_ : Interpolation class for 3D waves

– Functionality :

- initialize : Initialise HOS Vol2Vol class with HOS type, result file path, ...
- correct : Update HOS wave fields with real-time

- `getEta` : Get interpolated wave elevation
- `getU` : Get interpolated wave velocity
- `getPd` : Get interpolated dynamic pressure
- `getFlow` : Get flow information
- `destroy` : Class destroyer

2.2.3 How to use

– Initialise `Vol2Vol`

```
Call hosV2V%initialize(hosType, filePath, zMin, zMax, nZmin, nZmax,
 zMinRatio, zMaxRatio, iflag)

! hosV2V          : Vol2vol Class (Type)
! hosType         : HOS Type (Ocean or NWT)
! filePath        : HOS result file path (modes_HOS_SWENS.dat)
! zMin, zMax      : HOS grid z-minimum and z-maximul (vertical domain)
! nZmin, nZmax    : Number of z-directional Grid
!
! zMinRatio, zMaxRatio  (Optional)
! : Ratio of maximum and minimum height of grid (default=3)
!
! iflag : Wriging option (iflag = 1, write Grid2Grid information)

! or

Call hosV2V%initialize(dict)

! hosV2V        :  Vol2vol Class (Type)
! dict          : HOS dictionary to initialize HOS vol2vol (Type)
```

– Correct `Vol2Vol`

```
Call hosV2V%correct(simulTime)

! hosV2V          : Vol2vol Class (Type)
! simulTime       : Simulation time (real time value)
```

– Get wave elevation from `Vol2Vol`

```
eta = hosV2V%getEta(x, y, simulTime, iflag)

! hosV2V          : Vol2vol Class (Type)
! x, y            : x and y position
! simulTime       : Simulation time (real time value)
!
! eta             : Wave elevation
!
! iflag (Optional)
! : if iflag = 1, nondimensional x and y can be given (default = 0).
```

– Get wave velocity from `Vol2Vol`

```
Call hosV2V%getU(x, y, z, simulTime, u, v, w, iflag)

! hosV2V          : Vol2vol Class (Type)
! x, y, z         : x, y and z position
! simulTime       : Simulation time (real time value)
!
! u, v, z         : Wave velocity ( x, y, z )
!
! iflag (Optional)
! : if iflag = 1, nondimensional x and y can be given (default = 0).
```

– Get dynamic pressure from `Vol2Vol`

```
pd = hosV2V%getPd(x, y, z, simulTime, iflag)

! hosV2V          : Vol2vol Class (Type)
! x, y, z         : x, y and z position
! simulTime       : Simulation time (real time value)
!
! pd              : Dynamic velocity (pd = p - rho * g * z)
!
! iflag (Optional)
! : if iflag = 1, nondimensional x and y can be given (default = 0).
```

– Get flow information from `Vol2Vol`

```
Call hosV2V%getFlow(x, y, z, simulTime, eta, u, v, w, pd, iflag)

! hosV2V          : Vol2vol Class (Type)
! x, y, z         : x, y and z position
! simulTime       : Simulation time (real time value)
!
! eta             : Wave elevation
! u, v, z         : Wave velocity ( x, y, z )
! pd              : Dynamic velocity (pd = p - rho * g * z)
!
! iflag (Optional)
! : if iflag = 1, nondimensional x and y can be given (default = 0).
```

– Destroy `Vol2Vol`

```
Call hosV2V%destroy()
```

2.3 modGrid2Grid

2.3.1 Description

`modGrid2Grid` is not a class but the module which is desinged for the communication between other programming languages. The module structure is depicted in Fig. 2.5. It has a data base (DB) which has the input data information and `Vol2Vol`. The input data is saved in data base and dertermines whether new `Vol2Vol` is initialized. When the ohter program first initialise `Grid2Grid`, `modGrid2Grid` firstly checks the input variables. If the same input is previously given, it return previous `Vol2Vol` index. But it does not exist, it allocates new `Vol2Vol` and issues `Vol2Vol` index. Other programming language distingush the different HOS wave theory in a same simulation by `HOSIndex` (pointer in `ISO_C_BINDING` is complicate). Therefore the subroutines `correct` and `get*` should be called with `HOSIndex`, position and time.

The array size of `modGrid2Grid` is set to be 100 as a default (100 different HOS waves) but it does not consume much of memory because the classes of `Grid2Grid` are programmed with the dynamic arrays and consequently only a few static data are used. The variable `nMaxVol2Vol` in `src/libGrid2Grid/modGrid2Grid.f90` can be changed.

The functionalities of `modGrid2Grid` are similar with `Vol2Vol`.

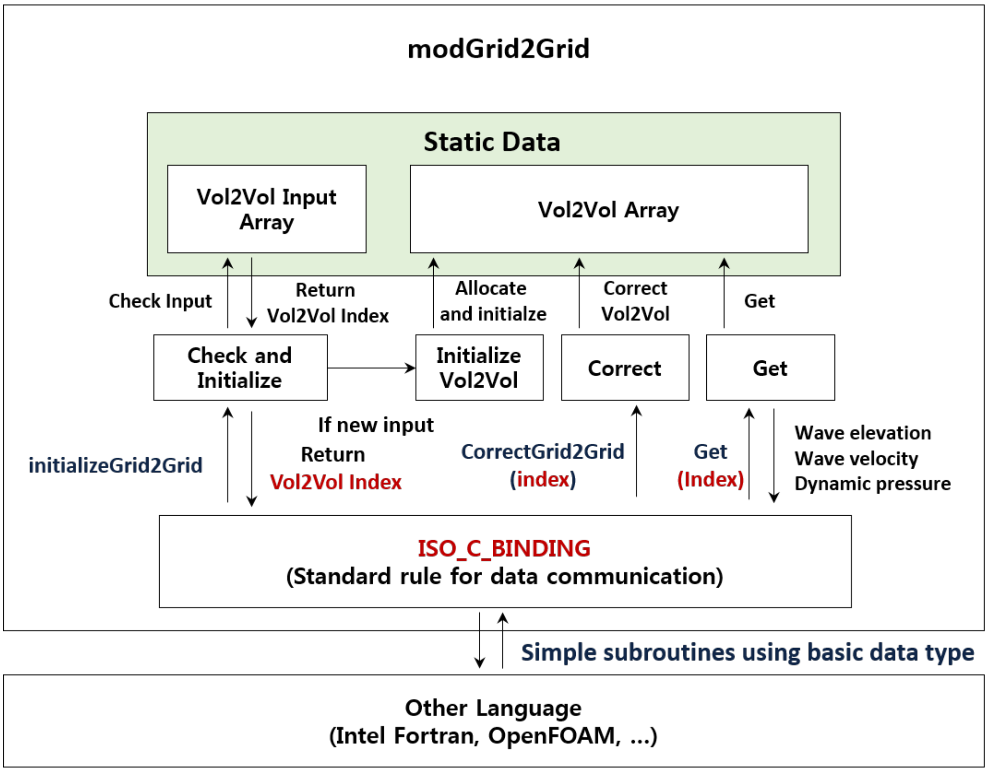


Figure 2.5. `Grid2Grid` Module structure

2.3.2 Subroutines

The interface between other languages with `modGrid2Grid` will be described in Chap. 4.

Subroutines :

– Initialise Grid2Grid
  - `__modgrid2grid_MOD_initializegrid2grid`
  - `__modgrid2grid_MOD_initializegrid2griddict`

– Correct Grid2Grid
  - `__modgrid2grid_MOD_correctgrid2grid`

– Get wave elevation
  - `__modgrid2grid_MOD_gethoseta`

– Get wave velocity
  - `__modgrid2grid_MOD_gethosu`

– Get dynamic pressure
  - `__modgrid2grid_MOD_gethospd`

– Get flow information
  - `__modgrid2grid_MOD_gethosflow`

– Get HOS simulation end time
  - `__modgrid2grid_MOD_gethosendtime`

– Get HOS water depth
  - `__modgrid2grid_MOD_gethoswaterdepth`

– Get logical data indicating HOS wave theory is initialized
  - `__modgrid2grid_MOD_isgrid2gridinitialized`

2.4 PostGrid2Grid

2.4.1 Description

¡¡¡¡¡¡¡ HEAD `PostGrid2Grid` is a HOS post-processing class. It generates 3D VTK files of wave fields for visualization and wave elevation time series computed from `Vol2Vol` class. Wave fields at desired simulation time and spatial domain and wave elevation time series can be re-generated at some provided wave probes position.

`PostGrid2Grid` algorithm is depicted in Fig. 2.6. `PostGrid2Grid` is initialised with input file. The input file `postGrid2Grid.dict` contains HOS grid information and post processing information which have dictionary format. `PostGrid2Grid` first reads and checks the input file and then build 3D visualization grid and wave probes. `Vol2Vol` class is also initialised. The subroutine `doPostProcessing` do time loop of `correct`. Subroutine `correct` first corrects the `Vol2Vol` class and gets the wave fields. If the grid option is set to no air mesh, 3D grid is fitted to wave elevation. It writes the results on files (3D VTK file and wave elevation time series).

