

## SeaFEM reference manual

<http://www.compassis.com>

[info@compassis.com](mailto:info@compassis.com)

November 2018

Version: 15.1.0

### 1. SeaFEM Introduction



SeaFEM is a suite of tools for the computational analysis of the effect of waves, wind and currents on naval and offshore structures, as well as for maneuvering studies. SeaFEM applications include ships, spar platforms, FPSO systems, semisubmersibles, TLPs, marine wind turbines and ocean energy harnessing devices. The wide range of analysis capabilities available in SeaFEM enables the assessment of different design alternatives, significantly reducing overall project costs and timescales.

SeaFEM includes a state-of-the-art radiation and diffraction BEM and FEM solver, enabling frequency domain and direct time-domain analyses of the dynamic response of the structure. Furthermore, SeaFEM is integrated in the Tdyn environment, allowing seamless connection with the FEM structural solver RamSeries, to perform hydroelastic studies.

The different tools available in SeaFEM are fully integrated in an advanced graphic user interface (GUI), for geometry and data definition, automatic mesh generation and visualizing the analysis results. SeaFEM GUI features a versatile tree-like interface for data managing, allowing an easy control of the whole process of entering the analysis data.

To facilitate the data definition process, SeaFEM provides tools to easily configure the type of the analysis to be carried out (seakeeping, maneuvering, towing or fluid-structure interaction). Furthermore, SeaFEM provides a variety of tools which allow having a perfect control over the process and

assess its quality.

### 2. SeaFEM Technical specifications

SeaFEM has been developed for the most realistic simulations of three-dimensional multi-body radiation and diffraction problems, by solving potential flow equations in the time domain, using the finite element method on unstructured meshes. This is highly recommended for the simulation of complex geometries in large and deep domains, and for considering non-linear phenomena in the analysis or multi-body studies. In fact, SeaFEM time-domain simulations can efficiently handle non-linear hydrodynamics effects due to the variable wetted surface, wave impact on the structure, as well as real forward speed or current effects.

Details of the theoretical background of SeaFEM can be found in the Theory manual available in the support page of <http://www.compassis.com>.

SeaFEM has been conceived to simulate seakeeping capabilities of ships and offshore structures, as well as calculating the hydrodynamics loads due to waves, currents, and translational velocities acting simultaneously. Moreover, the software has been equipped with the capability of introducing any external loads acting over the structure under study. Effects of mooring lines can be simulated by using the builtin models.

SeaFEM is also equipped with capabilities to simulate pressurized free surfaces. These capabilities provide the user with the tools for simulating complex devices such as air-cushion vehicles (surface effect ships, for instance) and wave energy converters based on the oscillating water column principle.

The CUDA® - GPU (Graphics Processing Unit) library and the Deflated Conjugated Gradient solver available in SeaFEM, are state of the art implementations aiming at reducing computational time. This leads to being capable of carrying out free floating simulations at full size much faster than real time.

Thanks to its advanced pre-processing capabilities, based on Compass FEM suite's GUI, SeaFEM can easily model complex geometric structures with a best-in-class model preparation time. Additionally, SeaFEM has direct connection with some popular CAD packages. This way, it is not only possible to import the geometrical model but also the parts definition and the tree-like layers structure. Moreover, it is also possible to adapt the GUI, allowing the user to automate and simplify the analysis processes.

SeaFEM is coupled with Compass FEM's structural solver,

Ramseries, allowing seamless one-way and two-way implicit structure-waves interaction analysis (hydro-elasticity) including tools for strength and fatigue assessment of the design (DNV-RP-C203, API RP 2A-WSD).

Furthermore, SeaFEM features a powerful scripting extension, enabling users to enhance simulations without recourse to external compiled subroutines. SeaFEM Tcl interface allows access to advanced features, including writing customized results files, operations on internal structures and execution/communication with external program by using a feature rich extension programming language.h extension programming language.

## 2.1. Applications

SeaFEM is a general-purpose hydrodynamics analysis tool that provides great flexibility to address most types of problems, including:

- Motion analysis of ships and offshore structures in different sea spectra
- Response amplitude operators RAOs with white noise spectrum
- Turning circle maneuver in irregular waves
- Evaluation of floating wind turbines platforms
- Concept design and analysis of wave and wind energy systems
- Seakeeping analysis of offshore structures, including drag effects based on Morison equation
- Multiple body interactions during LNG transfer
- TLP tether analysis
- Fluid-structure interaction analysis (hydro-elasticity) of ships and offshore structures
- Analysis of air-cushion vehicles in waves
- Evaluation of wave loading on lower decks of offshore structures
- Strength and fatigue assessment of offshore structures
- (DNV-RP-C203, API RP 2A-WSD)
- Design and analysis of mooring systems, including intermediate buoys and clump weights
- Motions analysis of FPSOs
- Determination of air gaps
- Estimation of power take out of a wave energy converters, including oscillating water column devices
- Discharging landing craft from mother ships
- Transportation of large offshore structures using barges/ships

## 2.2. System requirements

- Windows NT / XP / XP64 / Vista / Vista64 / 7 / 7 64-bit / 8 / 8 64-bit or Linux 32/64
- Minimum requirements: 1.0 GB RAM (1.5 GB for 64 bits editions) and 500 MB of free hard disk space
- Supports any graphics card with OpenGL acceleration
- Supports CUDA GPU acceleration (required any CUDAenabled and double precision GPU)

## 3. SeaFEM Reference

The following sections contain a reference of the different options available in SeaFEM.

Furthermore, it is possible to obtain help for several items in the data tree and windows simply by passing the mouse pointer over them.

### 3.1. Start Data window

When starting up the Tdyn environment, the start data window will pop up. This window is meant to define the interface so that only those features that are necessary for the case study will be available. This way, the interface will show only those parameters and boundary conditions required, hiding those unnecessary, and therefore making it easier to use and navigate through.

The following figure shows the Tdyn environment and the start data window. In order to use SeaFEM, make sure that **Seakeeping analysis** option is selected from the Simulation type box.

Within the Analysis domain section of the problem selection data tree you can select either frequency or time domain analysis. When selecting the frequency domain option, the remaining options are automatically set up. On the contrary, if the time domain option is selected, then first or second order diffraction radiation options can be chosen depending on the wave order you want to be used for the analysis. Furthermore, under the environment section of the data tree, you can select whether waves and/or currents are to be used. Finally, under the Type of analysis folder, the following three options are available:

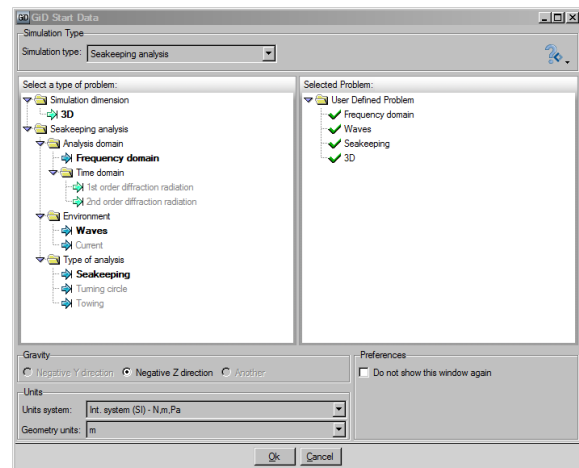
- **Seakeeping:** this option will allow the user to activate body movements on those unrestrained degrees of freedoms.
- **Turning circle:** this option is meant to simulate a body following a circular trajectory. Therefore, surge, sway and yaw will be restrained.
- **Towing:** this option is meant to simulate a ship following linear trajectory with a certain direction and speed. Therefore, surge, sway and yaw will be restrained.

It is obvious that Turning circle and Towing are not compatible options. On the other hand, any other combination of options are compatible simultaneously.

The Start data window can be accessed and modified at any time through the Data menu:

#### Data ► Start data

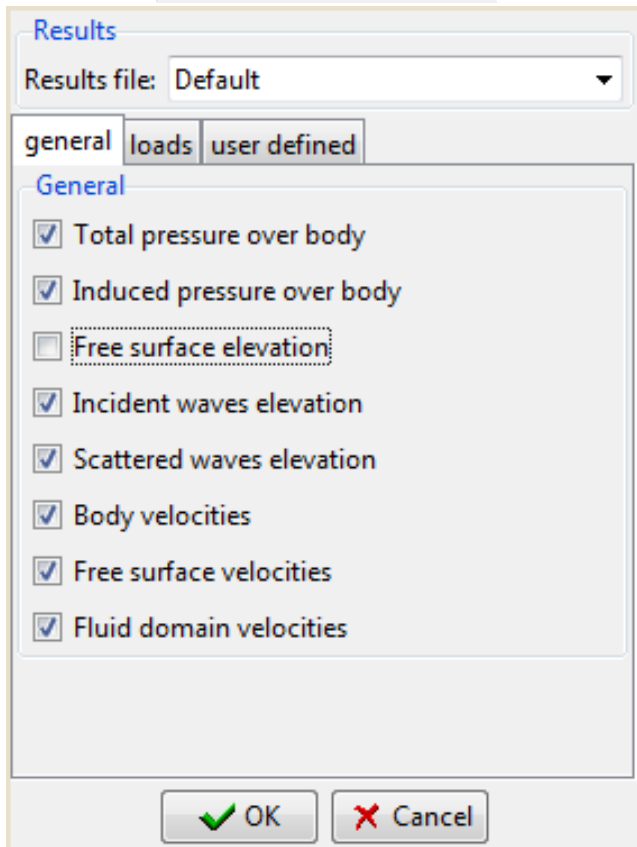
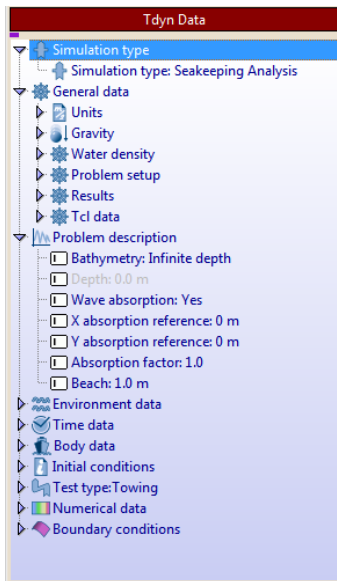
or through the Data tree.



Start Data window

### 3.2. General data

This section contains basic data necessary for simulating any kind of problem. The next figure shows the General data section in the Data tree as well as in the boxes underneath. Data can be modified in both locations, the data tree and in the boxes underneath when a specific section has been selected. It is divided into several sections described below:



**Units:** Set the type of units to be used.

**Gravity:** Set the gravity value, the direction and the units.

**Water density:** Set the water density and the metric unit.

**Problem setup:** This section is equivalent to the start data window. Values can be modified through the start data or here.

**Results:** This section is meant to set what kind of results we want to obtain, and in which format they must be written. Note that if the frequency domain type of analysis is being used it is only necessary to specify the results file format. The rest of the options listed here are only available when a time domain

analysis is undertaken.

- **Results file:** select the format in which the results are to be written. Binary formats are less memory consuming. Binary 1 has to be selected if traditional GID post process is to be used. Binary 2 format (default) is to be used if the newer post process is to be used. Additionally, the native Nemoh file format is available for frequency domain analysis.
- **General:** select those values to be written in the result files. The values shown under "general" are those field values over meshes. Then, they might cause the result fields to be quite large memory.
- **Loads:** Forces and moments acting over the body under study can be recorded along the simulation. Pressure load refers to those loads obtained by integrating the pressure over the body surfaces. total loads refers to all kind of loads, including pressure loads, hydrostatic restoring loads, and any other external load brought into the simulation.
- **Kinematics:** in this section, select the variables to be recorded during the simulation. Movements, velocities and accelerations are referred to the gravity center of the body. Raos stand for Response Amplitude operator.
- **User defined:** Here, the user can create time dependent outputs. These outputs might be written analytically and might be dependent on any variable involve in the simulation, such as position, velocity or acceleration of the body, pressure over pressurized free surfaces, etc.

### 3.3. Computational domain generation

In order to generate a good quality computational domain, it is advised to the user to follow these recommendations:

- For the cases where no currents are imposed, the most indicated shape for the computational domain is a cylinder.
- A region with higher mesh resolution in the free surface close to the floating object is usually needed in order to correctly capture the scattered waves solution (waves radiated and diffracted by the floating object). Hence, it is usually advisable to generate a surface surrounding the floating body where a smaller mesh size will be further assigned. Such a region shall usually roughly coincide with the analysis area (i.e. the region where no artificial absorption is introduced).
- When simulating wave spectra with multiple waves, the absorption area should be at least as long as the maximum wave length. Recommended length is twice the maximum wave length. Nevertheless, if monochromatic wave is used along with Sommerfeld radiation condition, the absorption area might be reduced to half the wave length.
- Computational depth should be no larger than physical depth.
- If simulating infinite depth, it is advised to set the computational depth to the maximum wave length.
- Computational depth might be smaller than physical and/or recommended if the bottom boundary condition is used. Care must be taken since the depth of the body should be small compared to the computational depth when using this option.

### 3.4. Problem description

In this section, some key parameters necessary to carry out the simulation have to be provided. The following figure shows the data interface. Note that this section of the data tree becomes more simple when using the frequency domain type of analysis. In that case, only the type of bathymetry to be used (and optionally their corresponding depth) must be specified.

**Problem description**

Bathymetry: Infinite depth

Depth: 0.0 m

☒ Wave absorption

X absorption reference: 0 m

Y absorption reference: 0 m

Absorption factor: 1.0

Beach: 1.0 m

OK Cancel

Problem description data interface

#### Bathymetry

- **Infinite depth:** to be used when the depth is much larger than the wave lengths. In this case, the depth of the domain is recommended to be at least equal to half the wave length of the largest incident wave. However, smaller computational depths can be used in combination with the *Bottom* boundary condition to simulate larger depths. This can be done when the characteristic length of the body under analysis is small compared to the computational depth.
- **Constant depth:** to be used when the bottom is flat, and the depth is constant and smaller than half the wave lengths of the largest wave. Smaller computational depths

can be used in combination with the *Bottom* boundary condition the same way it has been indicated previously.

- **Depth:** only available if *Bathymetry=Constant depth* was selected. The real depth of the problem must be introduced in the box

**Wave absorption:** select if scattered waves generated by the presence of the body are to be absorbed in order to avoid reflection at the edge of the computational domain. The absorption area will start at a specific distance (*Beach*) from a reference point located on the free surface. Therefore, there will be no absorption in the circle with center the reference point, and radius the value introduced in "*Beach*". The area with no dissipation will be referred as the analysis area since no artificial dissipation is introduced on purpose to damp waves refracted and radiated by the body.

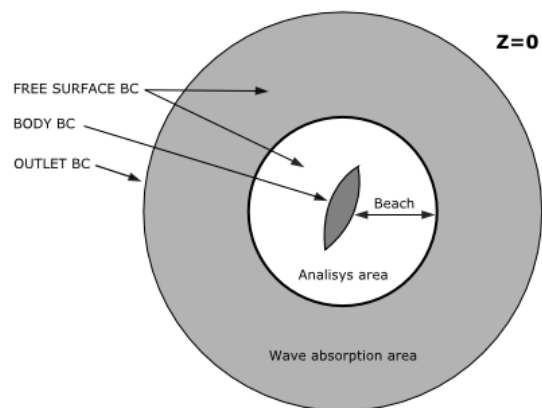
**X absorption reference:** X coordinate of the reference point to determine the analysis and absorption area.

**Y absorption reference:** Y coordinate of the reference point to determine the analysis and absorption area.

**Absorption factor:** determines how strong the dissipation is (recommended value "1"). Large absorption factors might cause instabilities and/or wave reflection. Smaller values, while being less likely to cause instabilities, but will require larger computational domains to damp refracted and radiated waves.

**Beach:** determine how far, from the reference point, the free surface absorption starts.

**Sommerfeld radiation condition:** This option is only available in those cases where the body is subjected to waves, has no translational movement (turning circle/towing) and is in the absence of currents. In these cases, the largest waves can be hard to be dissipated in the absorption area unless a large computational domain is used. To avoid this situation, a Sommerfeld radiation condition can be used to allow the largest waves to leave the domain across the edge of the computational domain. Therefore, the combined action of the dissipation area and the Sommerfeld radiation condition is the best choice to avoid reflection of waves onto the edges.



Analysis area and wave absorption area

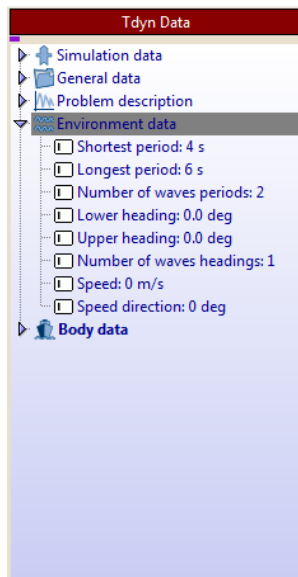
### 3.5. Environment data

This section is meant to provide the data necessary to simulate the marine environment. Different options are available depending on whether frequency or time domain is under consideration.

#### • Frequency domain:

In this case, the wave spectra will be a set of monochromatic waves defined by the period and heading of each wave. The data to be inserted is quite simple and the user only needs to define the following inputs:

- Shortest and Longest Period: Shortest and Longest wave periods that the wave spectra will have.
- Number of waves periods: total number of wave periods to be used in the wave spectra.
- Lower and Upper heading: Lower and Upper wave heading that the wave spectra will have.
- Number of waves headings: Total number of wave headings to be used in the wave spectra.
- Speed: Total forward speed that the bodies will have.
- Speed direction: Direction of the speed for all bodies.



#### • Time domain:

Different sort of wave spectra are available for time domain analyses, each one requiring of specific data. The wave spectra is introduced in SeaFEM as an incident velocity potential.



**Currents**

Velocity: 0.0 m/s

Direction: 0.0 deg

OK Cancel

Environment data interface

Wave environment data and currents environment data can be defined through the menu options:

**Environment data ► Waves**

and

**Environment data ► Currents**

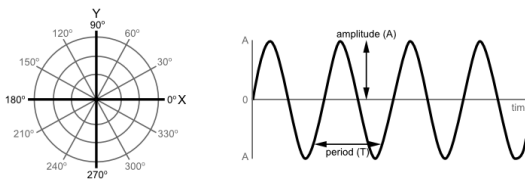
respectively.

### 3.5.1. Wave environment data

Next is a description of the different inputs to be provided by the user:

**Wave spectrum type:** select the type of incident wave environment. The following options are available.

- **Monochromatic wave:** this is the simplest spectrum available. This option generates an incident wave that corresponds to a monochromatic wave. In order to determine this monochromatic wave, the wave amplitude, period and direction of propagation must be provided.



Wave parameters

- **Pearson Moskowitz spectrum:**

$$S(T) = H_s^2 T_m \left( \frac{0.11}{2\pi} \right) \left( \frac{T_m}{T} \right)^{-5} \exp \left[ -0.44 \left( \frac{T_m}{T} \right)^4 \right]$$

where T is the wave period;  $H_s$  is the significant wave height;  $T_m$  is the mean wave period, which is obtained via  $T_m = 2\pi m_0 / m_1$ , with  $m_0$  and  $m_1$  the zero and first moments of the wave spectrum. This is probably the simplest idealized spectrum, obtained by assuming a fully developed sea state, generated by wind blowing steadily for a long time over a large area. For further information, please refer to the SeaFEM Theory Manual.

- **Jonswap2 spectrum:** The JONSWAP spectrum was established during a joint research project, the "JOint North Sea Wave Project". This is a peak-enhanced Pierson-Moskowitz spectrum given on the form

$$S(\omega) = \left( \frac{5}{16} \cdot H_s^2 \cdot \frac{T_p^4}{2\pi} \right) \cdot \exp \left( -1.25 \left( \frac{T_p}{T} \right)^4 \right) \cdot (1 - 0.287 \cdot \log(\gamma)) \cdot \gamma^Y$$

$$Y = \exp \left[ - \left( \frac{0.159 \omega T_p - 1}{\sigma \sqrt{2}} \right)^2 \right]$$

where  $\omega = 2\pi/T$ ,  $\sigma = 0.07$  for  $\omega \leq 6.28/T_p$ ,  $\sigma = 0.09$  for  $\omega > 6.28/T_p$ , T is the wave period;  $H_s$  is the significant wave height;  $T_p$  is the peak wave period and  $\gamma$  is the peakedness parameter. For further information, please refer to the SeaFEM Theory Manual.

- **Jonswap spectrum:** An alternative definition of the JONSWAP spectrum given by

$$S(\omega) = \left( \frac{155 H_s^2}{T_m^4 \omega^5} \right) \cdot \exp \left[ -944 T_m^4 \omega^4 \right] (3.3)^Y$$

$$Y = \exp \left[ - \left( \frac{0.191 \omega T_m - 1}{\sigma \sqrt{2}} \right)^2 \right]$$

where  $\omega = 2\pi/T$ ,  $\sigma = 0.07$  for  $\omega \leq 5.24/T_m$ ,  $\sigma = 0.09$  for  $\omega > 5.24/T_m$ , T is the wave period;  $H_s$  is the significant wave height;  $T_m$  is the mean wave period, which is obtained via  $T_m = 2\pi m_0 / m_1$ , with  $m_0$  and  $m_1$  the zero and first moments of the wave spectrum. For further information, please refer to the SeaFEM Theory Manual.

- **White noise:** introduce a number of waves with frequencies uniformly distributed across an interval, and with same amplitude and direction. This spectrum is used to carry out response amplitude operator (RAO) analysis with the time-domain solver.
- **Customize:** a spectrum can be defined based on the significant wave height and mean wave period.
- **Read from file:** by using this option, any generic wave spectrum can be defined by the user. To this aim, a text file must be provided in the data tree entry that becomes available when the 'Read from file' option is selected. In the text file the user must indicate the relevant wave parameters (period T, amplitude A, wave direction G and wave phase P) for each wave component conforming the spectrum. The specific format of the wave spectrum file can be viewed in the following example.

SeaFEM Spectrum\_file version 1.0

NWaveComponents 20

TWaves AWaves GWaves PWaves

14.6613 2.56028e-005 -0.349066 2.94053

12.6662 0.000229263 -0.116355 3.14788

13.7174 0.000229263 0.116355 4.55531

11.8723 2.56028e-005 0.349066 2.25566

7.21275 0.0123038 -0.349066 2.92168

7.73874 0.110176 -0.116355 0.917345

8.79714 0.110176 0.116355 5.20248

7.21466 0.0123038 0.349066 3.09133

5.66508 0.0427017 -0.349066 5.92504

5.86992 0.382376 -0.116355 2.74575

6.4778 0.382376 0.116355 3.80133  
5.77667 0.0427017 0.349066 0.967611  
5.33589 0.0272918 -0.349066 2.40646  
5.20454 0.244386 -0.116355 4.50504  
4.92538 0.244386 0.116355 5.62973  
5.17884 0.0272918 0.349066 4.56788  
4.13839 0.0189668 -0.349066 3.38664  
4.07775 0.16984 -0.116355 5.73655  
4.20474 0.16984 0.116355 1.88496  
4.65867 0.0189668 0.349066 5.62345

This user defined spectrum file is equivalent to a Jonswap spectrum realization with the following characteristics:

- Mean wave period = 5 sec.
- Significant wave height = 2 m.
- Shortest period = 4 sec.
- Longest period = 15 sec.
- Number of wave periods = 5
- Mean wave direction = 0 deg.
- Spreading angle = 40 deg.
- Number of wave directions = 4

**Customize spectrum:** if the option "customize" has been selected in "Wave spectrum type", the user can introduced a spectrum based on the significant wave height and mean wave period. For instance, a Pearson moskowitz spectrum could be written as follows:

$$(H_s^2 \cdot T_m) \cdot \frac{0.11}{2 \cdot \pi} \cdot \left( w_s \cdot \frac{T_m}{2 \cdot \pi} \right)^{-5} \cdot \exp \left( -0.44 \cdot \left( w_s \cdot \frac{T_m}{2 \cdot \pi} \right)^4 \right)$$

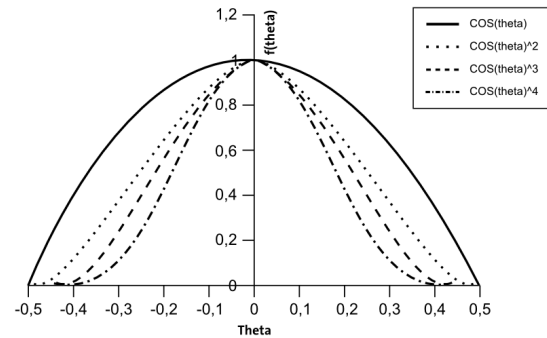
where  $H_s$  represents the significant wave height;  $T_m$  represents the mean wave period,  $w_s$  represent the wave frequency  $\omega=2\pi/T$  (See appendix A).

**Directional wave energy:** this variable allows the introduction of a directional wave nergy distribution  $f(\theta)$ , aiming at reproducing spectra with higher energy around the mean direction of propagation, and decaying as the direction diverge from the mean A typical directional wave energy distribution can be expressed as:

$$f(\theta) = \cos^2 \left( \frac{2\theta}{3} \right), \text{ where } \theta = \pi \cdot \frac{\gamma - \gamma_m}{\gamma_{\max} - \gamma_{\min}}$$

$$\cos^2 \left( \frac{2}{3} \cdot \gamma_s \cdot \pi \right)$$

and  $\theta$  goes from  $-\pi/2$  to  $\pi/2$ . This directional energy distribution should be introduced with the following syntax:



Directional wave energy examples.

**Amplitude:** amplitude of monochromatic wave or amplitude for white noise spectrum waves.

**Period:** period of the monochromatic wave.

**Heading:** direction of the monochromatic wave or direction for white noise spectrum waves. The wave heading (direction) is defined as the angle from the positive global X axis to the direction in which the wave is travelling, measured anti-clockwise when seen from above. Therefore waves travelling along the X axis (from -X to +X) have a 0 degree wave direction, and waves travelling along the Y axis (from -Y to +Y) have a 90 degree wave direction.

**Mean wave period:** mean wave period for wave spectrum such as Pearson Moskowitz, Jonswap, etc.

**Significant height:** significant wave period for wave spectrum such as Pearson Moskowitz, Jonswap, etc.

**Shortest period:** correspond to the wave with maximum frequency to be considered when discretizing a spectrum.  $T_{\min}=T_m/2.2$  recommended.

**Longest period :** correspond to the wave with minimum frequency to be considered when discretizing a spectrum.  $T_{\max}=2.2T_m$  recommended.

**Number of wave periods:** or number of wave frequencies to be used.

**Mean wave heading:** mean direction of wave propagation. It is provided in the form of an angle  $\theta$  measured with respect to the X global axis. The wave heading (direction) is defined as the angle from the positive global X axis to the direction in which the wave is travelling, measured anti-clockwise when seen from above. Therefore waves travelling along the X axis (from -X to +X) have a 0 degree wave direction, and waves travelling along the Y axis (from -Y to +Y) have a 90 degree wave direction.

**Spreading angle:** angular sector  $\Delta\theta$  within which the waves propagate. Such an angular sector is always centered at the mean wave heading so that the waves propagate within the range  $[\theta - \Delta\theta/2, \theta + \Delta\theta/2]$

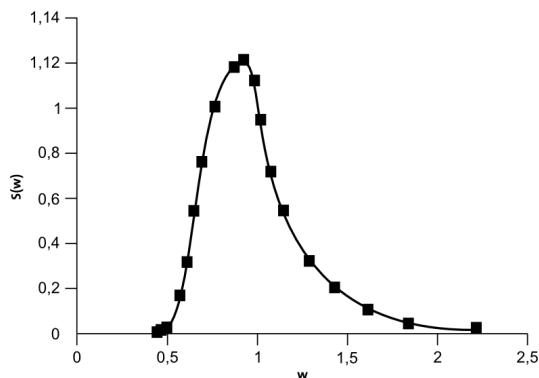
**Number of wave headings:** in case the waves propagate within



an angular sector specified by the mean wave heading and the spreading angle, this parameter determines how many directions such an angular sector will be discretized into.

**Realization repeatability:** this option must be activated if the user wishes to run exactly the same spectrum realization in further simulations. By contrast, if such an option remains deactivated, random realizations of the same given spectrum will be used when running the simulation several times.

*Note:* the total number of waves used in the realization will be the "number of wave periods" times "Number of wave directions".



Pearson Moskowitz discretization.  $H_s=1$ ;  $T_m=1$ ;  $T_{max}=2.2T_m$ ;  $T_{min}=T_m/2.2$ ;  $N=20$ .

### 3.5.2. Currents environment data

Next is a description of the various inputs to be provided by the user when using currents:

**Velocity:** velocity of the water current.

**Direction:** direction of the water current.

### 3.6. Time data

In this section, several parameters regarding the timing of the simulation are to be defined. Note that time data presented in this section only concern to time domain analysis but not frequency domain.

Time data interface

**Simulation time:** length in time of the simulation.

**Time step:** time step to be use for the time marching schemes. The time step introduced in this box will be the one used unless a zero value is introduced. If a zero value is introduced, the time step will be internally calculated by SeaFEM based on the minimum mesh size and the stability parameter  $\beta=g\Delta t^2/\Delta z_{min}$ .

**Output step:** time lag between recordings. Values corresponding between two time steps are linearly interpolated between the previous and the next time step.

**Start time recording:** set the point in time when the writing of the results will start.

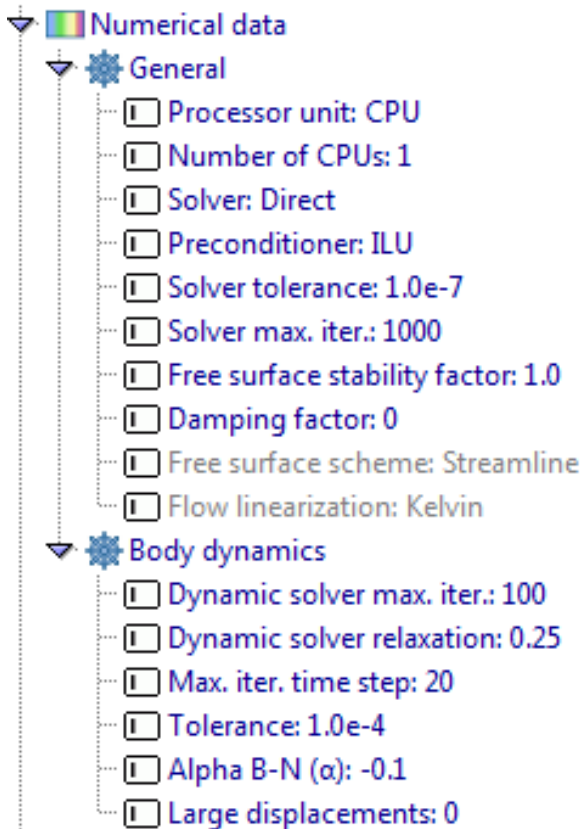
**Initialization time:** set an initialization time period. During this period, the wave amplitudes and currents will be increased smoothly from zero to their final values following this expression: Initialization factor =  $0.5 \cdot (1 - \cos(\pi \cdot \text{time} / \text{time}_{init}))$ . This initialization process is meant to avoid long transient behaviours due to sudden initializations. Sudden initializations may lead to an unrealistic and highly energetic transient behaviours. In this cases, longer simulations are required so the unrealistic high energy behaviour will dissipate over time.