======= `PostGrid2Grid` is a HOS post-processing class. It generates 3D VTK files of wave fields for the visualization and wave elevation time series computed from `Vol2Vol` class. `PostGrid2Grid` algorithm is depicted in Fig. 2.6. `PostGrid2Grid` is initialised with the input file. The input file (`postGrid2Grid.dict`) contains the parameters needed for the post processing and HOS-reconstruction. It is composed with many dictionary input (HOS, simulation, vtkMesh and waveProbe dictionaries). `PostGrid2Grid` reads and checks the input file and then build 3D visualization grid and wave probes with given dictionaries. `Vol2Vol` class is also initialised with HOS dictionary. The subroutine `doPostProcessing` run a time simulation of reconstruction. The subroutine `correct` updates the `Vol2Vol` class and gets the wave fields. If the grid option `airMesh` is equal to `false`, 3D grid is fitted to the wave elevation.

¿¿¿¿¿¿¿ master

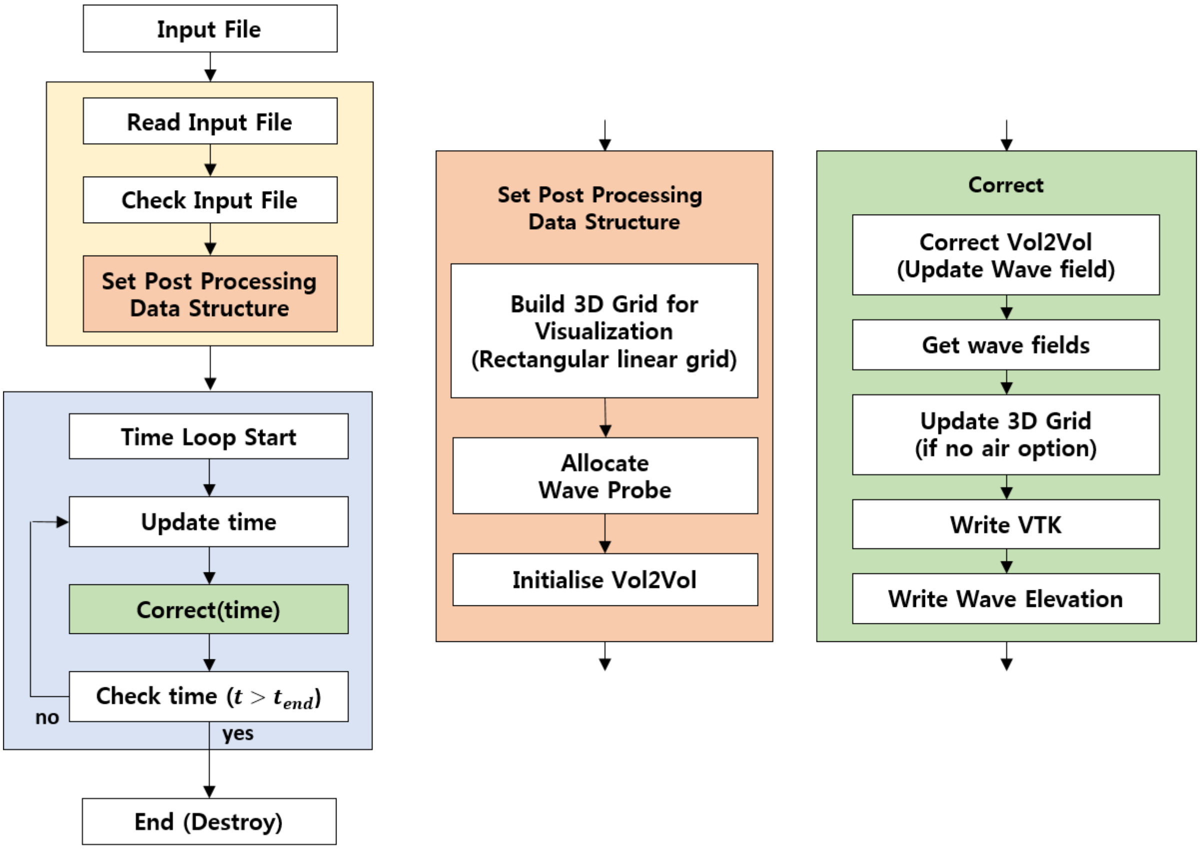


Figure 2.6. `PostGrid2Grid` Algorithm

2.4.2 Class(Type)

Class : `PostGrid2Grid`

- Data :
  - `hosVol2Vol_` : `Vol2Vol` class
  - `rectLGrid_` : Rectangular linear grid for 3D wave fields (VTK output)
  - `waveProbe_(:)` : Wave probe

- Functionality (Public) :
  - `initialize` : Initialise PostGrid2Grid class
  - `correct` : Correct `Vol2Vol`, `rectLGrid_` and `waveProbe_` and write output
  - `writeVTK` : Write 3D wave fields in VTK format
  - `doPostProcessing` : Do post processing
  - `destroy` : Destuctor of PostGrid2Grid class

- Functionality (Private) :
  - `readPostG2GInputFileDict` : Read `PostGrid2Grid` input file
  - `checkPostG2GParameter` : Check `PostGrid2Grid` input file
  - `writeVTKtotalASCII` : Write wave fields (Including air domain)
  - `writeVTKnoAirASCII` : Write wave fields (Grid is fitted to wave elevation)
  - `writeWaveProbe` : Write wave elevation time series

2.4.3 Input File of PostGrid2Grid

`PostGrid2Grid` needs the input file. The input file name is `postGrid2Grid.dict`. The input is recognized by `dictionary(keyword + value with sub-dictionary)`. The special characters are treated as a comment (`!`, `#` and \\) and C++ style comment block is also allowed.

– PostGrid2Grid variables

```
<<<<<<< HEAD

  ### Select HOS dictionary and set post processing option
   ----------------------------------- #

  Grid2Grid          HOSDict;  # HOS dictionary

=======

  ### Select HOS dictionary and set post processing option
   ----------------------------------- #

  Grid2Grid          HOSDict;  # HOS dictionary

>>>>>>> master
  writeVTK           true; # 3D visualization option; true or false

  writeWaveProbe     true; # Wave probe option; true or false
```

– HOS dictionary

```
  ### HOS dictionary ----------------------------------- #

  HOSDict
  {
    type   Ocean;    # HOS solver type; NWT or Ocean

    filePath  example/modes_HOS_SWENSE.dat; # HOS file path

    fileType  ASCII;  # Option; HOS result file type, ASCII or HDF5

    interpolationOrder  3;  # Option; 3:Cubic spline [Default]

    zMin    -0.6;    # HOS domain zMin
    zMax    0.6;     # HOS domain zMax

    nZMin   50;   # HOS domain nZMin
    nZMax   50;   # HOS domain nZMax
```

```
  zMeshType meshRatio;    # Mesh type (uniform, sine, meshRatio)

  zMinRatio 3.0;  # Option, z<=0 meshing ratio (meshRatio)
  zMaxRatio 3.0;  # Option, z<=0 meshing ratio (meshRatio)

  writeLog        true; # Option; write log option
}
```

– Simulation Dictionary

```
### Simulation dictionary -------------------------------------------
 #

simulation
{
  startTime   2712.0;

  endTime     2812.0;

  dt          0.1;
}
```

– 3D visualization dictionary

```
### VTK mesh dictionary ------------------------------ #

vtkMesh
{
  airMesh   false;    # Air meshing option

  xMin      0.0;     # VTK mesh xMin
  xMax    100.0;     # VTK mesh xMax

  yMin      0.0;     # VTK mesh yMin
  yMax    100.0;     # VTK mesh yMax

  nX      50;        # VTK mesh nX
  nY      50;        # VTK mesh nX

  zMin    -0.6;      # VTK mesh zMin
  zMax    0.6;       # VTK mesh zMax

  nZMin   50;        # VTK mesh nZMin
  nZMax   50;        # VTK mesh nZMax

  zMeshType meshRatio;    # Mesh type (uniform, sine, meshRatio)
```

```
    zMinRatio 3.0;  # Option, z<=0 meshing ratio (meshRatio)
    zMaxRatio 3.0;  # Option, z<=0 meshing ratio (meshRatio)

}
```

– Wave probe dictionary

```
### Wave probe dictionary ------------------------------------------ #

waveProbe
{
  waveProbeFile waveElevation.dat;    # Output file name

  waveProbes
  {
    probe1
    {
      position  ( 0.0   10.0 );   # wave probe position
    }
    probe2
    {
      position  ( 0.0   15.0 );
    }
  }
}
```

2.4.4 How to use

```
!! Subroutine to run PostGrid2Grid -----------------------------------
Subroutine runPostGrid2Grid(inputFileName)
!! -------------------------------------------------------------------
Use modPostGrid2Grid          !! Use PostGrid2Grid Module
!! -------------------------------------------------------------------
Implicit none
Character(Len = * ),intent(in) :: inputFileName   !! postGrid2Grid.inp
Type(typPostGrid2Grid)         :: postG2G       !! postGrid2Grid Class
!! -------------------------------------------------------------------

!! Initialize PostGrid2Grid with Input File
Call postG2G%initialize(inputFileName)

!! Do Post Processing
Call postG2G%doPostProcessing()

!! Destroy
Call postG2G%destroy

!! -------------------------------------------------------------------
End Subroutine
!! -------------------------------------------------------------------
```

3 Installation

3.1 Pre-Install

3.1.1 FFTW Install

`Grid2Grid` needs a fast Fourier transform (FFT) library. `FFTW` is GNU licenced FFT library. The installation order is the following:

1. Download FFTW library

At the FFTW website (http://www.fftw.org/download.html), the latest version of FFTW is available. Download `FFTW` library.

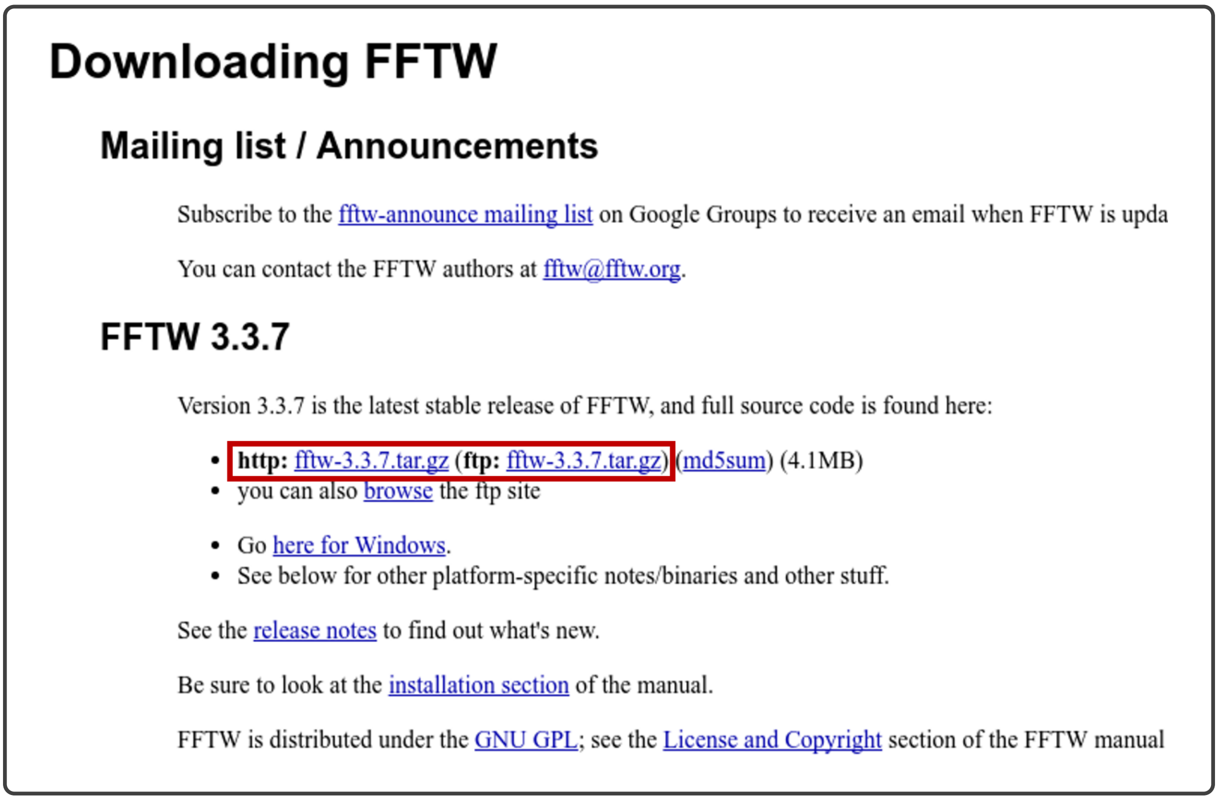


Figure 3.1. Download `FFTW library`

2. Extract FFTW library

```
$ tar -xvzf fftw-3.3.7.tar.gz
```

3. Compile FFTW library

```
$ cd fftw-3.3.7/
$
$ export FFTW_PATH=$PWD
$
$ ./configure --prefix=$FFTW_PATH
$
$ make CFLAGS='-fPIC'
$
$ make install
```

4. Make soft link of FFTW library

If user has a super user authority

```
$ sudo ln -s $FFTW_PATH/lib/libfftw3.a  /usr/local/lib/libfftw3.a
```

¡¡¡¡¡¡¡ HEAD If user has no super user authority, FFTW library path should be changed manually. If FFTW library (`libfftw3.a`) locates at `/home/user/lib/libfftw3.a`, the FFTW3 library path should be changed as follows: ======= If user has no super user authority, FFTW library path should be changed manually. If FFTW library (`libfftw3.a`) locates at `/home/user/lib/libfftw3.a`, the FFTW3 library path should be changed as following : ¿¿¿¿¿¿¿ master

- Compile with makefile

Change the library path in `config/config.mk`

```
FFTW_LIB=/home/user/lib/
```

- Compile with cmake

Change the library path in `CMakeLists.txt`

```
set(FFTW3_LIB_PATH /home/user/lib)
```

3.1.2 HDF5 Install

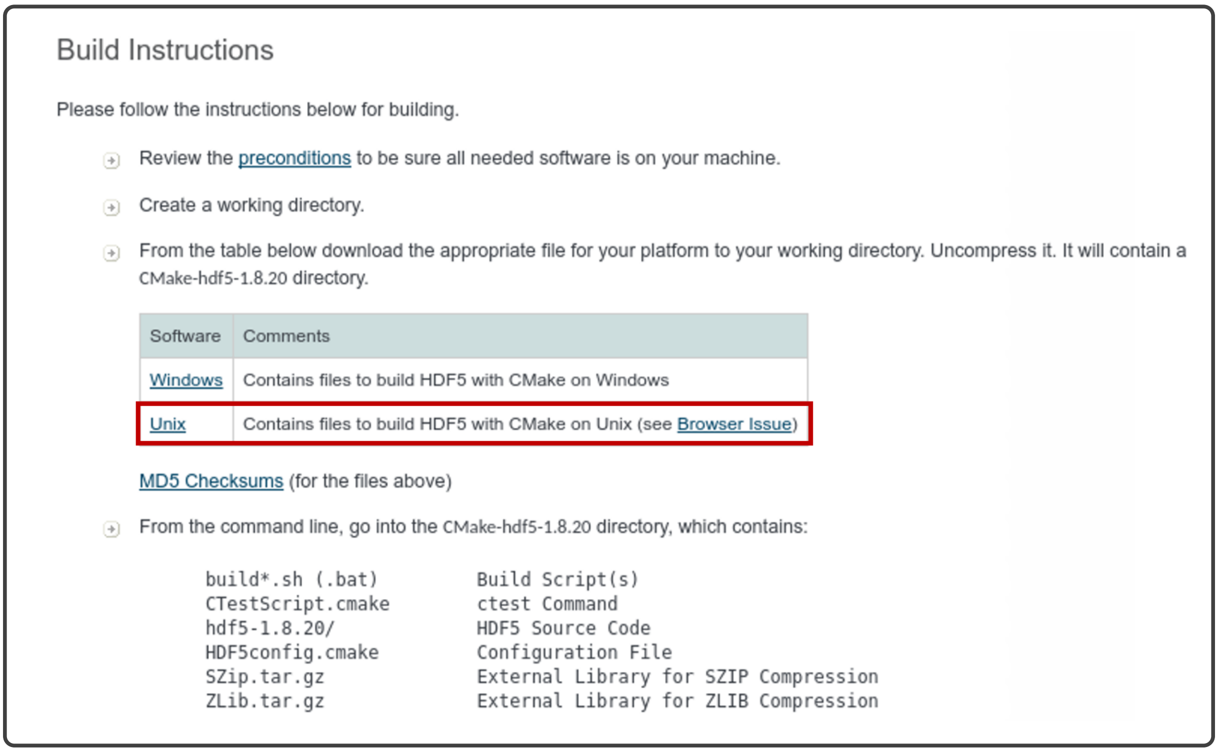
¡¡¡¡¡¡¡ HEAD In version 2.0, HDF5 library can be used as a option. Because the result file of HOS for a 3 hours simulation is huge, the file size can be reduced thanks to use of the HDF5 library. In our experience, the size is reduced around 60% when the result file is generated in a HDF5 format.

When compiling `Grid2Grid`, the default installation setting with HDF library is `disbled`. The compilation with HDF library will be explained in section 3.2; The installation procedures for the HDF5 library is given below. ======= In version 2.0, HDF5 library can be used as a option. Because the result file of HOS for 3 hours simulation is huge, the file size can be reduced thanks to HDF5 library. In a experience, the size is reduced around 60% when the result file is generated in a HDF5 format.

When compiling `Grid2Grid`, the default installation setting with HDF library is `disbled`. The compilation with HDF library will be explaind in section 3.2; The installation procedures for the HDF5 library is given in below. ¿¿¿¿¿¿¿ master

1. Download HDF5 library

At the HDF5 website (https://support.hdfgroup.org/HDF5/release/cmakebuild518.html), the source code of HDF5 with cmake is available (26 March 2018)



Build Instructions

Please follow the instructions below for building.

- Review the preconditions to be sure all needed software is on your machine.
- Create a working directory.
- From the table below download the appropriate file for your platform to your working directory. Uncompress it. It will contain a CMake-hdf5-1.8.20 directory.