### 3.7. Numerical data

As well as the time data presented in the previous section of the manual, numerical data shown herein concerns only time domain analysis.

The **Numerical data** section of the SeaFEM data tree collects

information concerning the numerical algorithms underlying the SeaFEM solver. Most of the computational time required by SeaFEM is spent in solving the linear system of equations resulting from the discretization of the governing equations. Therefore, selecting the correct parameters will enhance having faster and even more accurate simulations. Numerical parameters that affect the behavior and performance of the main SeaFEM solver (that devoted to solve the finite element based potential flow problem) appear within the **General** subsection of the SeaFEM data tree. On the other hand, numerical parameters related to the multi-body dynamics solver are collected within the **Body dynamics** subsection of the SeaFEM data tree.



Body dynamics solver numerical data

General numerical data

A brief description of the meaning of each numerical parameter and the possible values they can take is provided in the following list:

#### General numerical data

**Processor Unit:** while most part of the computations are carried out by the CPU, SeaFEM provides the option of solving the linear system of equations faster by means of the graphic processor unit (GPU). Since most of the computational time is spent in solving these linear systems, GPU provides a way to speed up our SeaFEM computational time. Any GPU device supporting CUDA and double precision is suitable of being used (see <http://developer.nvidia.com/cuda-gpus> for further information).

- **CPU:** all computations are carried out in the CPU.
- **CPU+GPU:** linear systems of equations concerning the main SeaFEM problem are solved in the GPU. Any other computation, as for instance the solution of the body dynamic's problem is carried out in the CPU.

**Number of CPUs:** those calculations carried out in the CPU may take advantage of the multithread parallel computing capabilities of modern processors. The number of CPUs indicates how many of the available computer's processors are

going to be used during the calculation.

**Solver:** these is the list of available solvers. It actually depends on the type of analysis under consideration. In those cases where either current or body translational movements (turning circle/towing simulations) are present, only Stab-BiCG and Direct solvers are available. When neither currents nor translational movements exist, the complete list of solvers becomes available. When the CPU+GPU option is being used, the Direct solver is not available in any case.

- **Conjugate gradient:** iterative solver suitable to solve problems leading to a linear set of algebraic equations with a symmetric matrix structure
- **Bi-conjugate gradient:** iterative solver suitable to solve problems leading to a linear set of algebraic equations with a non-symmetric matrix structure
- **Stab bi-conjugate gradient:** modified version of the bi-conjugate gradient solver that can provide more stability in some ill-posed numerical problems.
- **Deflated conjugate gradient:** deflated version of the conjugate gradient iterative solver. Be aware that the deflation process is not always guaranteed to speed up the solving process. Based on our experience, most of the time it does, but care must be taken when selecting this option. If you feel that SeaFEM is running slow under this option, stop the calculation and select the CG solver instead, compare the speed of the simulation, and act consequently.
- **Direct:** direct solver based on the third-party IntelMKL numerical solvers library.

**Preconditioner:** these is the list of available preconditioners to be used in conjunction with the iterative solvers listed above in order to speed-up the calculations.

- if the Processor Unit is set to CPU, the *ILU* preconditioner is recommended.
- if the Processor Unit is set to CPU + GPU and neither currents nor translational movements are simulated: the *SPAI* preconditioner is recommended.
- if the Processor Unit is set to CPU + GPU and either currents or translational movements are simulated: the *Diagonal* preconditioner is recommended.

**Solver tolerance:** maximum tolerance allowed to reach convergence when using iterative solvers. The default value  $10^{-7}$  is recommended.

**Solver max iterations:** maximum number of iterations to be carried out by the solver until convergence is achieved. The default value 1000 is recommended.

**Free surface stability factor:** this factor controls the time step as explained in the time data section, unless a positive time step has been prescribed. If neither currents nor translational movements are simulated, typical values for stability are in the order of 1. If either currents or translational movements are simulated, typical values for stability are in the order of 0.1-0.001, depending on the Froude number. The larger the froude number, the smaller the stability parameter and the time step.

**Damping factor:** this parameter is available in those cases where the body under study has unrestrained degrees of freedom. Sometimes, the only mechanism to dissipate energy by the body is through wave radiation. However, this mechanism might not be dissipative enough and might cause very long transient periods due to its low dissipation capabilities. This might be a problem specially when the body is excited with waves whose frequencies are near the resonance

frequency of the body. This problem can be mitigated by introducing a small dissipation which is only noticeable near resonance. This is carried out by bringing a percentage of the critical damping into the dynamic equations of the body. It is recommended to use this option only in those cases where it is necessary due to the low wave radiation capacity of the body, and where the body is excited near its resonance frequency. Values between 0 and 0.05 are recommended, depending on the case.

**Free surface scheme:** this option is only available when either currents or translational movements exist. This parameter actually determines the numerical scheme to be used when solving the free surface boundary condition with convective terms.

- **Streamlines:** in this case, the convective term of the free surface boundary condition is obtained by using a streamline differential operator that actually uses two points upstream and one point downstream to evaluate the derivatives along the streamline.
- **FEM SUPG:** this is an alternative method for the integration of the free surface boundary condition. In this case a finite element based SUPG stabilization scheme is used.

**Flow linearization:** this option is only available when either currents or translational movements exist. It specifies the type of linearization to be used when integrating the convective terms within the free surface boundary condition.

- **Kelvin:** flow around the body is assumed as if is not perturbed by the presence of the body.
- **Slow Kelvin:** Kelvin linearization above is used but convective terms in the free surface are not taken into account.
- **Double body:** flow around the body is assumed as if the free surface behaves as a wall.
- **Slow double body:** double body linearization above is used but convective terms in the free surface are not taken into account.
- **Non-linear:** flow around the body is continuously updated to take into account the epsepsence of the ody and the effects of waves generated at the free surface.

## Body dynamics numerical data

**Dynamic solver max. iterations:** máximo number of iterations allowed for the dynamic solver.

**Dynamic solver relaxation:** this parameter concerns the numerical relaxation of the dynamic solver. It must be greater than zero.

**Max iterations time step:** This parameter is available in those cases where the body under study has unrestrained degrees of freedom. In these cases, an iterative procedure must be carried out within each time step to reach convergence of the body dynamics driven by the hydrodynamic and external loads acting on the body. This parameter sets the maximum number of iterations allowed per time step until convergence is achieved.

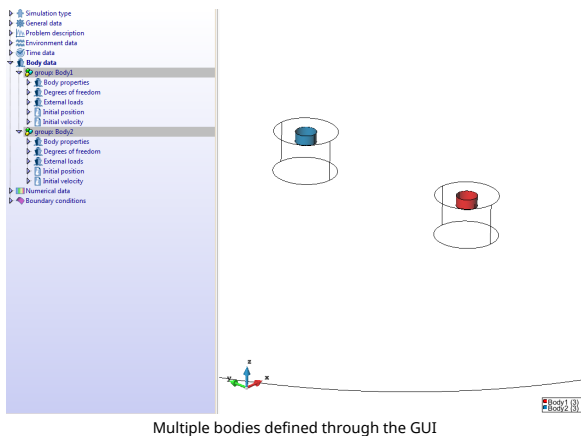
**Tolerance:** maximum tolerance allowed to reach convergence in the iterative procedure carried out within each time step.

**Alpha B-N:** this parameter concerns the stability of the Bossak-Newmark scheme used to solve the multibody dynamics system. It usually takes a negative value. A positive value may be advisable when the time step is very small since in this case a first order scheme can increase stability while precision is preserved inherently because of the small value of the time step.

**Large displacements:** this option must be activated to take into account large displacements when solving the multi-body dynamics system. If this option is active, the inertia matrix of the bodies is updated every time step to take into account the finite rotation of the body. Forces and moments are updated as well to take into account this effect. Note that large displacements have limited application within SeaFEM since the actual position of the body regarding the incident wave is not updated. Hence, caution is advised when interpreting the results obtained using the large displacements option.

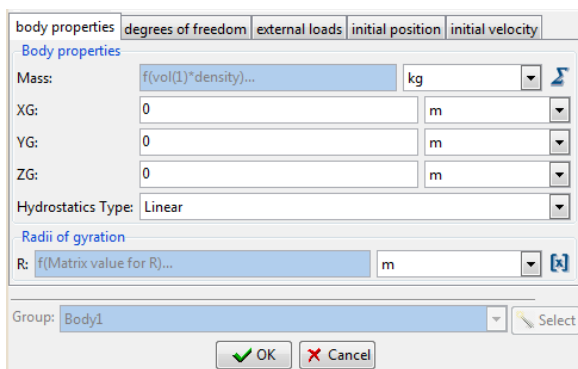
### 3.8. Body data

Body Data section is intended to allow the user to define several bodies and their corresponding properties. The user can create as many bodies as necessary, each one being assigned to a different group of geometrical entities. In the figure below for example, two different bodies have been defined, each one being assigned to a different cylindrical floating body.



Multiple bodies defined through the GUI

For each body, information regarding the mass and the radii's of inertia and unrestrained degrees of freedom must be provided. This is so irrespectively of whether frequency or time domain options are under consideration. If a time domain calculation is undertaken, additional external forces and moments can be defined for each body.

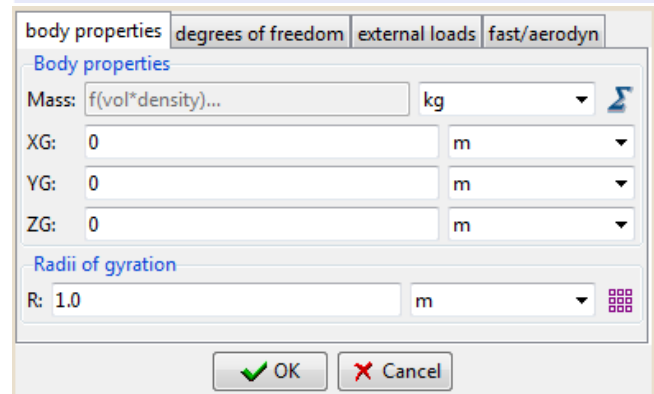
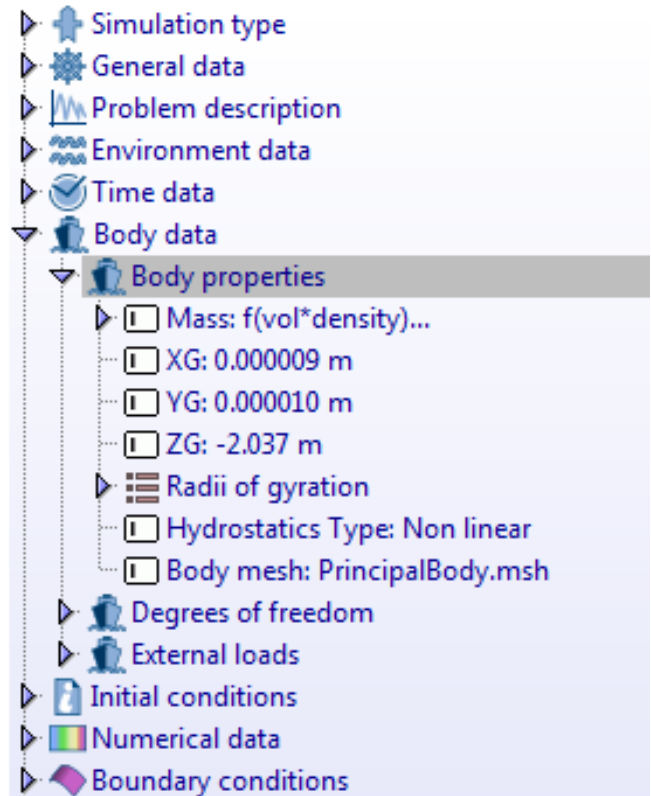


Body data definition window

Each body data section is described in detail in what follows.

#### 3.8.1. Body properties

Basic data regarding the body must be provided in order to solve the body's dynamics if the body has unrestrained degrees of freedom.

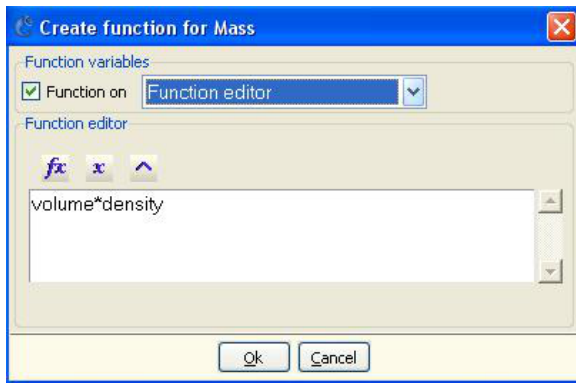


Body properties interface

To define body properties use the menu option:

#### Body data ► Body properties

**Mass:** the body mass can be introduced in two different ways: either introducing the exact value or using the function editor. When using the function editor, the mass can be calculated as an analytical value depending of some variables used by SeaFEM. For example, for freely floating objects where the mass must equal the mass of the displaced water, we could write  $Mass=volume \cdot density$ , where volume refers to the displaced water volume, and density refers to the water density. This specific case is shown in the following figure.



Mass function editor

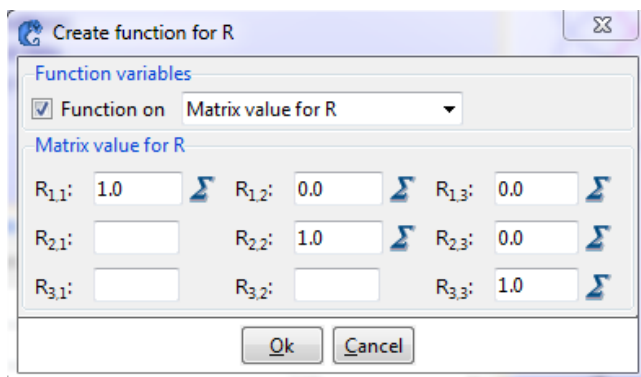
**XG:** introduce the x coordinate of the gravity center of the body.

**YG:** introduce the y coordinate of the gravity center of the body.

**ZG:** introduce the z coordinate of the gravity center of the body.

**Radii of gyration:** the elements of the inertial matrix are related to the radii of gyration as:  $I_{ii} = \text{Mass} \cdot r_{ii}^2$ ;  $I_{ij} = \text{Mass} \cdot r_{ij}^2$ . Then:

- **rxx:**  $r_{xx} = \sqrt{\frac{I_{xx}}{\text{Mass}}}$
- **rxxy:**  $r_{xxy} = \left( \frac{P_{xy}}{|P_{xy}|} \right) \cdot \sqrt{\frac{|P_{xy}|}{\text{Mass}}}$
- **rxz:**  $r_{xz} = \left( \frac{P_{xz}}{|P_{xz}|} \right) \cdot \sqrt{\frac{|P_{xz}|}{\text{Mass}}}$
- **ryy:**  $r_{yy} = \sqrt{\frac{I_{yy}}{\text{Mass}}}$
- **ryz:**  $r_{yz} = \left( \frac{P_{yz}}{|P_{yz}|} \right) \cdot \sqrt{\frac{|P_{yz}|}{\text{Mass}}}$
- **rzz:**  $r_{zz} = \sqrt{\frac{I_{zz}}{\text{Mass}}}$



Radii of gyration matrix interface

**Hydrostatic type:** this parameter has two possible values, **Linear** and **Non-Linear**. By default, the **linear** option is used so that the calculation of the hydrostatic recovery forces is linearized. By doing this, the displacements of the floating structure are assumed to be small and the hydrostatic restoration coefficients assumed to be constant. This allows for the coefficients to be calculated just once at the beginning of the simulation taking into account the initial configuration. By using the **non-linear** option, the hydrostatic restoration coefficients are assumed to depend on the actual movement of

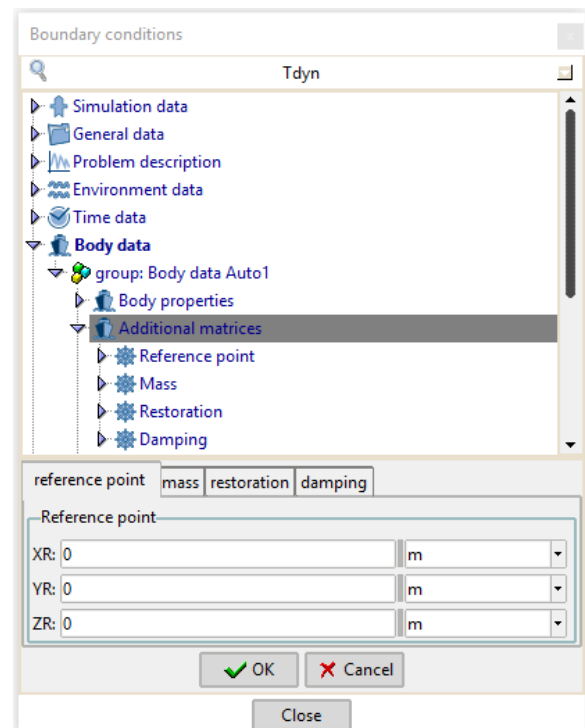
the floating structure and they must be evaluated at each time step. To this aim, an auxiliary body mesh (containing the entire body) is necessary for tracking the actual position of the body surface during the simulation, thus allowing for the proper integration of the hydrostatic pressure. Such an auxiliary mesh must be generated for each body under analysis and exported in a text file using GiD mesh format. The generated mesh file must be provided as "Body mesh" option under Body properties.

The Non-Linear hydrostatic type option allows for the simulation of those phenomena that are inherently non-linear as for instance the parametric resonance.

### 3.8.2. Additional matrices

The *Additional matrices* option can be used to provide additional mass, restoration and damping matrices for the body under consideration. This can be useful when we want to take into consideration mass, stiffness and damping effects different from the hydrodynamic effects directly accounted for by SeaFEM's solver. This is the case, for instance, when parts of the body located above the free surface may be anticipated to suffer significant aerodynamic forces affecting the dynamics of the whole system. A typical example is the dynamic effect of the wind turbine attached to a floating TLP structure (see SeaFEM's tutorials manual for further details on this example).

In these cases, a single mass, restoration and damping matrix can be specified for each body through the data tree's GUI, as shown in the following figure.



Additional matrices option as it appears in the SeaFEM's data tree. As can be seen, a reference point must be provided.

If more than one individual matrix needs to be provided, it can still be done by using the SeaFEM's TCL extension. In this case, an index identifying the body to which the matrix is associated and the corresponding reference point must be provided for each individual matrix (see also SeaFEM's tutorials manual for further details on this usage). The actual syntax looks like this:



*TdynTcl\_Add\_Mass\_Matrix* <bIdx> [list <xcoord> <ycoord> <zcoord>]  
[list <M11> <M12> ... <M66>]

where <bIdx> is the index that identifies the body, <xcoord>, <ycoord>, <zcoord> are the coordinates of the reference point and <M11> ... <M66> are the 36 coefficients of the matrix.

Similarly, the TCL's extension syntax for additional stiffness and damping matrixes reads as follows.

*TdynTcl\_Add\_Stiffness\_Matrix* <bIdx> [list <xcoord> <ycoord> <zcoord>] [list <S11> <S12> ... <S66>]

*TdynTcl\_Add\_Damping\_Matrix* <bIdx> [list <xcoord> <ycoord> <zcoord>] [list <D11> <D12> ... <D66>]

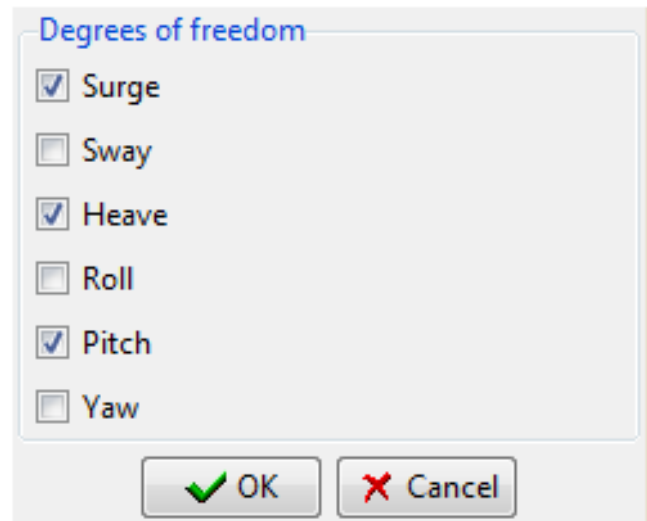
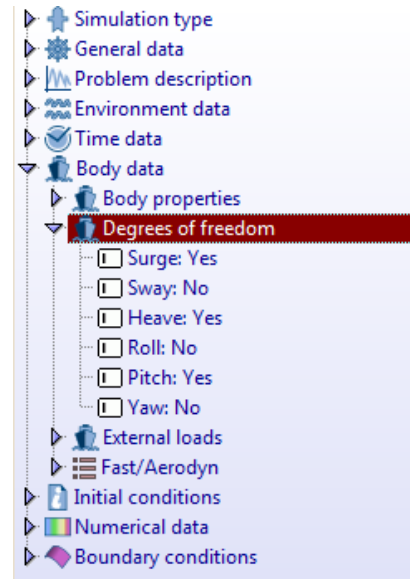
### Managing an arbitrary reference point

It has already been noted that a reference point, to which the matrix coefficients are referred, must be provided for all additional matrices. With this information, SeaFEM internally evaluates the actual position of the gravity center of the body system and transforms the matrices so that they are all referred to the new gravity center. Note that such an internally updated gravity center may not coincide in general with that specified by the user in the "Body properties" tab that actually corresponds to the reference point of the body's main component. In any case, the actual position of the gravity center of the system is output for each body at the beginning of the information file when running the simulation. Finally, the dynamics of the system are finally solved in the body's frame of reference so that all the results are referenced to the corresponding updated gravity center.

Sometimes it can be useful to set the mass of the body to zero and actually enforce the mass and inertia properties of the body by using the additional mass matrix option. If this is the case, the radii of gyration data in the *Body properties* section of the data tree has no effect since the corresponding inertia coefficients evaluate to zero (i.e.  $I_{ij} = R_{ij}^2 \cdot M$  being  $M$  equal zero) and the actual inertia matrix corresponds to that provided by the additional mass matrix option. Note that in the case of using the frequency domain module of SeaFEM, setting the mass of the body to zero has the effect of the solver to internally compute the actual mass of the body from the displacement and the water density. This is equivalent to what is done when using the time domain module if the "vol\*density" formula is used in the "Mass" field of the body properties data.

### 3.8.3. Degrees of freedom

Next figure shows the interface where the user can activate/deactivate the unrestrained degrees of freedom. When performing analysis such as *Turning circle* or *Towing*, the *Surge*, *Sway* and *Yaw* are restrained degrees of freedom since the body is forced to follow a specific trajectory, while the *Heave*, *Roll* and *Pitch* are unrestrained.



Degrees of freedom interface

Degrees of freedom can be activated/deactivated through the menu option:

#### Body data ► Degrees of freedom

**Surge:** select if the floating object is supposed to translate along the x axis.

**Sway:** select if the floating object is supposed to translate along the y axis.

**Heave:** select if the floating object is supposed to translate along the z axis.

**Roll:** select if the floating object is supposed to rotate around the x axis.

**Pitch:** select if the floating object is supposed to rotate around the y axis.

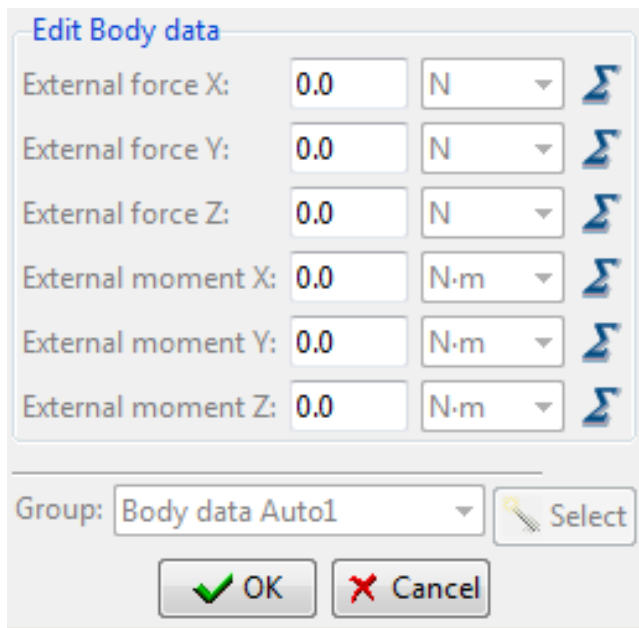
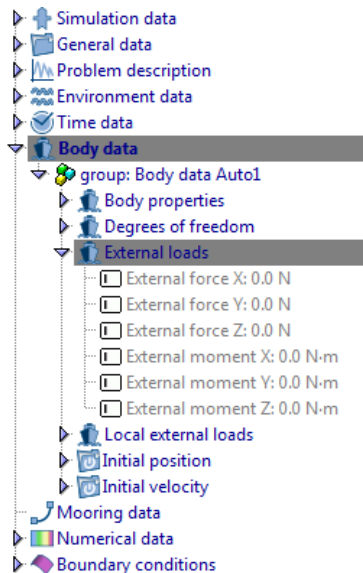
**Yaw:** select if the floating object is supposed to rotate around the z axis.

### 3.8.4. External loads

External loads (forces and moments) can be defined either as a



constant value or via a set of analytical functions. These functions can be tuned to model, for instance, tension legs acting as springs. These external loads may be dependent on any of the variables listed in Appendix A.



External loads interface

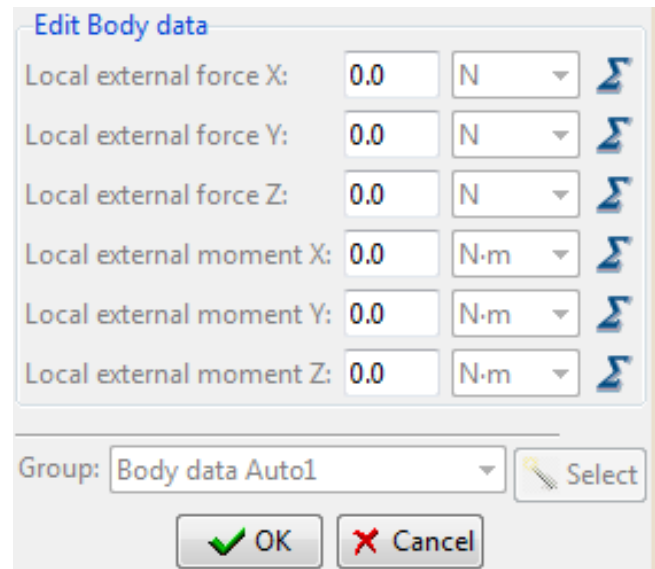
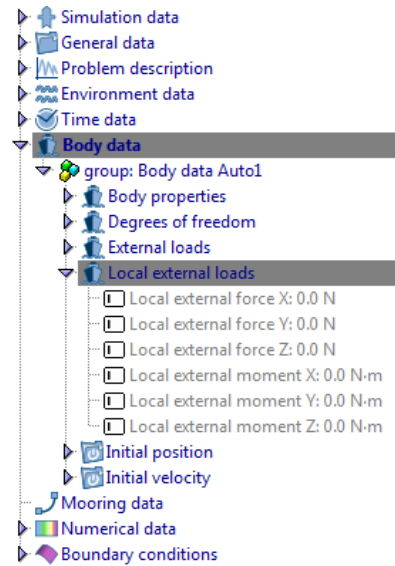
External loads can be defined using the menu option:

**Body data ► External loads**

### 3.8.5. Local external loads

External loads (forces and moments) can also be defined relative to a local reference frame fixed to the body. This can be useful, for instance, to model the action of a PTO system when the displacements of the bodies cannot be considered to be small.

Local external loads can be defined either as a constant value or via a set of analytical functions. These external loads may be dependent on any of the variables listed in Appendix A.



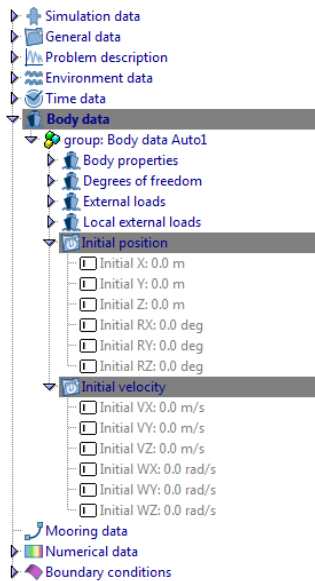
Local external loads interface

External loads can be defined using the menu option:

**Body data ► Local external loads**

### 3.8.6. Initial conditions

This section is used to introduced initial conditions for each body under analysis. For instance, if a extinction test in roll is to be carried out, the initial roll angle has to be known and introduced in the corresponding box. Initial velocities are applied in a similar way.

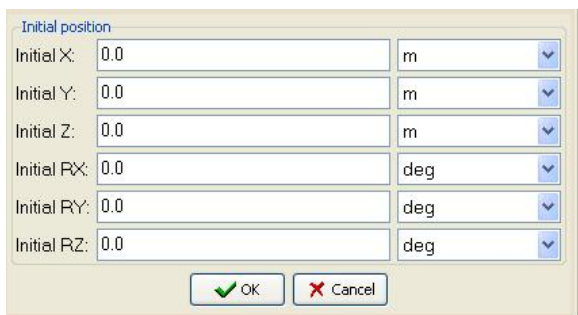


### Initial Position

Initial position options are available through the menu:

#### Body data ► Initial position

- Initial X: set a initial X position for the object.
- Initial Y: set a initial Y position for the object.
- Initial Z: set a initial Z position for the object.
- Initial RX: set a initial RX rotation angle for the object.
- Initial RY: set a initial RY rotation angle for the object.
- Initial RZ: set a initial RZ rotation angle for the object.



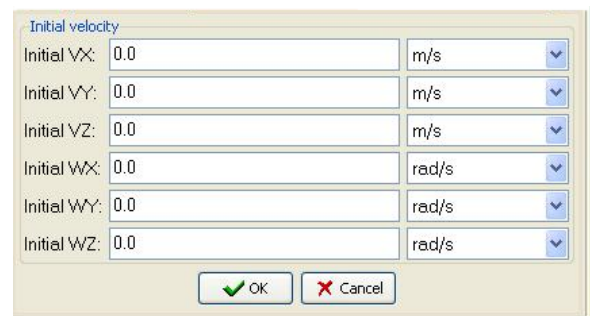
Initial position interface

### Initial Velocity

Initial velocity data can be defined through the menu:

#### Body data ► Initial velocity

- Initial VX: set a initial X velocity for the object.
- Initial VY: set a initial Y velocity for the object.
- Initial VZ: set a initial Z velocity for the object.
- Initial WX: set a initial WX angular velocity for the object.
- Initial WY: set a initial WY angular velocity for the object.
- Initial WZ: set a initial WZ angular velocity for the object.