Software	Comments
Windows	Contains files to build HDF5 with CMake on Windows
Unix	Contains files to build HDF5 with CMake on Unix (see Browser Issue)

MD5 Checksums (for the files above)

- From the command line, go into the CMake-hdf5-1.8.20 directory, which contains:

```
build*.sh (.bat)        Build Script(s)
CTestScript.cmake       ctest Command
hdf5-1.8.20/            HDF5 Source Code
HDF5config.cmake        Configuration File
SZip.tar.gz             External Library for SZIP Compression
ZLib.tar.gz             External Library for ZLIB Compression
```

Figure 3.2. Download `HDF5` source code with Cmake

2. Unzip the compressed folder

```
$ tar -xvzf CMake-hdf5-1.10.1.tar.gz
```

3. Add compiling rules in HDF5options.cmake

¡¡¡¡¡¡ HEAD Open the file `HDF5options.cmake` inside of extracted folder and add compiling rules as follows: ======= Open the file `HDF5options.cmake` inside of extracted folder and add compiling rules as following : ¿¿¿¿¿¿¿ master

```
### disable packaging

#set(ADD_BUILD_OPTIONS "${ADD_BUILD_OPTIONS} -DHDF5_NO_PACKAGES:BOOL=ON
   ")
### Create install package with external libraries (szip, zlib)
set(ADD_BUILD_OPTIONS "${ADD_BUILD_OPTIONS} -DHDF5_PACKAGE_EXTLIBS:BOOL=
   ON")
set(ADD_BUILD_OPTIONS "${ADD_BUILD_OPTIONS} -DHDF5_BUILD_FORTRAN:BOOL=ON
   ")
set(ADD_BUILD_OPTIONS "${ADD_BUILD_OPTIONS} -DBUILD_SHARED_LIBS:BOOL=ON"
   )
set(ADD_BUILD_OPTIONS "${ADD_BUILD_OPTIONS} -DHDF5_BUILD_CPP_LIB:BOOL=ON
   ")
#set(ADD_BUILD_OPTIONS "${ADD_BUILD_OPTIONS} -DHDF5_ENABLE_THREADSAFE:
   BOOL=ON")
#set(ADD_BUILD_OPTIONS "${ADD_BUILD_OPTIONS} -DHDF5_BUILD_JAVA:BOOL=ON")

set(CMAKE_Fortran_FLAGS "-fPIC")
```

4. Compile the source code with Cmake

```
 $ ctest -S HDF5config.cmake,BUILD_GENERATOR=Unix -C Release -V -O
hdf5.log
```

5. Check the compiled libraries exist in sub-directory build/bin

```
 $ ls build/bin/libhdf5.a build/bin/libhdf5_fortran.a build/bin/
libszip.a build/bin/libz.a
```

6. Make soft link of HDF5 or connect HDF5 library with Grid2Grid

If the user has a super user authority, and want to add HDF5 library on user library path :

```
export HDF5_PATH=/home/CMake-hdf5-x

$ ln -s $HDF5_PATH /usr/local/lib/hdf5
```

If the user have no super user authority, or do not want to add HDF5 library on user library path :

- Compile with makefile

Change the library path in `config/config.mk`

```
HDF5_LIB=/home/CMake-hdf5-x/build/bin
```

- Compile with cmake

Change the library path in `CMakeLists.txt`

```
set(HDF5_LIB_PATH /home/CMake-hdf5-x/build/bin)
```

3.1.3 HOS Ocean and NWT (Optional)

`Grid2Grid` needs the result file of HOS solver. Installation of HOS is the following:

1. Install FFTW and HDF5

See Chapters 3.1.1 and 3.1.2

2. Install makedepf90

```
$ sudo apt-get install makedepf90
```

3. Install LAPACK

```
$ sudo apt-get install liblapack-dev liblapack-doc-man liblapack-doc
 liblapack-pic liblapack3 liblapack-test liblapack3gf liblapacke
liblapacke-dev
```

4. Download HOS Ocean and NWT

```
$ # Path to desired installation path
$
$ cd $HOS_INSTALLATION_PATH
$
$ git clone https://github.com/LHEEA/HOS-ocean.git
$
$ git clone https://github.com/LHEEA/HOS-NWT.git
```

5. Change the library path defined in makefile

Open `makefile` of HOS Ocean and NWT and change as follows:

```
HDF5_INCLUDE=/usr/local/lib/hdf5/build/bin/static/
HDF5_LIB=/usr/local/lib/hdf5/build/bin/

#LINKLIB = $(LIBDIR)libfftw3.a $(LIBDIR)liblapack.a $(LIBDIR)
librefblas.a

LINKLIB = $(LIBDIR)libfftw3.a -llapack
```

```
 LINKLIB+= $(HDF5_LIB)libhdf5_fortran.a $(HDF5_LIB)libhdf5_f90cstub.a
 $(HDF5_LIB)libhdf5.a $(HDF5_LIB)libszip.a $(HDF5_LIB)libz.a -ldl -
pthread
```

The setting of FFTW3 and HDF5 library is depicted in the previous chapters

6. Comple HOS NWT and HOS Ocean

```
$ cd HOS-ocean/
$
$ make
$
$ cd ../HOS-NWT/
$
$ make
$
$ cd ..
```

7. Check executable is generated

```
$ # Check HOS NWT
$ ls HOS-NWT/bin/HOS-NWT
$
$ # Check HOS Ocean
$ ls HOS-ocean/bin/HOS-ocean
```

8. Make soft link (optional)

```
$ # Make Soft Link (Optional)
$
$ export HOS_PATH=$PWD
$
$ sudo ln -s $HOS_PATH/bin/HOS-NWT    /usr/bin/HOS-NWT
$
$ sudo ln -s $HOS_PATH/bin/HOS-ocean  /usr/bin/HOS-ocean
```

3.2 Grid2Grid Installation

3.2.1 Download Grid2Grid

Download Grid2Grid

```
$ # Path to desired installation path
$
$ cd $HOS_INSTALLATION_PATH
$
$ git clone https://github.com/LHEEA/Grid2Grid.git
```

Set FFTW and HDF5 library path

See chapters 3.1.1 and 3.1.2

3.2.2 Compile PostGrid2Grid with makefile

Compiling with `makefile` without HDF5 library is not realised. To unable HDF5, please install with `cmake`

- Compile with makefile

```
$ cd Grid2Grid
$
$ make create
```

3.2.3 Compile Grid2Grid Shared Library

- Compile `libGrid2Grid.so` in `Grid2Grid/lib/`

```
$ make createlib
```

- Compile `libGrid2Grid.so` in `$FOAM_USER_LIBBIN`

If OpenFOAM is installed, `libGrid2Grid.so` can be compiled at `$FOAM_USER_LIBBIN`. If OpenFOAM environment is called, following make rule can be used directly.

```
$ make createOFlib
```

3.2.4 Compile with cmake

- Compile `libGrid2Grid.so` in `lib/` and `postG2G` in `Grid2Grid`

```
$ cd Grid2Grid
$
$ cmake -H. -Bbuild
$
$ cmake --build build
```

- Compile with HDF5 library

```
$ cd Grid2Grid
$
$ cmake -H. -Bbuild -DHDF_LIBRARY:STRING="ON"
$
$ cmake --build build
```

- Compile `libGrid2Grid.so` in `$FOAM_USER_LIBBIN` and `postG2G` in `Grid2Grid`

If OpenFOAM is installed, `libGrid2Grid.so` can be compiled at `$FOAM_USER_LIBBIN`.

```
$ cd Grid2Grid
$
$ cmake -H. -BbuildOF -DBUILD_OF_LIB=ON
$
$ cmake --build buildOF
```

4 Interface

4.1 GNU and Intel Fortran

An interface example fortran program is included in `interface/fortGrid2Grid`. To communicate with `Grid2Grid`, the fortran script `modCommG2G.f90` in `interface/fortGrid2Grid` is needed. It is a communication module with `libGrid2Grid.so`. The fortran interface example program is the following :

```
!! Program Start ---------------------------------------------------
Program Main
!! ----------------------------------------------------------------
use modCommG2G       !! Use Communication Module
!! ----------------------------------------------------------------
Implicit None
!! Variables ------------------------------------------------------
Integer,Parameter       :: nChar = 300     !! Default Character Length
Character(len = nChar) :: grid2gridPath   !! libGrid2Grid.so Path

integer                 :: hosIndex        !! HOS Index
Character(len = nChar) :: hosSolver       !! HOS Solver (Ocean or NWT)
Character(len = nChar) :: hosFileName     !! HOS Result File Path

Double precision        :: zMin, zMax      !! Surf2Vol Domain
integer                 :: nZmin, nZmax    !! Number of vertical grid
Double precision        :: zMinRatio, zMaxRatio  !! Grading ratio (=3)

Double precision        :: t, dt      !! Simulation Time, dt
Double precision        :: x, y, z    !! Computation Point
Double precision        :: eta, u, v, w, pd  !! HOS Wave Information
!! Dummy variables ------------------------------------------------
integer                 :: it         !! Dummy time loop integer

!! Program Body ---------------------------------------------------

!!!... Write Program Start
write(*,*) "Test program (Connect to Fortran) to use Grid2Grid shared
 library"

!!!... Set libGrid2Grid.so path.
!!!    It is recommended to use absolute path
! grid2gridPath = "/usr/lib/libGrid2Grid.so"  (if soft link is made)
grid2gridPath = "../../obj/libGrid2Grid.so"

!!!... Load libGrid2Grid.so and connect subroutines
Call callGrid2Grid(grid2gridPath)

!!!... Declare HOS Index
hosIndex = -1
```

```fortran
!!!... Set HOS Type (Ocean or NWT)
hosSolver = "NWT"

!!!... Set HOS Result file Path
hosFileName = "modes_HOS_SWENSE.dat"

!!!... Set HOS Surf2Vol Domain and Vertical Grid
zMin = -0.6d0;         zMax =  0.6d0
nZmin = 50;            nZmax = 50
zMinRatio = 3.d0;      zMaxRatio = 3.d0

!!... Initialize Grid2Grid and Get HOS Index
Call initializeGrid2Grid(hosSolver, hosFileName, zMin, zMax, nZmin,
 nZmax, zMinRatio, zMaxRatio, hosIndex)

!! Time Information
t  = 0.0d0;     dt = 0.1d0

!! Given Point
x = 0.5d0;      y = 0.5d0;      z = -0.5d0

!! Time Loop
do it = 1,10

  !! Correct HOS Vol2VOl for given time
  Call correctGrid2Grid(hosIndex, t)

  !! Get Wave Elevation
  Call getHOSeta(hosIndex, x, y , t, eta)

  !! Get Flow Velocity
  Call getHOSU(hosIndex, x, y, z, t, u, v ,w)

  !! Get Dynamic Pressure
  Call getHOSPd(hosIndex, x, y, z, t, pd)

  !! Write Flow Information
  write(*,*) t, eta, u, v, w, pd

  !! Time Update
  t = t + dt
enddo

!! Write End of Program
write(*,*) "Test program (Connect to Fortran) is done ..."
!! -----------------------------------------------------------------
End Program
!! -----------------------------------------------------------------
```

4.2 OpenFOAM

An interface example OpenFOAM program is included in `interface/ofGrid2Grid`. The shared library `libGrid2Grid.so` should be compiled at `$FOAM_USER_LIBBIN`. To check `libGrid2Grid.so` exists at `$FOAM_USER_LIBBIN`, use following shell command :

```
$ ls $FOAM_USER_LIBBIN/libGrid2Grid.so
```

If `libGrid2Grid.so` not exists, refer to Chapter 3.2.

To call shared library `libGrid2Grid.so` in `$FOAM_USER_LIBBIN`, OpenFOAM compiling option is added at `Make/option`. Open `Make/option` and add following compiling option.

```
EXE_LIBS = \
          ...
          -lgfortran \
          -L$(FOAM_USER_LIBBIN) \
          -lGrid2Grid
```

OpenFOAM interface example program is given next page.

```
#include "fvCFD.H"

namespace Foam
{
  //- Grid2Grid Initial Character Length
  const int nCharGridGrid(300);

  //- Initialize Grid2Grid Class in Fortran
  //
  //  __modgrid2grid_MOD_initializegrid2grid
  //  (
  //      hosSolver,
  //      hosFileName,
  //      zMin,
  //      zMax,
  //      nZmin,
  //      nZmax,
  //      zMinRatio,
  //      zMaxRatio,
  //      hosIndex
  //  )
  //
  //    Input
  //      hosSolver             : "NWT" or "Ocean"
  //      hosFileName           : filePath of HOS mode result file
  //      zMin, zMax            : HOS grid zMin and zMax
  //      nZmin, nZmax          : HOS number of z grid
  //      zMinRatio, zMaxRatio : HOS z grid max/min ratio
  //
  //    Output
  //      hosIndex              : HOS Vol2Vol Index
  //
  extern "C" void __modgrid2grid_MOD_initializegrid2grid
  (
  const char[nCharGridGrid],
  const char[nCharGridGrid],
  const double*,
  const double*,
  const int*,
  const int*,
  const double*,
  const double*,
  int*
  );

  //- Correct Grid2Grid for given simulation Time
  //
  //  __modgrid2grid_MOD_correctgrid2grid(hosIndex, simulTime)
  //
  //    Input
```

```
//      hosIndex   : HOS Vol2Vol Index
//      simulTime  : Simulation Time
//
extern "C" void __modgrid2grid_MOD_correctgrid2grid
(
const int *,
const double *
);

//- Get HOS Wave Elevation
//
//  __modgrid2grid_MOD_gethoseta(hosIndex, x, y, t, eta)
//
//    Input
//      hosIndex : HOS Vol2Vol Index
//      x, y, t  : (x and y) position and simulation Time (t)
//
//    Output
//      eta      : wave elevation
//
extern "C" void __modgrid2grid_MOD_gethoseta
(
const int *,
const double *,
const double *,
const double *,
double *
);

//- Get HOS Flow Velocity
//
//  __modgrid2grid_MOD_gethosu(hosIndex, x, y, z, t, u, v, w)
//
//    Input
//      hosIndex    : HOS Vol2Vol Index
//      x, y, z, t  : (x, y, z) position and simulation Time (t)
//
//    Output
//      u, v, w     : (x, y, z) - directional flow velocity
//
extern "C" void __modgrid2grid_MOD_gethosu
(
const int *,
const double *,
const double *,
const double *,
const double *,
double *,
double *,
double *
);
```

```
  //- Get HOS Dynamic Pressure
  //
  //   __modgrid2grid_MOD_gethospd(hosIndex, x, y, z, t, pd)
  //
  //     Input
  //       hosIndex     : HOS Vol2Vol Index
  //       x, y, z, t   : (x, y, z) position and simulation Time (t)
  //
  //     Output
  //       pd       : Dynamic Pressure p = -rho*d(phi)/dt-0.5*rho*|U*U|
  //
  extern "C" void __modgrid2grid_MOD_gethospd
  (
  const int *,
  const double *,
  const double *,
  const double * ,
  const double *,
  double *
  );

  //- Get HOS Wave Elevation, Flow Velocity and Dynamic Pressure
  //
  //   __modgrid2grid_MOD_gethosflow(hosIndex, x, y, z, t, eta, u, v, w,
   pd)
  //
  //     Input
  //       hosIndex     : HOS Vol2Vol Index
  //       x, y, z, t   : (x, y, z) position and simulation Time (t)
  //
  //     Output
  //       eta          : wave elevation
  //       u, v, w      : (x, y, z) - directional flow velocity
  //       pd           : Dynamic Pressure p = -rho * d(phi)/dt - 0.5 *
   rho * |U * U|
  //
  extern "C" void __modgrid2grid_MOD_gethosflow
  (
  const int *,
  const double *,
  const double *,
  const double * ,
  const double *,
  double *,
  double *,
  double * ,
  double *,
  double *
  );

}
```

```
// Main OpenFOAM Program Start
int main(int argc, char *argv[])
{

  // Write Program Start
  Info << "OpenFOAM Program Example to Call Grid2Grid (HOS Wrapper) in
    OpenFOAM" << endl;

  // Set HOS Solver Type
  const word HOSsolver_("NWT");
  const word HOSFileName_("./modes_HOS_SWENSE.dat");

  // Set File Name
  string strHOSSolver = string(HOSsolver_);
  string strHOSFileName = string(HOSFileName_);

  // Set HOS Solver Type
  const char *HOSsolver = strHOSSolver.c_str();

  // Set HOS Mode Result File Path
  const char *HOSfileName = strHOSFileName.c_str();

  // Set HOS Z Grid Information
  int indexHOS(-1);

  double zMin(-0.6), zMax(0.6);
  int nZmin(50), nZMax(50);

  double zMinRatio(3.0), zMaxRatio(3.0);

  // Initialize Grid2Grid
  __modgrid2grid_MOD_initializegrid2grid(HOSsolver, HOSfileName,
  &zMin, &zMax,
  &nZmin, &nZMax,
  &zMinRatio, &zMaxRatio, &indexHOS);

  Info << "HOS Label : " << indexHOS << endl;

  // Set Position
  double x(0.5), y(0.0), z(-0.5);

  // Define Flow Quantities
  double eta, u, v, w, pd;

  // Set Simulation Time and Time Difference
  double simulTime(0.0);
  double dt(0.1);
```

```
  // Time Loop
  for (int it = 0; it < 11; it++)
  {
    // Correct Grid2Grid
    __modgrid2grid_MOD_correctgrid2grid(&indexHOS, &simulTime);

    // Get Wave Eta
    __modgrid2grid_MOD_gethoseta(&indexHOS, &x, &y, &simulTime, &eta);

    // Get Flow Velocity
    __modgrid2grid_MOD_gethosu(&indexHOS, &x, &y, &z, &simulTime, &u, &v
   , &w);

    // Get Dynamic Pressure
    __modgrid2grid_MOD_gethospd(&indexHOS, &x, &y, &z, &simulTime, &pd);

    Info << " sumulTime : " << simulTime << endl;
    Info << "   eta      : " << eta << endl;
    Info << "   u, v, w : " << u << " " << v << " " << w << endl;
    Info << "   pd       : " << pd << nl << endl;

    // Get whole Information
    __modgrid2grid_MOD_gethosflow(&indexHOS, &x, &y, &z, &simulTime, &
   eta, &u, &v, &w, &pd);
    Info << "   eta      : " << eta << endl;
    Info << "   u, v, w : " << u << " " << v << " " << w << endl;
    Info << "   pd       : " << pd << nl << endl;

    // Time Update
    simulTime+=dt;
  }

  return 0;
}
```

5 Validation

6 Validation

`foamStar` is used to validate `Grid2Grid`. `foamStar` is developed by Bureau Veritas and shared with Ecole Centrale de Nantes and it can simulate nonlinear waves, seakeeping problem and also hydro-elasticity problem. It solves multiphase problem with Reynolds Averaged Navier-Stokes equations (RANS) with Volume of fraction (VOF). It is based on standard multiphase solver in OpenFOAM (`interDymFoam`). To generate waves, `foamStar` uses explicit blending scheme which blends computed flow values to target values with weight function. The blending function is given as equation (6.1). Some details of `foamStar` are explained in Seng (2012) and Monroy et al. (2017).

$$\mathrm{U} = (1-w)\mathrm{U} + w\mathrm{U}^{target} \tag{6.1a}$$

$$\alpha = (1-w)\alpha + w\alpha^{target} \tag{6.1b}$$

where

$$w = 1-(1-w_{base})^{\chi} \qquad \chi = \frac{\Delta \mathrm{U} \Delta t}{\Delta x} \qquad w_{base} = -2\xi^3 + 3\xi^2$$

By using `Grid2Grid`, the target values at blending zone can be replaced by the wave components computed from HOS wave theory. For the validation, considered HOS simulation condition is given in Table 6.1.