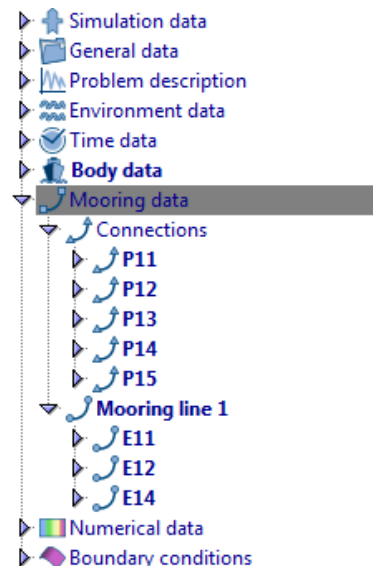
Initial velocity interface

## 3.9. Mooring data

This section is devoted to present the GUI options available in SeaFEM to define a wide variety of mooring systems. All the information presented in this section is complementary to that in sections [Appendix B: Tcl extension](#) and [Appendix E: Mooring definition by Tcl](#) where the definition of mooring systems by using the Tcl extension of SeaFEM is described.

### Mooring lines definition

A mooring system may consist on several independent mooring lines connected to a given body. At the same time, each mooring line may consist on several interconnected mooring segments (also called elements within the GUI). Elements (i.e. mooring segments) are linked at connection points (also called connections within the GUI). The image below shows an example of mooring system definition. In this case, the system consists on a single mooring line composed by three mooring segments (E11, E12, E14). The segments are connected between themselves, but also with the seabed and with the floating body through the connections points (P11, P12, ..., P15).



New mooring lines can be created by right-clicking over the "Mooring data" option of the data tree and selecting the option "Create a new mooring line". By default, when creating a new mooring line it contains a single element.

#### Mooring data ► Create new mooring line

### Elements definition

When creating a new mooring line, a single mooring segment is

created by default and assigned to the current segment. New mooring segments can be created by copying the previous one and editing the corresponding parameters as they are described in what follows:

Mooring element

Type of mooring: Catenary

Length: 0.0 m

Area: 0.0 m<sup>2</sup>


Young modulus: 0.0 Pa


Effective weight: 0.0 N/m



Number of elements: 30

Damping  $\alpha$ : 0.0

Damping  $\beta$ : 0.0

End A:  

End B:  

 OK  Cancel

**Type of mooring:** this parameter determines the type of mooring segment to be used. The possible values of this parameter are:

- Spring : quasi-static elastic bar (spring able to work in both, tension and compression regimes).
- Spring only tension : quasi-static cable (spring able to work only in tension)
- Catenary : quasi-static elastic catenary
- Dynamic cable : dynamic cable

*Length* [m] : length of the current mooring element

$Area [m^2]$ : cross section area of the element

*Young modulus* [Pa] : Young modulus of the current mooring element

*Effective weight* [N/m] : effective weight (actual weight minus bouyancy) per unit length

**End A:** This field is used to specify the first end point of the current mooring segment. It can be specified by either giving the coordinates of a new point, or by selecting an already existing point of the actual geometry. (See the "Connection definition" section below).

**End B:** This field is used to specify the second end point of the current mooring segment. It can be specified by either giving the coordinates of a new point, or by selecting an already existing point of the actual geometry. (See the "Connection definition" section below).

**Number of elements**: this parameter is only enabled when the dynamic FEM cable type of mooring is selected. It determines the number of line elements used in the cable discretization.

*Damping  $a, b$*  : user defined damping ratios for dynamic cables

### Connections definition

As it was shown in the previous section, any mooring segment must be defined by specifying the two end points (End A and End B) of the segment's initial configuration. New connection points can be created in-situ when editing a given mooring element. If the connection point already exists, it will be available from the drop-down list next to the End A and End B entries in the "Mooring element" definition window. The next figures show the connection's definition window that is used to define a new connection point. Specific parameters depend on the actual type of connection point being defined.

GD Connection

Connection definition

Name: Link 1

Type: Connection point

Connection data

Point: 0.723473, 0.0321543, 0 m

Buoy/Weight: 0.0 N

Ok Cancel

**Connection**

Connection definition

Name: Link 1

Type: Fairlead point

Connection data

Body: Body data Auto1

Point: 0.723473, 0.0321543, 0 m

Ok Cancel

The screenshot shows the 'Connection' dialog box with the following details:

- Connection definition:**
  - Name: Link 1
  - Type: Anchor point
- Connection data:**
  - Seabed contact: Frictionless seabed
  - Point: 0.723473, 0.0321543, 0 m

The 'Ok' button is highlighted in blue.

The required parameters and their possible values are

described next.

*Name* : name used to identify the connection point

*Type* : type of connection point.

- Connection point : connection point used to link two mooring segments
- Fairlead point : connection point used to link a mooring segment with a body
- Anchor point : connection point used to link a mooring segment with the seabed

*Point* : coordinates of the connection point

*Buoy/Weight* : buoyancy or weight force to be applied vertically at the connection point. This parameter is available only for connection points between mooring segments

*Body* : it determines to which body the mooring segment is linked to when using the current connection point. This parameter is only available for fairlead connection points

*Seabed contact* : type of interaction between the seabed and the mooring segment that ends at the current connection point. This parameter is only available for anchor connection points, and has effect only when using catenary mooring segments and cables. Possible values for this parameter are as follows:

- Frictionless seabed
- Chain with sandy seabed
- Chain with mud/sand seabed
- Chain with mud/clay seabed
- Wire rope with sandy seabed
- Wire rope with mud/sand seabed
- Wire rope with mud/clay seabed

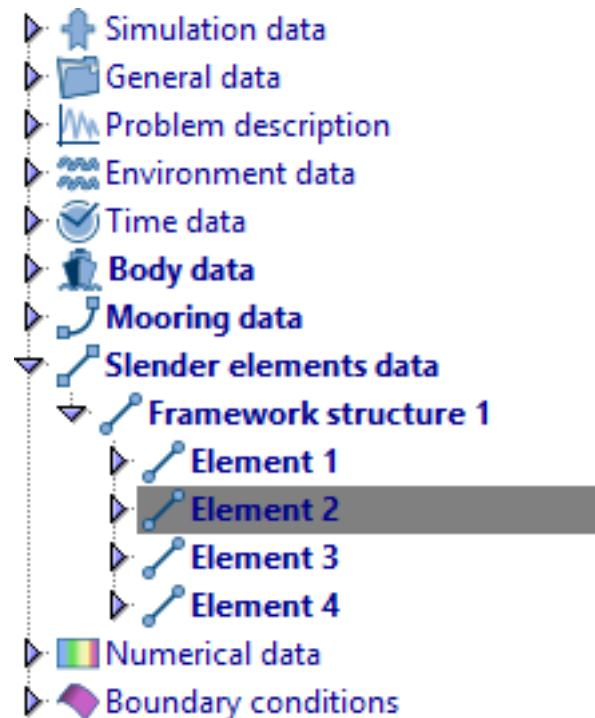
### 3.10. Slender elements data

This section is devoted to present the GUI options available in SeaFEM to define slender elements (i.e. Morison's elements). All the information presented in this section is complementary to that in sections [Appendix B: Tcl extension](#) and [Appendix F: Morison's forces effect](#) where the definition of slender elements by using the Tcl extension of SeaFEM is described.

#### Slender element sets to define framework structures

When viscous effects are anticipated to play a role on the dynamic behavior of an offshore structure, Morison's equation can be used in SeaFEM to introduce force corrections due to such viscous effects. To this aim slender elements must be created and associated to a given body. For the sake of visual organization, such elements can also be grouped in the GUI of SeaFEM forming auxiliary framework structures. Based on the information provided by the user, SeaFEM evaluates Morison's forces per unit length acting on each element, and after integration along the elements, the resultant forces are incorporated to the dynamics of the rigid body system.

The image below shows an example of framework structure definition. In this case, the system consists on a single framework structure composed by four slender elements.



Note that grouping slender elements in element sets has no special meaning and may be used only for organization purposes. But each individual element must be explicitly associated to a given body. New slender element sets can be created by right-clicking over the "Slender elements data" option of the data tree and selecting the option "Create new slender element set". By default, when creating a new set it contains a single element.

#### Slender elements data ► Create new slender elements set

##### Slender elements definition

When creating a new slender elements set, a single slender element is created by default and assigned to the current set. New slender elements can be created by copying the previous one and editing the corresponding parameters as they are described in what follows:

**Element**

Body:  ▼

Diameter:   ▼

Section area:   ▼

Cm:

Cd:

Cv:

Cf:

Cl:

☐ Virtual element

Initial point:  ,  ,  •

End point:  ,  ,  •

**Body:** this parameter identifies the body to which the slender element is going to be connected.

**Diameter:** diameter of the slender element cross section.

**Section area:** total cross section area of the element.

**Cm:** added mass correction coefficient.

**Cd:** non-linear drag coefficient.

**Cv:** linear drag coefficient.

**Cf:** friction coefficient.

**Cl:** lift coefficient.

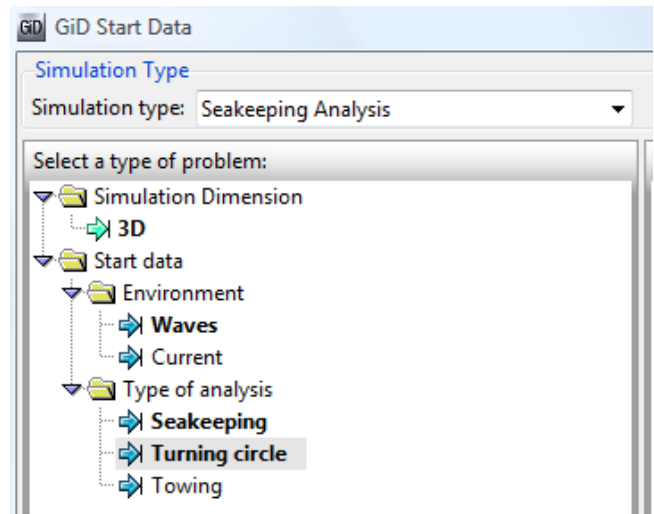
**Virtual element:** this parameter determines if the corresponding slender element is going to be considered a virtual element. If the virtual element option is checked floatability forces over the element are neglected.

**Initial point:** coordinates of the point where the element begins.

**End point:** coordinates of the end point of the element.

### 3.11. Test type

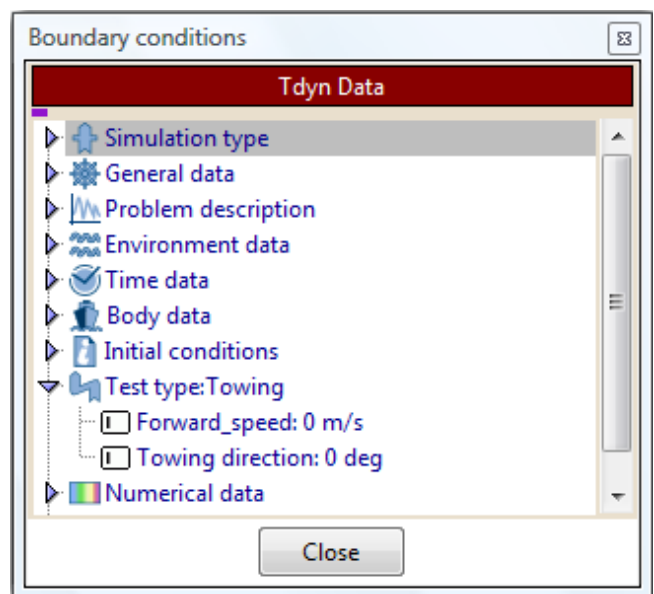
Test type data will be available in the SeaFEM data tree when either 'Turning circle' or 'Towing' type of analysis are selected. Note that these type of analysis are only compatible with the time domain type of problem.

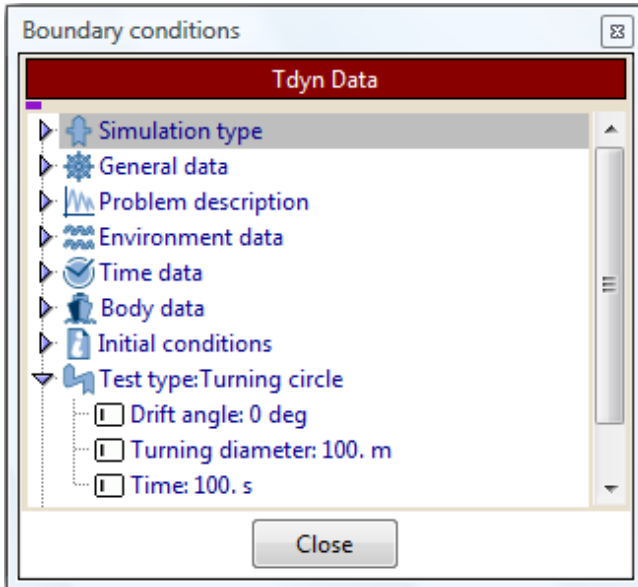


Type of analysis selection in the Start Data window

The type of analysis is also available through the menu:

**General data ► Problem setup ► Type of Analysis**





Turning circle test type options

Towing test type options

Next is a description of the different inputs to be provided by the user depending on the test type being considered.

#### Towing test type:

- **Speed:** towing velocity applied during the test.
- **Direction:** direction along which the towing velocity is applied. It is provided in the form of an angle measured with respect to the X global axis.

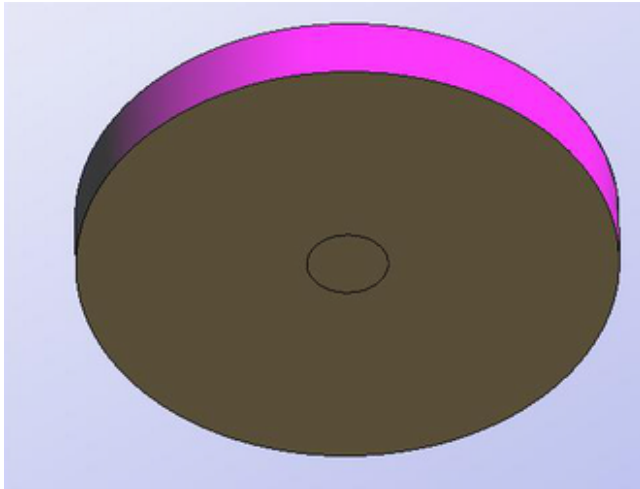
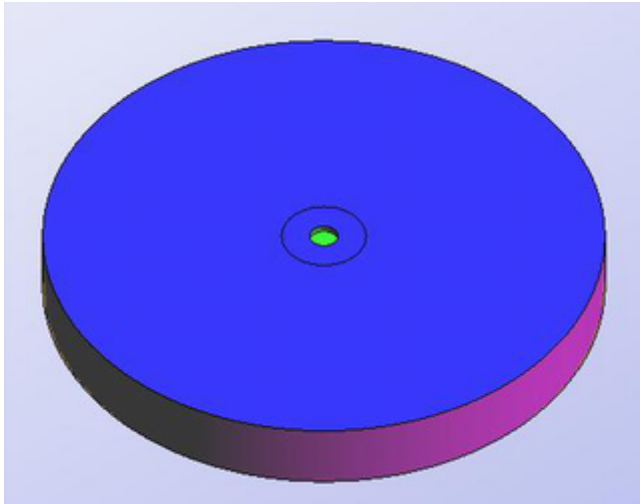
#### Turning circle test type:

- **Drift angle:** maneuver drift. It is provided in the form of an angle measured with respect to the X global axis.
- **Turning diameter:** diameter of the turning circle of the test. It must be a positive number.
- **Time:** time used to complete the maneuver.

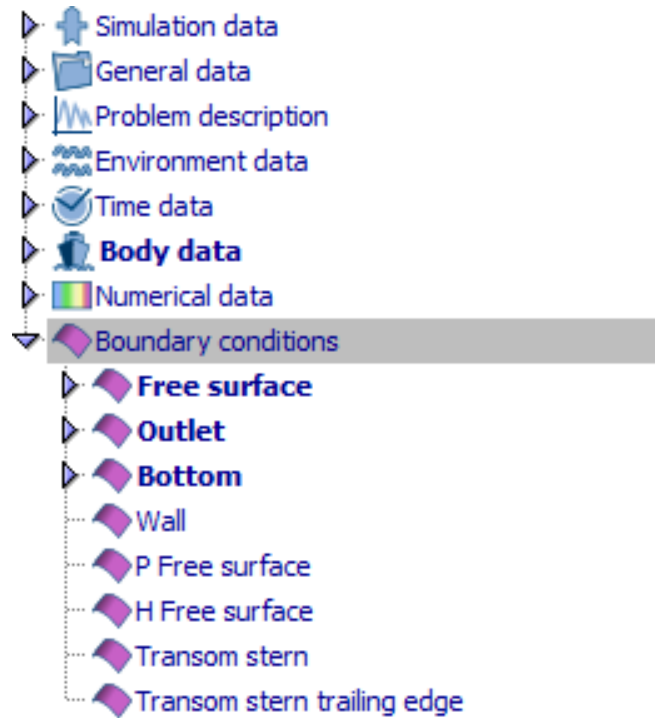


### 3.12. Boundary conditions

For time domain analysis, boundary conditions must be applied on the surfaces surrounding the volume, so the stated problem can be solved. Different type of boundary conditions can be applied, each of them aiming at different purposes. In order to explain each boundary conditions, the case study of a floating cylinder will be used as an example (see Application example section for further information). The following figure shows the geometry for this specific case study:



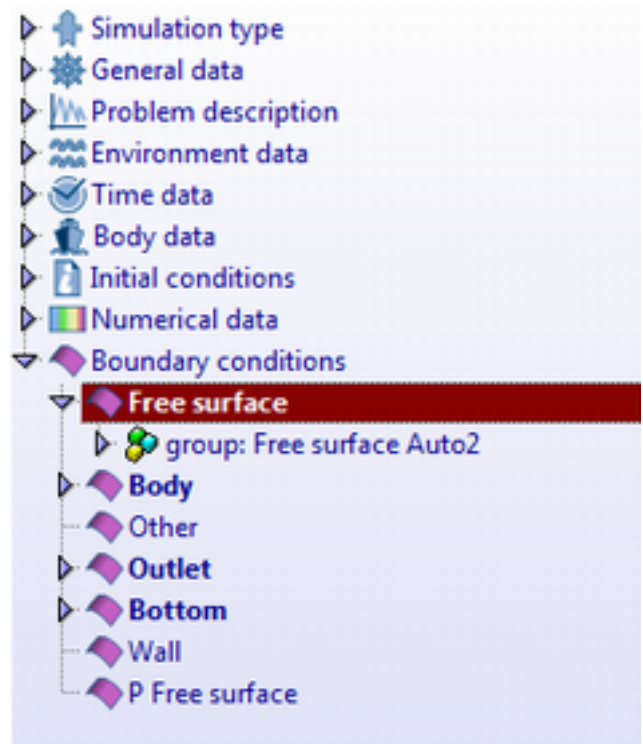
Body geometry and computational domain

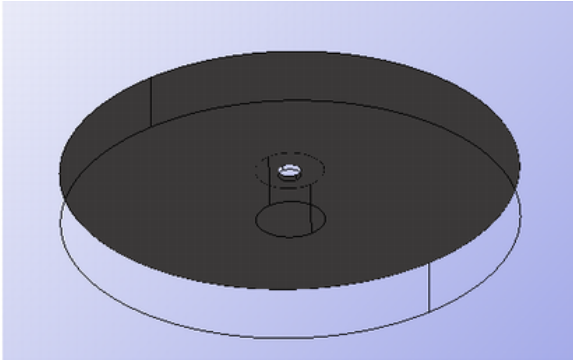


Boundary conditions interface

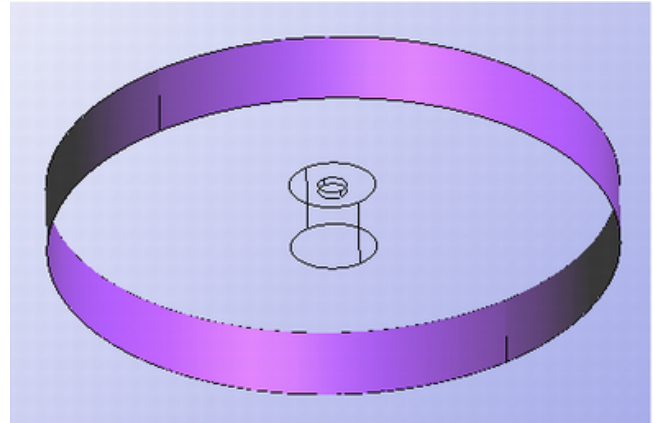
A description of each type of boundary condition is given next:

**Free Surface:** Condition to be applied on the surfaces located at the plane  $z=0$ . On this surfaces, the kinematic and dynamic free surface boundary conditions will be applied.





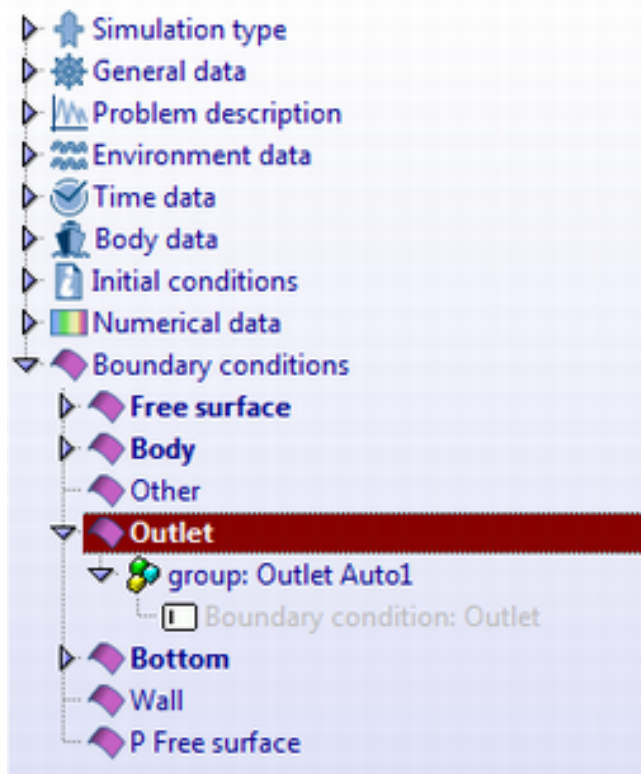
Free surface BC: data interface and geometry where the boundary condition has been applied on



Outlet BC: data interface and geometry where the boundary condition has been applied on

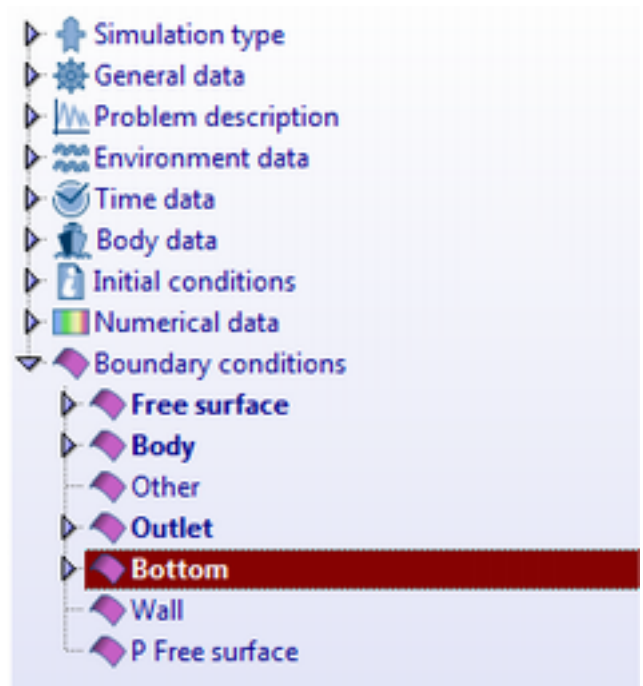
**Outlet:** Condition to be applied to the vertical surfaces bounding the computational domain. It is used to apply two types of condition:

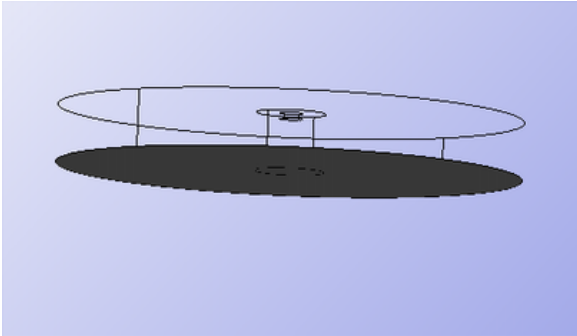
- **Case 1:** this case has neither body translational velocities nor currents. Then the Sommerfeld radiation condition is applied for the longest waves.
- **Case 2:** this case has body translational velocities and/or currents. Then no flow boundary condition for the scattered velocity potential is applied.



**Bottom:** Condition to be applied to the lower horizontal surface bounding the computational domain. It is used to apply two types of conditions:

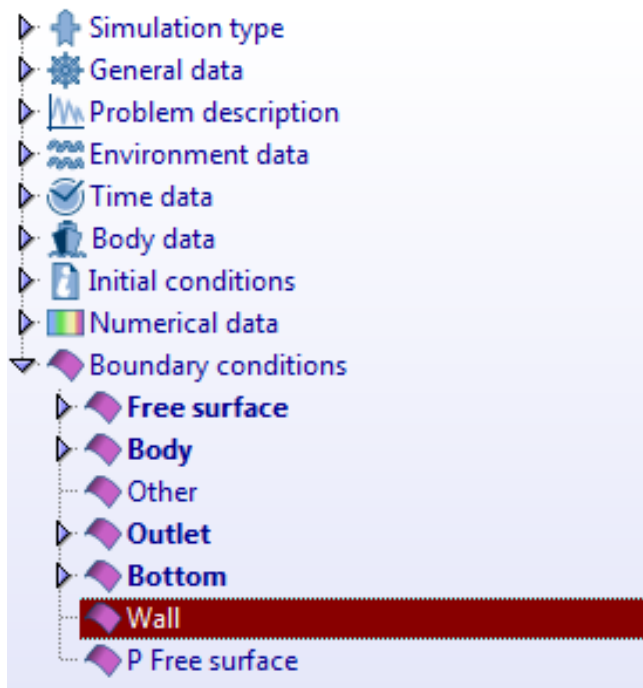
- **Case 1:** this case has neither body translational velocities nor currents. Then a Neumann boundary condition is used to impose a flow through the boundary to match the effect of the actual depth to the depth of the computational domain. For example, this condition can be used for infinite depth simulations and using computational depths smaller than half the wave length of the largest waves. However, care must be taken, because this can be done as long as the computational depth is large enough so that the body will not realize the presence of the computational bottom
- **Case 2:** this case has body translational velocities and/or currents. Then no flow boundary condition for the total velocity potential (incident+scattered potential) is applied. For this case, this condition is equivalent to a *Wall* type boundary condition.





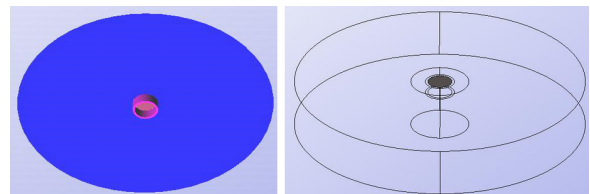
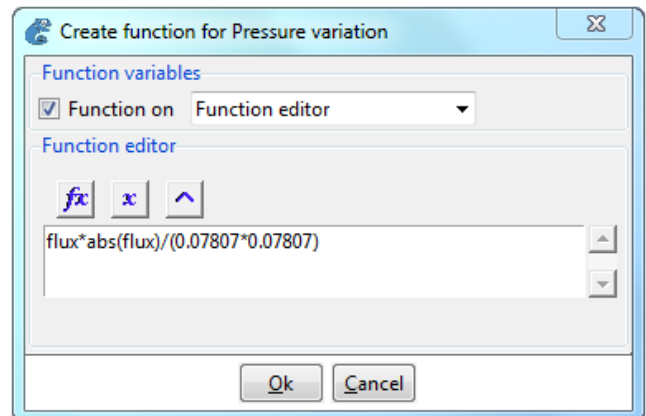
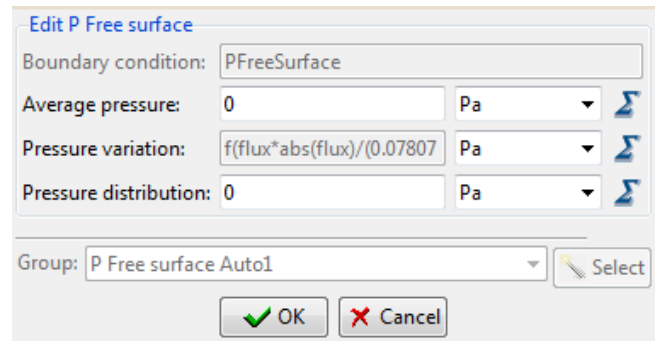
Bottom BC: data interface and geometry where the boundary condition has been applied on

**Wall:** Condition to be applied on fixed surfaces where waves will bounce back. This condition impose a no flow boundary condition for the total velocity potential. It can be applied on non-horizontal surfaces bounding the lower part of the computational domain to simulate irregular sea bottom (then the *Bottom* boundary condition should not be applied).