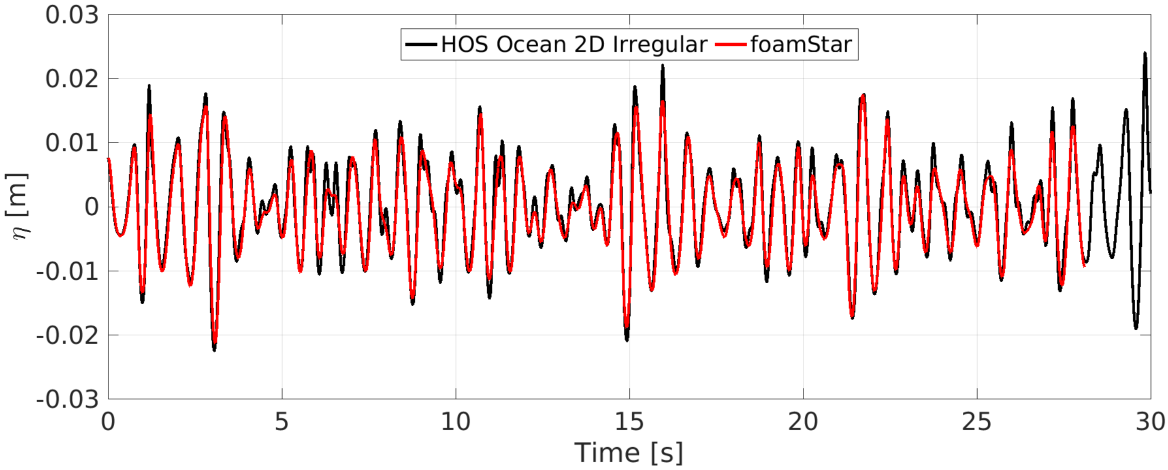
Table 6.1. HOS Wave condition for validation

Wave Type	Value	HOS-Ocean 2D	HOS-Ocean 3D	HOS-NWT 2D	HOS-NWT 3D
Regular Wave	T [s]	-	-	0.702	0.702
	H [m]	-	-	0.0431	0.0288
Irregular Waves	T_p [s]	0.702	1.0	1.0	0.702
	H_s[m]	0.0288	0.10	0.05	0.0384
	γ [-]	3.3	3.3	3.3	3.3
Extreme Waves	T_p [s]	-	-	17.5	-
	H_s[m]	-	-	15.5	-
	γ [-]	-	-	3.3	-

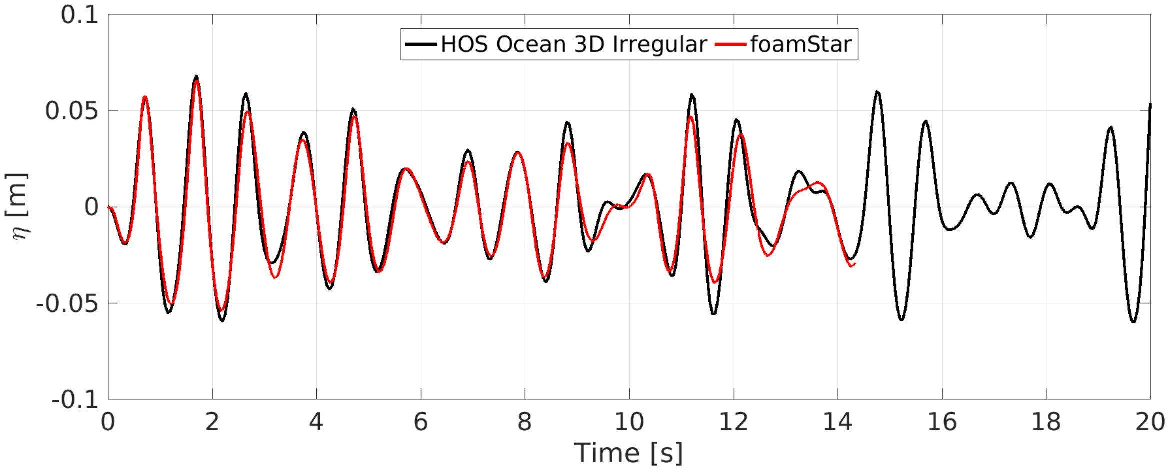
6.1 Simulation results

The wave elevation computed by using `foamStar` and HOS-Ocean are compared in Fig. 6.1. Small differences are observed but it is assumed to be caused by the resolution of the finite volume mesh which is not sufficient near the free surface. The VOF solver gives then those differences. Snapshots of HOS-Ocean wave fields are shown in Fig. 6.1.

The results of HOS-NWT for a two dimensional case is shown in Fig. 6.3. And a simulation snapshot is shown in Fig. 6.4. Three dimensional waves with HOS-NWT are also given in Figs. 6.5 and 6.6.



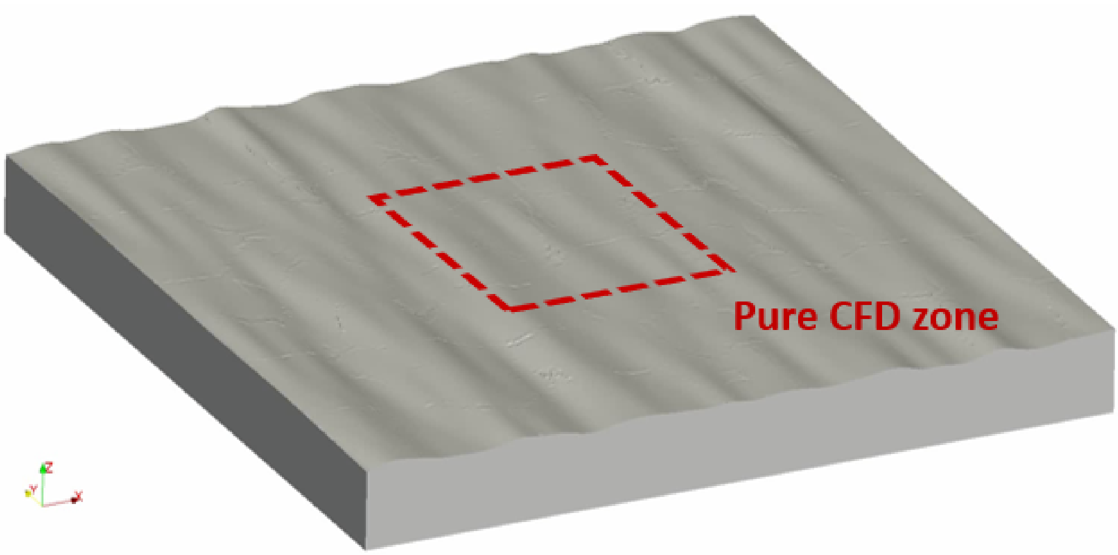
(a) HOS-Ocean 2D Irregular Waves



(b) HOS-Ocean 3D Irregular Waves

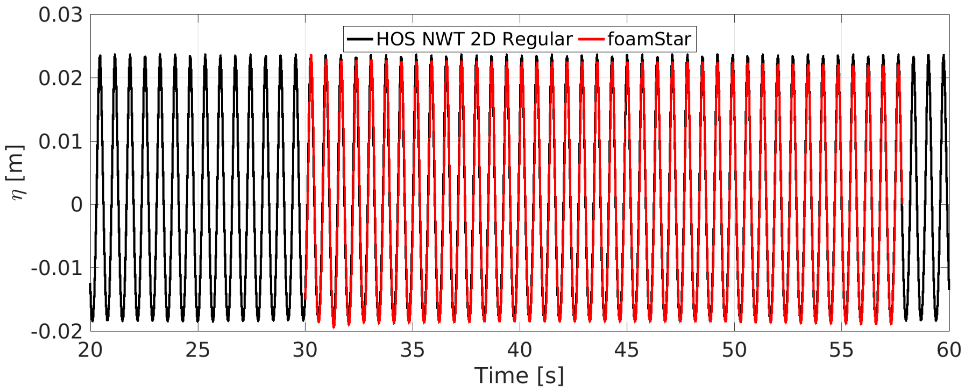
Figure 6.1. Comparison of HOS-Ocean wave elevation