Wall BC interface

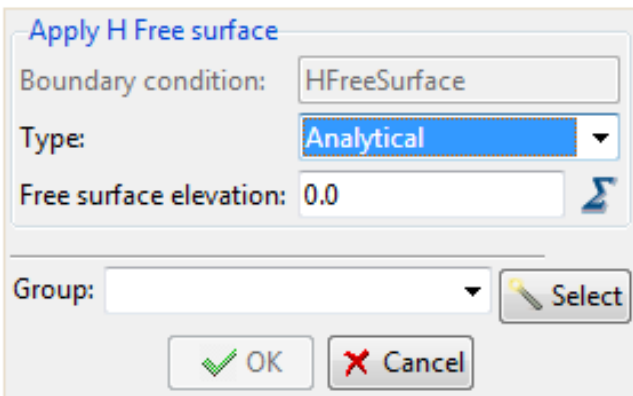
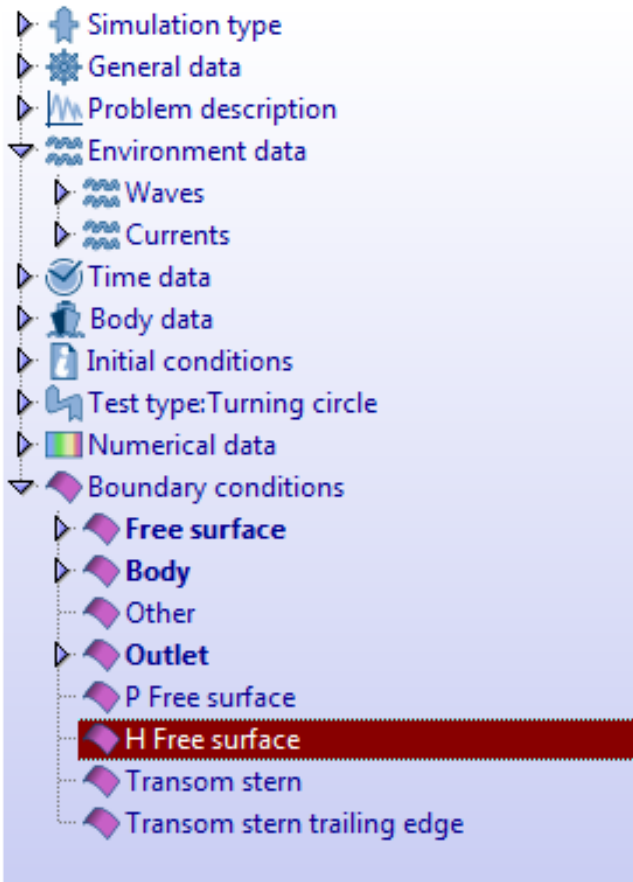
**PfreeSurface:** Condition to be applied on free surfaces where pressure will be applied. Next figure shows an example of a PFreeSurface for a wave energy device based on the oscillating water column principle. This condition is to be applied on surfaces located at the plane  $z=0$ , and the *Free Surface* boundary condition have to be applied as well. On each node over the selected free surfaces, a specific pressure will be applied. This pressure is obtained as:  $P=(P_{average}+P_{variation}(t))\cdot P_{distribution}(x,y)$ , where  $P$  is the pressure to be applied,  $P_{average}$  is an average pressure constant in time and uniform in space,  $P_{variation}(t)$  is a time dependent pressure uniform in space, and  $P_{distribution}(x,y)$  is a pressure distribution in space and constant in time. The formulation for each component of the pressure is introduced via the function editor, whose variables are described in Appendix A.



PFreeSurface BC: data interface and geometry where the boundary condition has been applied

**HfreeSurface:** Condition to be applied on specific free surfaces where the free surface elevation is limited in height (no higher than specified values). This condition is to be applied on surfaces located at the plane  $z=0$ , and the *Free Surface* boundary condition has to be applied as well. The limitation in height can be imposed in two different ways:

- **Analytical:** the elevation limit is introduced via an analytical function using the function editor.
- **Dry body:** the elevation limit is introduced via a three dimensional surface mesh. The vertical projection of this mesh should cover the portion of free surface where the *HfreeSurface* boundary condition has been imposed. This mesh will move following the body movements, and pressure over the vetted portions will be calculated, as well as the resulting forces and moments. These forces and moments can be introduced into the body dynamics via the function editor provided in the external loads section.
- **Tcl script:** The elevation limit is defined in a Tcl script. See [Appendix B: Tcl extension](#) section for further information.

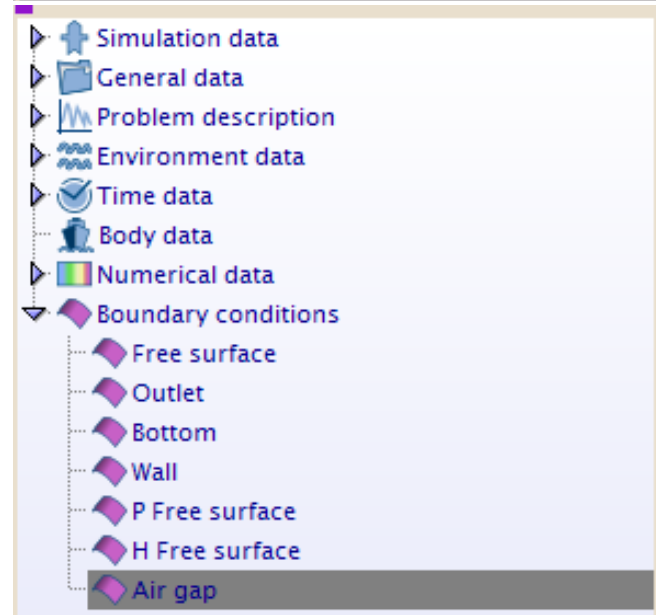
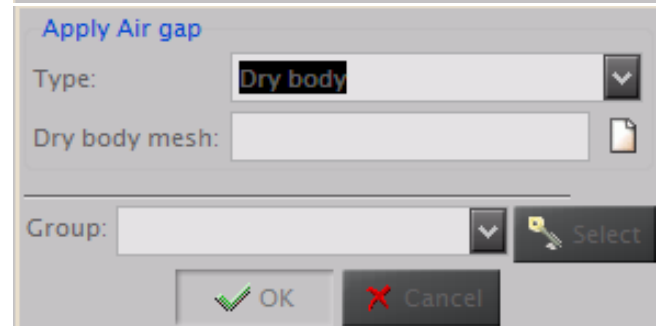
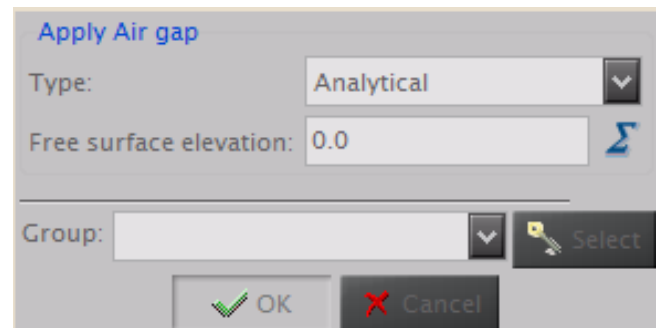


HFreeSurface BC: data interface

**Air gap:** Condition to be applied on specific free surfaces where

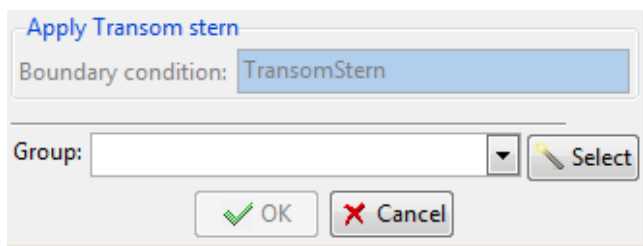
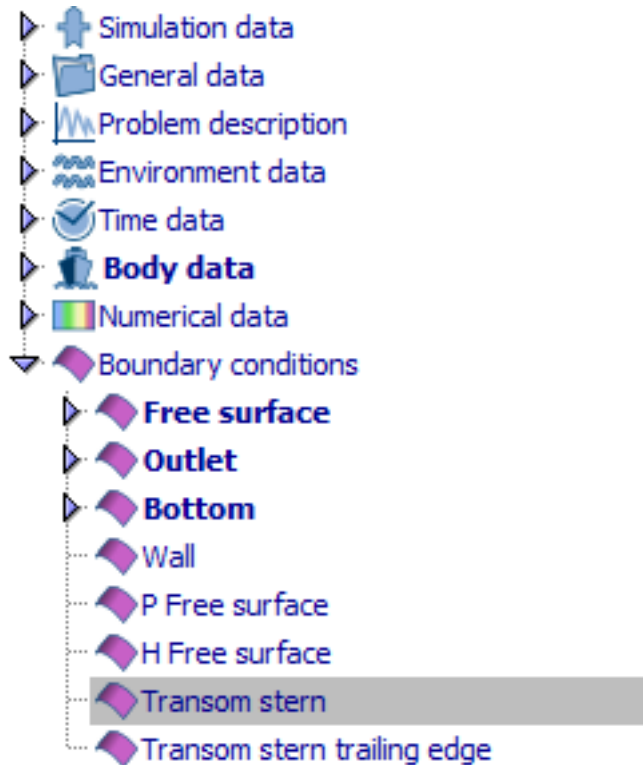
the free surface elevation is limited in height (no higher than specified values), and air gap results needs to be known. This condition is to be applied on surfaces located at the plane  $z=0$ , and the *Free Surface* boundary condition has to be applied as well. The limitation in height can be imposed in two different ways:

- **Analytical:** the elevation limit is introduced via an analytical function using the function editor.
- **Dry body:** the elevation limit is introduced via a three dimensional surface mesh. The vertical projection of this mesh should cover the portion of free surface where the *Air gap* boundary condition has been imposed. This mesh will move following the body movements, and air gap over the vetted portions will be calculated.



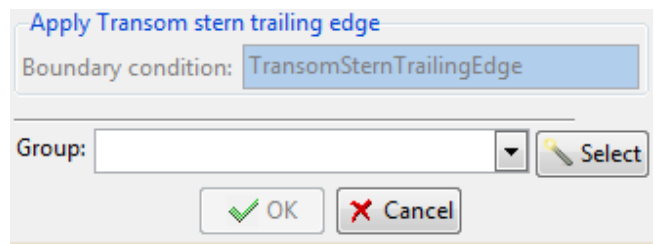
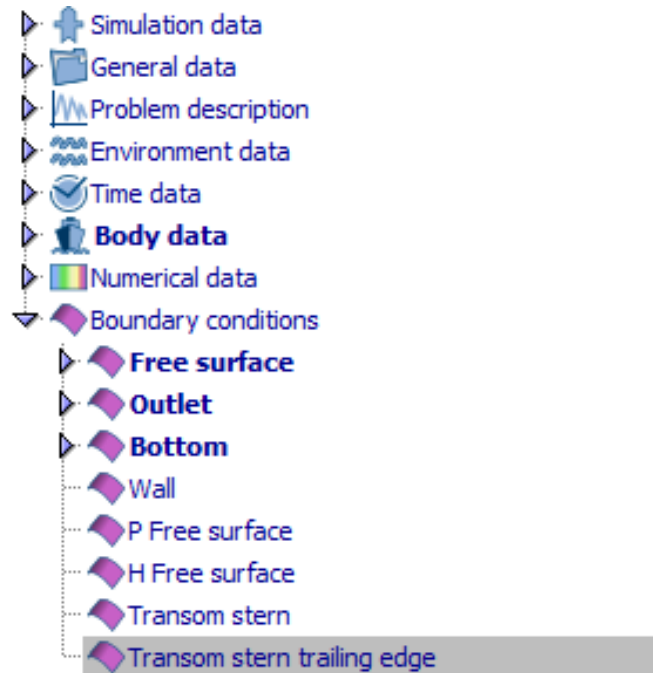
Air gap BC: data interface

**Transom stern:** when considering transom sterns, flow detachment happens at the lower edge of the transom. Since potential flow is incapable of predicting this sort of detachment, a transpiration model is used in SeaFEM to enforce it. To do so, the no flux body boundary condition is no longer applied to the body surfaces where the transom stern condition is applied. On the contrary, a flux is allowed on these surfaces.



Transom stern BC interface

**Transom stern trailing edge:** this condition is used to enforce that the detachment edge belong to the free surface stream surface.

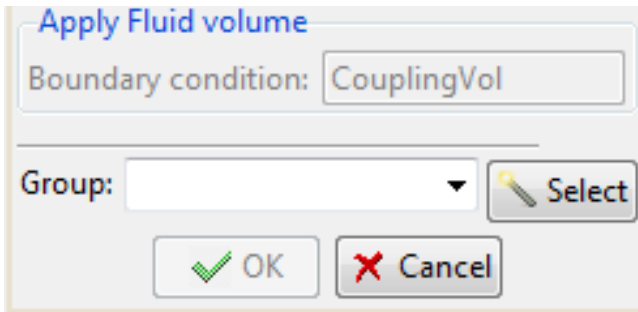


Transom stern trailing edge BC interface

**Fluid volume:** this condition is only available in Coupled Seakeeping-Structural analysis. For this kind of analysis, it is mandatory to apply this boundary condition to the entire volume defining the SeaFEM domain of analysis. Strictly speaking, this is not a boundary condition, but a property of the model that is necessary to identify the seakeeping domain and to differentiate it from the structural mesh.







Fluid volume coupling BC interface

### 3.13. Mesh generation

Mesh requirements are different depending on whether frequency or time domain analysis are used. Frequency domain analysis just require a body boundary mesh consistent on quadrilateral elements. Hence, it usually suffices to provide a global mesh size and mesh transition, since SeaFEM automatically sets the type of element required. Complicated geometries may require additional assignment of local mesh sizes.

On the other hand, in order to generate a good quality mesh for time domain analysis, it is advised to the user to follow these recommendations:

- The mesh size at the body and free surface in the analysis area must be small enough so that the geometry and the smaller wave can be represented accurately. The mesh size should be no larger than one fifth of the smallest wave length. Recommended value, at least, one tenth.
- Mesh size at the outlet should be no larger than one fifth of the distance to the reference point.
- Mesh size at the portion of the bottom located right below the body should be no larger than one fifth of the computational depth. Recommended value is one tenth when the computational depth is at least as large as the maximum wave length. Otherwise, the smaller the computational depth, the smaller the mesh size.
- Mesh size transition should be reasonable smooth around the analysis area.
- Lines should be assigned a mesh size corresponding to the minimum size assigned to the surfaces they belong to.
- Points should be assigned a mesh size corresponding to the minimum size assigned to the lines they belong to.

### 3.14. Executing SeaFEM solver

#### 3.14.1. Automatically executing SeaFEM

SeaFEM solver can be comfortably started through Tdyn pre-processing environment **Calculate** menu. Once the analysed problem is defined (i.e. the geometry is created) the boundary conditions assigned, and the mesh is generated, the **Start** button in the **Calculate** menu (or the Calculate icon) can be pressed.

##### Calculate ► Calculate window

Then the calculation is started and creates a number of output files (see section [Output files](#) for a brief description of the files generated during the execution).

#### 3.14.2. Manually executing SeaFEM

Sometimes it can be interesting to run the Tdyn executable

(tdyn.exe) manually (without using the graphic user interface of the software) in order to run SeaFEM analysis. The necessary steps are described here.

From here in advance, the following notation will be employed for description purposes:

**\$gidpath** : root directory of the installed program. It contains, among others, the gid.exe executable file called to run the GiD custom GUI.

**\$CompassFEM\_version** : CompassFEM problemtype version name.

**\$inputpath** : directory that contains the input data file associated to the SeaFEM model under analysis.

**\$modelname** : name of the input data file.

First, the input data file required by the Tdyn executable must be generated before execution. To this aim, the corresponding SeaFEM model must be loaded to the GiD custom interface.

Next, the input data file must be exported, assuming that the model setup has been finished successfully (applying material properties and boundary conditions) and that the mesh has been already generated. In order to export the input file, the following menu sequence must be used:

##### Files ► Export ► Calculation file...

By doing this, the user will be asked for a file name (\$modelname) and location (\$inputpath). By default, .dat extension will be used to export the input file. If desired, .flavia extension can be also specified for instance, trying to mimic the file name convention used when running SeaFEM solver automatically.

Before execution, you must ensure that the tdy.exe process is able to find a password.txt file containing a valid tdy password. You can create such a text file manually and copying the password inside. Alternatively, the password.txt file can be copied from the directory **\$gidpath\problemtypes\CompassFEM\_version\compassfem.gid**, where it is automatically saved if tdy passwords have been previously registered through the GiD custom GUI. For manual execution of the solver, the password file must be located either next to the tdy.exe executable (this is on the previously mentioned **\$gidpath\problemtypes\CompassFEM\_version\compassfem.gid\exec** directory) or next to the input file.