(a) HOS-Ocean 2D Irregular Waves

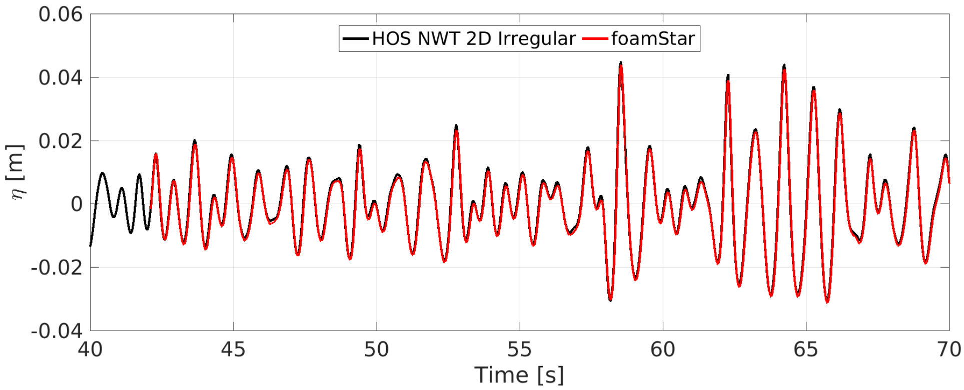


(b) HOS-Ocean 3D Irregular Waves

Figure 6.2. Snapshot of HOS-Ocean wave fields by `foamStar`



(a) HOS-NWT 2D Regular Waves



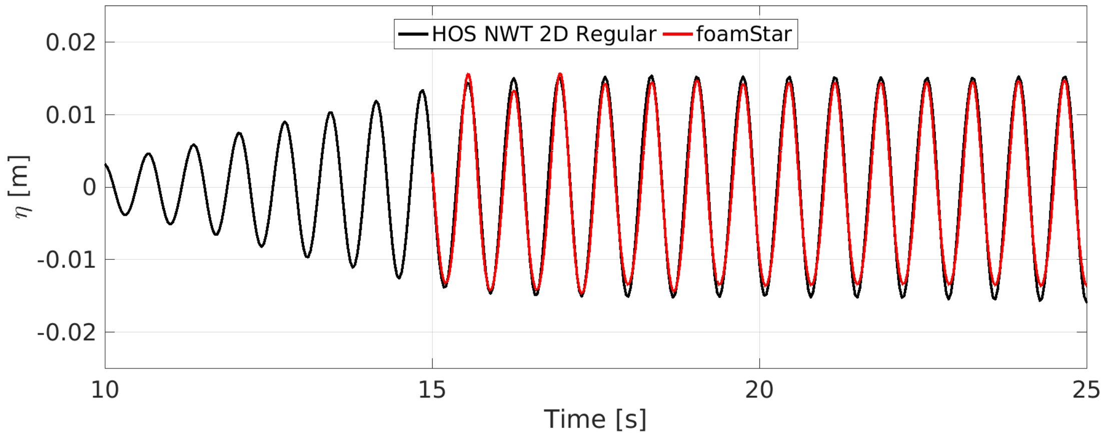
(b) HOS-NWT 2D Irregular Waves

Figure 6.3. Comparison of HOS-NWT 2D wave elevation

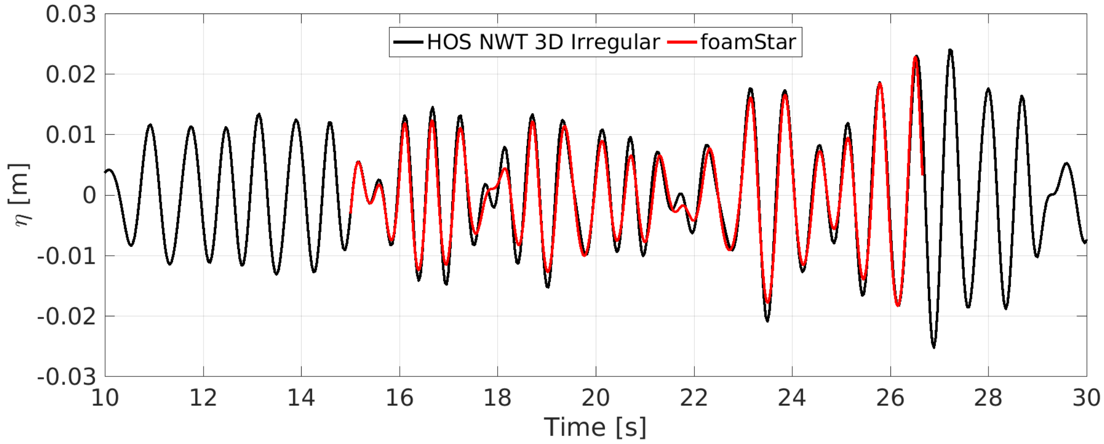
(a) HOS-NWT 2D Irregular Waves

(b) HOS-NWT 2D Irregular Waves

Figure 6.4. Snapshot of HOS-NWT 2D wave fields by `foamStar`

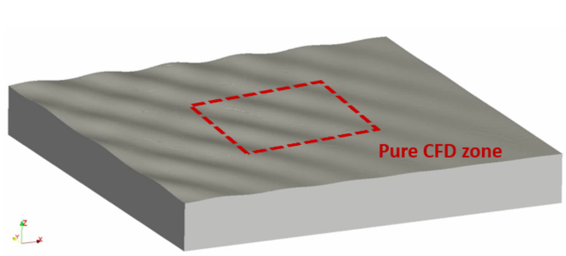


(a) HOS-NWT 3D Regular Waves

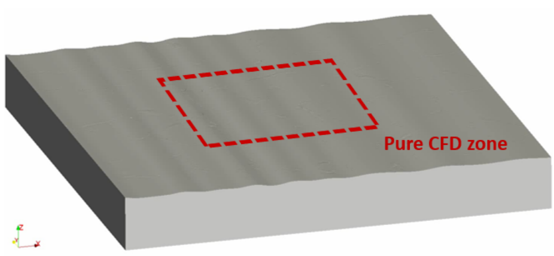


(b) HOS-NWT 3D Irregular Waves

Figure 6.5. Comparison of HOS-NWT 3D wave elevation



(a) HOS-NWT 3D Regular Waves



(b) HOS-NWT 3D Irregular Waves

Figure 6.6. Snapshot of HOS-NWT 3D wave fields by `foamStar`

Simulation of extreme waves (1000 year return period waves in Gulf of Mexico(GOM) (Condition : $H_s = 17.5m, T_p = 15.5s, \gamma = 3.3$) is simulated with HOS-NWT for 2D cases and used to generate waves in `foamStar`. The waves are compared with experiments performed in the wave basin of Ecole Centrale de Nantes (ECN). To simulate nonlinear breaking waves, the wave breaking model in HOS is utilized and allows also to capture when wave breaking occur. The expected wave breaking events are shown in Fig. 6.7. In the experiments, wave breaking is observed at the expected position and time by HOS wave theory. The breaking moment in the experiment is shown in Fig. 6.8. The time series of wave elevation measured in the experiments are compared with the results of simulation using `Grid2Grid` in Fig 6.9. The simulation snapshot when wave breaking occur is shown in Fig. 6.10. The small disturbance at the wave front is observed.

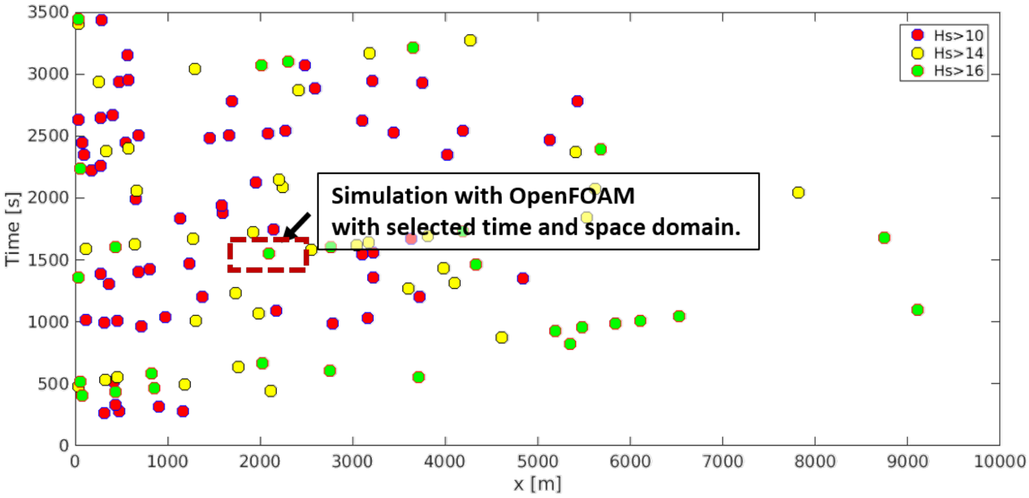


Figure 6.7. Expected wave breaking by HOS

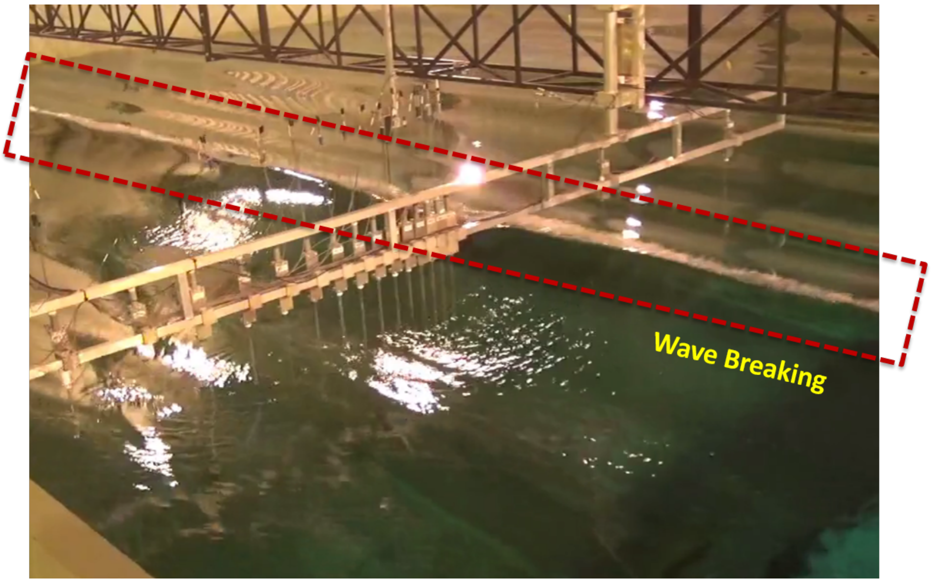
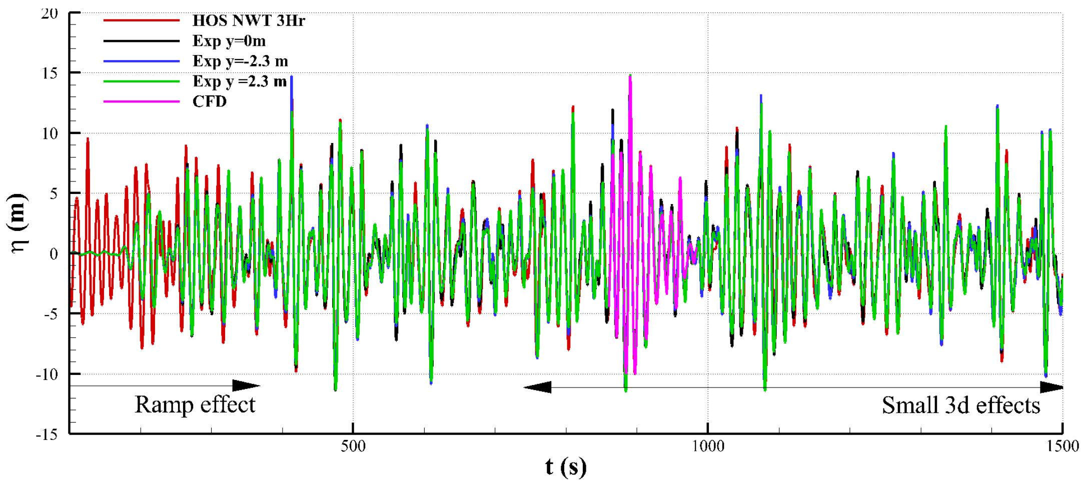
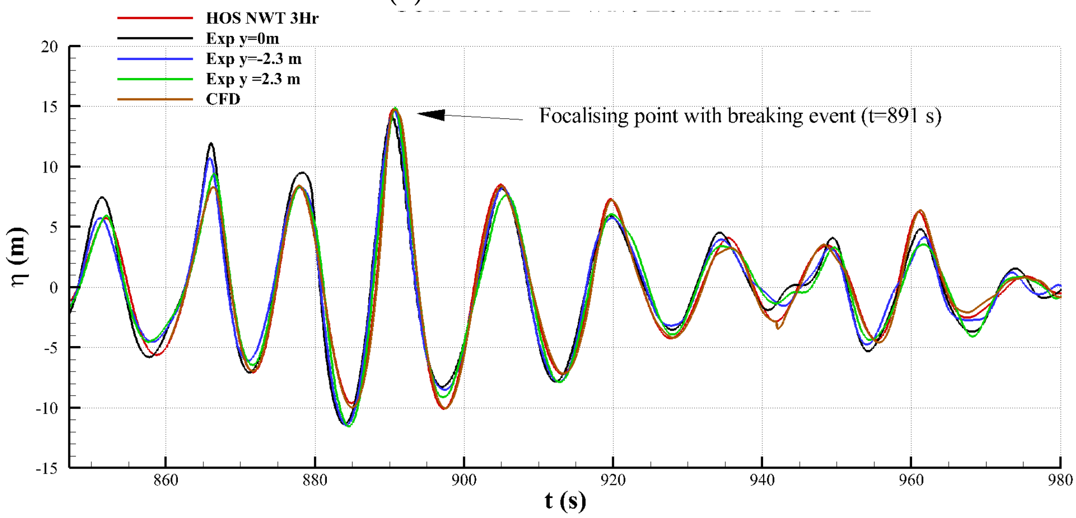


Figure 6.8. Observed wave breaking in experiment



(a) Overall wave time series



(b) Wave elevation at the wave breaking moment

Figure 6.9. Snapshot of HOS-NWT 3D wave fields by `foamStar`

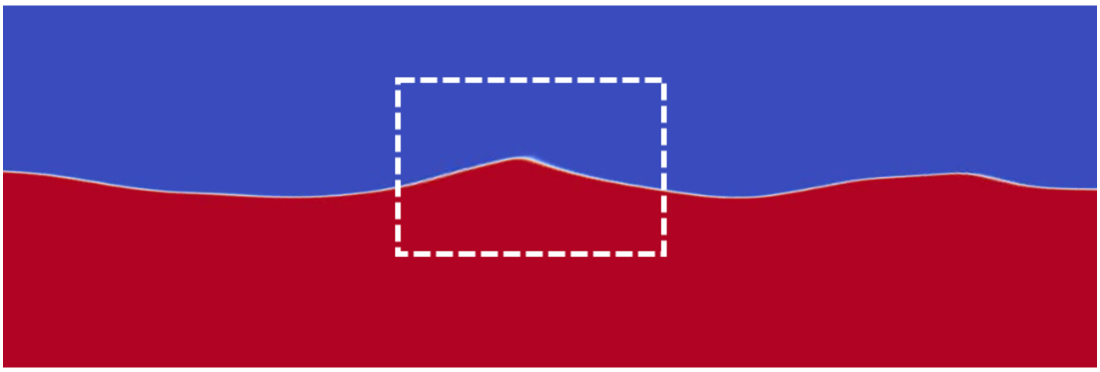


Figure 6.10. Simulation of wave breaking by `foamStar` and `Grid2Grid`

7 Summary

A HOS wrapper program called `Grid2Grid` is developed for the nonlinear wave simulation of numerical solvers. Most of data and functionality is encapsuled as a class to be easily used and maintained. `Grid2Grid` generates dynamic linked library as an independent package to be called and easily used in other languages.

The post processing of HOS is also possible by using `Grid2Grid`. Included post processing program is called `postGrid2Grid`. The usage is explained in Chapter 2.4.

`Grid2Grid` is validated by using the code `foamStar` based on standard multiphase solver of OpenFOAM and also with an experiment. In the experiment, waves corresponding to 1000 year return period in Gulf of Mexico are generated. The wave elevation is measured at the wave breaking position expected by HOS wave theory and compared with the results of the simulation. Good agreement is shown between the measurement and the numerical solutions and the nonlinear wave phenomenon is observed both in experiment and in simulation.

In this document, the `Grid2Grid` program architecture, class and module structure, principle class data and functionality are explained to understand the feature of `Grid2Grid` and to be easily applied to numerical solvers. The interface examples with other programming languages are given as a source code and also in `Grid2Grid` package.

References

Open-source release of hos-nwt. `https://github.com/LHEEA/HOS-NWT`. Accessed: 2017-11-17.

Open-source release of hos-ocean. `https://github.com/LHEEA/HOS-ocean`. Accessed: 2017-11-17.

Bonnefoy, F., Ducrozet, G., Touzé, D. L., and Ferrant, P. (2011). *Time Domain Simulation of Nonlinear Water Waves using Spectral Methods*, pages 129–164. World Scientific.

Ducrozet, G., Bonnefoy, F., Le Touzé, D., and Ferrant, P. (2007). 3-d hos simulations of extreme waves in open seas. *Natural Hazards and Earth System Sciences*, 7(1):109–122.

Ducrozet, G., Bonnefoy, F., Touzé, D. L., and Ferrant, P. (2012). A modified high-order spectral method for wavemaker modeling in a numerical wave tank. *European Journal of Mechanics - B/Fluids*, 34(Supplement C):19 – 34.

Ducrozet, G., Bonnefoy, F., Touzé, D. L., and Ferrant, P. (2016). Hos-ocean: Open-source solver for nonlinear waves in open ocean based on high-order spectral method. *Computer Physics Communications*, 203(Supplement C):245 – 254.

Monroy, C., Seng, S., and Malenica, S. (2017). Développements et validation de l'outil CFD OpenFOAM pour le calcul de tenue à la mer. 15émes Journées de l'hydrodynamique. French.

Seng, S. (2012). *Slamming and Whipping Analysis of Ships*. PhD thesis, Technical University of Denmark, Lyngby, Denmark.

Williams, J. (2015). Open-source release of multidimensional b-spline written in fortran. `https://github.com/jacobwilliams/bspline-fortran`. Accessed: 2017-11-17.