After exporting the input file and copying a valid password.txt file, everything is ready to launch tdy.exe manually. To this aim, open a command shell and move to the location of the tdy.exe executable. Such a location will be typically of the form:

**\$gidpath\problemtypes\CompassFEM\_version\compassfem.gid\exec**

Note that tdy.exe may be also executed from an arbitrary location if the directory path above is conveniently added to the environment system variable **PATH**.

Finally, launch tdy by using the following command line:

```
>> tdy.exe -name "$inputpath$modelname" -SeaWaves
```

#### 3.14.3. Output files generated during process execution

The output files described in the following section concern to the global analysis.



ProblemName.flavia.inf : Text file containing global information as well as process information for each time step. The content of this file can also be accessed during calculation through the GUI by using the menu option *Calculate > View process info*.

ProblemName.flavia.out : Text file containing iterations and convergence history.

ProblemName.flavia.tim : XML file that contains a timetable in XML format giving information on the CPU time consumption of the process. The timetable contains a report of the execution time used by different parts of the problem. This file is only available after complete successful calculation of a problem.

ProblemName.flavia.err : Text file containing error messages (file created only if tdyn.exe exits with an error).

ProblemName.flavia.res : Main results file that contains all field valued results. When pressing **Postprocess** in *Custom GiD*, this file is loaded, and the results it contains can be visualized in the post-processing module. Also note that each calculation will delete a previous results file that might exist in this directory, unless it has been renamed before the new calculation process has been started.

ProblemName.flavia.ram.msh : Mesh for structural analysis using Ram-Series.

**The output files described in the following section concern to the results of the mooring system.**

MooringResults.msh : Mesh file containing the nodes coordinates of the mooring segments mesh for the initial configuration. For post-process animation, the coordinates of the initial mesh are updated according to the displacement results saved in MooringResults.res during calculation.

ProblemName.MooringData.res : Text file containing time evolution data of the tension force on both ends of each mooring segment.

MooringResults.res : Text file that contains displacement results (components and module) at mooring mesh nodes. This results are necessary for animation of the mooring lines in the postprocess.

**The output files described in the following section concern to time evolution results on bodies.**

Results contained in these files can be visualized using the SeaFEM graphs utility in the SeaFEM postprocessor (see for instance section [Results visualization](#)).

ProblemName.BodyKinematics.res : Text file containing time history data of body movements, velocities and accelerations of all bodies under analysis. The complete set of data (all those kinematic results selected for output in the data tree) is written first for the first body defined in the GUI. The rest of bodies data results are written following the order the bodies are defined in the GUI.

ProblemName.BodyLoads.res : Text file containing time history data of body loads and moments acting over all bodies under analysis. The complete set of data (all those kinematic results selected for output in the data tree) is written first for the first body defined in the GUI. The rest of bodies data results are written following the order the bodies are defined in the GUI. Different components of forces and moments are written separately. Hence, total forces, reactions, hydrostatic pressure forces and dynamic pressure forces can be assessed separately.

ProblemName.Outputs.res : Text file containing time evolution data of the user defined results. (Up to 10 custom results can be

defined by the user in the *User defined* data tree entry).

MooringLoads.res : Text file containing time evolution data of the mooring loads acting on bodies.

**The output files described in the following section concern the body animation results.**

Results contained in these files can be used to setup body animations within the postprocessor (see for instance section [Results visualization](#)).

ProblemName.BodyMovements\_animations.res : Text file containing time history data of the movements of the first body defined in the GUI of SeaFEM (referred to the global frame of reference located at the origin 0,0,0).

BodyMovements\_animations.#.res : # stands for an integer index that identifies the body to which the file under consideration refers. These are text files similar to the previous one and concerning the remaining bodies defined in the analysis.

### 3.15. Body load results

Various load components acting on each body can be plot within the SeaFEM postprocessor. To this aim, the following menu option can be used.

#### Postprocess ► SeaFEM Graphs

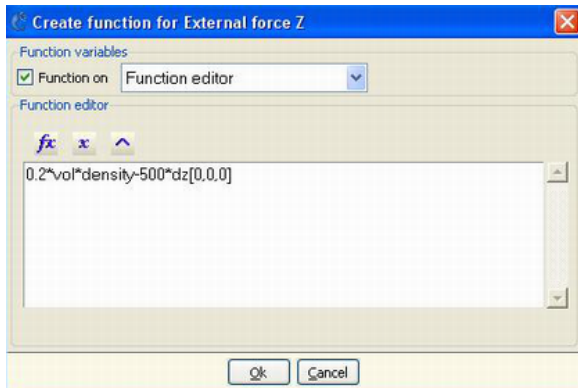
Once the graphs window appears, the 'Body Loads' option must be selected from the results type combobox. By doing this, a separate tab will appear for each body the user has defined, which will contain the various load components acting on the body at hand. Available load components read as follows:

- **Total:** resultant of all external forces acting on the body. Usually, this is the resultant from adding the hydrostatic and hydrodynamic pressure loads, mooring loads, the gravity effect and any external forces defined by the user. In other special cases, such a total load may also contain the forces and moments due to slender (Morison) elements (if defined by the user), and the effect of additional added mass, stiffness and damping matrices in the case these matrices are also provided by the user through the GUI interface of SeaFEM. Finally, if a damping resonance factor is used (see numerical data for details), the corresponding damping load will be also added to the total load.
- **Reactions:** these are the body reactions evaluated as  $\mathbf{R} = \mathbf{M} \cdot \mathbf{a} - \mathbf{F}_t$  where  $\mathbf{M}$  is the body mass matrix,  $\mathbf{a}$  is the vector of linear and angular accelerations and  $\mathbf{F}_t$  is the vector of total forces and moments defined above. Note that in the case the reactions are due to the forces and moments from other bodies transmitted through the body links, the reactions as defined here are the resultant of all body links attached to the body under consideration. Detailed information on body link reactions is provided in a separate file where body loads are reported individually for each body link instead of the resultant provided here.
- **Hydrostatic:** these are the hydrostatic pressure loads evaluated using the hydrostatic restoring matrix, if linear hydrostatics is used, or by actual integration of the hydrostatic pressure over the auxiliary body mesh if the non-linear hydrostatics option is selected.
- **Dynamic:** this are the dynamic pressure loads obtained by integration over the body surface of the dynamic pressure obtained when solving the waves (diffraction-radiation) problem.

Mooring: forces acting over the body due to the action of the mooring lines attached to the body under consideration.

### 3.16. Appendix A: function editor

The function editor is a flexible tool to bring dependent values, external to the code, into the computations. Next figure shows an example of the use of the function editor.



Function editor interface

The following operators can be used in the definition of functions:

#### Basic operators:

- **+** : adding operator.
- Syntax: [adding\_expression] + [adding\_expression].
- **-** : subtraction operator.
- Syntax: [subtraction\_expression] - [subtraction\_expression].
- **^** : exponent operator.
- Syntax: [exponent\_expression] ^ [function\_expression].
- **\*** : multiplicative operator.
- Syntax: [multiplicative\_expression] \* [multiplicative\_expression].
- **/** : division operator.
- Syntax: [multiplicative\_expression] / [quotient\_expression].
- **div** : integer division operator  $\text{int}(x/y+0.5)$ .
- Syntax: ([multiplicative\_expression]) div ([quotient\_expression]). Example: (x)div(2+y).
- **idiv** : integer division operator  $\text{int}(x/y+0.5)$ . Similar to div operator but with different syntax.
- Syntax: idiv ([multiplicative\_expression], [quotient\_expression]). Example: idiv(x,2+y).
- **mod** : integer division module operator  $\text{int}(x+0.5)\%\text{int}(y+0.5)$ .
- Syntax: ([multiplicative\_expression]) mod ([quotient\_expression]). Example: (t)mod(2).
- **imod** : integer division module operator  $\text{int}(x+0.5)\%\text{int}(y+0.5)$ . Similar to mod operator but with different syntax.
- Syntax: imod ([multiplicative\_expression],[quotient\_expression]). Example: imod(t,2).
- **rdiv** : real division operator  $\text{int}(x/y)$ .
- Syntax: ([multiplicative\_expression]) rdiv ([quotient\_expression]). Example: (t)rdiv(5).

**ddiv** : real division operator  $\text{int}(x/y)$ . Similar to rdiv operator but with different syntax.

- Syntax: ddiv ([multiplicative\_expression], [quotient\_expression]). Example: ddiv(t,5).
- **rmod** : real division module operator  $x/y-\text{int}(x/y)$ .
- Syntax: ([multiplicative\_expression]) rmod ([quotient\_expression]). Example: (t)rmod(5).
- **dmod** : real division module operator  $x/y-\text{int}(x/y)$ . Similar to rmod operator but with different syntax.
- Syntax: dmod ([multiplicative\_expression], [quotient\_expression]). Example: dmod(t,5).
- **max** : maximum operator.
- Syntax: max ([expression], [expression]). Example: max(x,y).
- **min** : minimum operator.
- Syntax: min ([expression], [expression]). Example: min(x,y).
- **not** : not operator.
- Syntax: not([function\_expression]).
- **~** : not operator.
- Syntax: ~([function\_expression]).

#### Examples:

```
(2*dy)
(5*(dy+1))/2
(dz*dy)mod(5)
imod(dz*dy) (5)
(5^4)
```

#### Relational operators:

The relational (binary) operators compare their first operand with their second operand to test validity of the specified relationship. The result of the relational expression is 1 if the tested relationship is true and 0 if it is false. The binary operators that can be used for functions definitions are:

- **<** : less than operator.
- Syntax: [expression] < [expression].
- **<=** : less or equal than operator.
- Syntax: [expression] <= [expression].
- **>=** : greater or equal than operator.
- Syntax: [expression] >= [expression].
- **>** : greater than operator.
- Syntax: [expression] > [expression].
- **=** : equal operator.
- Syntax: [expression] = [expression].
- **~=** : not equal operator.
- Syntax: [expression] != [expression].
- **&** : and operator.
- Syntax: [expression] & [expression].
- **|** : and operator.
- Syntax: [expression] | [expression].

#### Examples:

```
(dy>2)
(dx<=1)
(dx!=1)
```

(dy>2)&(dx>2)&(dx<3)&(dy<3)

• *if-else statement:*

The if statement controls conditional branching. The body of the if statement (elif\_expression) is executed if the value of the expression is non zero. The syntax for the if statement is the following:

```
if(expression)then(elif_expression)else(next_expression)endif
```

being elif\_expression an additional expression that may include an elif clause with next form:

```
(expression2)elif(elif_expression2)then(next_expression2)
```

*Examples:*

```
if(dy>2)then(if(x<1)then(1)else(0)endif)else(0)endif
if(dy>2)then(1)elif(dx<1)then(2)else(0)endif
```

*Function operators:*

The function operators calculate the value of a standard function at the point defined by the given argument. The function operators that can be used for the definition of the functions are:

- *sqrt* : the sqrt function calculates the square root of the argument. Syntax: sqrt(.)
- *abs* : the abs function calculates the absolute value of the argument. Syntax: abs(.)
- *ln* : logarithm of the argument, e base. Syntax: ln(.)
- *log* : logarithm of the argument, decimal base. Syntax: log(.)
- *fac* : factorial of the argument. Syntax: fac(.)
- *sin* : sine of the argument. Syntax: sin(.) (argument given in radians).
- *cos* : cosine of the argument. Syntax: cos(.) (argument given in radians).
- *tan* : tangent of the argument. Syntax: tan(.) (argument given in radians).
- *asin* : The asin function returns the arcsine of the argument in the range  $-\pi/2$  to  $\pi/2$  radians. Syntax: asin(.)
- *acos* : The acos function returns the arccosine of the argument in the range 0 to  $\pi$  radians. Syntax: acos(.)
- *atan* : The atan function returns the arctangent of the argument in the range  $-\pi/2$  to  $\pi/2$  radians. Syntax: atan(.) (result given in radians).
- *sinh* : hyperbolic sine of the argument. Syntax: sinh(.)
- *cosh* : hyperbolic cosine of the argument. Syntax: cosh(.)
- *tanh* : hyperbolic tangent of the argument. Syntax: tanh(.)
- *exp* : the exp function calculates the exponential value of the argument. Syntax: exp(.)
- *heaviside* : the heaviside function evaluates  $H_s$  defined as:

$$H_s(\Phi)=0 \quad \Phi < -\epsilon$$

$$H_s(\Phi)=1/2 \quad (1+\Phi/\epsilon+1/\pi \sin(\pi*\Phi/\epsilon)) \mid \Phi| < \epsilon$$

$$H_s(\Phi)=1 \quad \Phi > \epsilon$$

The syntax of the function is heaviside(.,.), where the first argument is  $\Phi$  and the second  $\epsilon$ .

- *Interpolate* : performs a linear interpolation, based on the given data. Two arguments are required: a list of pairs ( $\xi, \eta$ ), defining a polylineal curve, and a function defining the point ( $\xi$ ) where the evaluation is to be done. Syntax:

```
interpolate(# $\xi_1, \eta_1, \xi_2, \eta_2, \xi_3, \eta_3, \dots, \#$ ).
```

- *InterpolateSpline* : performs a spline interpolation, based on the given data. Two arguments are required: a list of pairs ( $\xi, \eta$ ), defining a the curve, and a function defining the point ( $\xi$ ) where the evaluation is to be done. Syntax: interpolatespline(# $\xi_1, \eta_1, \xi_2, \eta_2, \xi_3, \eta_3, \dots, \#$ ).
- *InterpolateFile* : performs a spline interpolation, based on the data given in a file. Two arguments are required: a file name where a list of pairs ( $\xi, \eta$ ), defining a the curve, is given, and a function defining the the point ( $\xi$ ) where the evaluation is to be done. Syntax: Interpolatefile(., .), where the first argument in the filename, and the second a function defining the value.
- *rand* : The rand function returns a pseudorandom integer in the range 0 to 1, based on the argument given as seed. Syntax: rand(.).
- *int* : Integer conversos. Syntax: int(.).
- *-* : change sign operator. Syntax: (-expression).
- *j0* : Calculates Bessel function of first kind and order 0, at the given point. Syntax: j0(.).
- *j1* : Calculates Bessel function of first kind and order 1, at the given point. Syntax: j1(.).
- *jn* : Calculates Bessel function of first kind and order n, at the given point. Syntax: jn(.,.), where the first argument is the evaluation point and the second is the order of the Bessel function.
- *y0* : Calculates Bessel function of second kind and order 0, at the given point. Syntax: y0(.).
- *y1* : Calculates Bessel function of second kind and order 1, at the given point. Syntax: y1(.).
- *yn* : Calculates Bessel function of second kind and order n, at the given point. Syntax: yn(.,.), where the first argument is the evaluation point and the second is the order of the Bessel function.
- *Readfile* : Execute the function in the ASCII file defined by the argument. The file must include a first line defining the maximum time to use the function and a second line containing the function to be executed. If the current time is greater than the one defined in the file, the execution is paused until the file is updated.

Syntax readfile(.) where the argument is the path and name of the file. Example readfile(C:\Temp\velx.dat). Example of file format:

```
Time = 0.1;
Function = "interpolate(#0.0,1.1,1.0,2.0#t);";
```

- *Tcl* : Executes a TCL script or procedure returning a double value.

Syntax tcl(.) where the argument is the script to be executed. Example tcl(set var) return the value of tcl variable var.

Note: In order to use this function, TCL extension must be enabled by activating Tcl extension.

- *CloudOfDataFile* : performs a local interpolation based on the cloud of points (x,y,z) and data (.) given in a file. The argument is the path and name of the text file. Syntax: CloudOfDataFile(.,) where the argument in the filename. Example CloudOfDataFile(C:\Temp\velx.dat). Example of file format:

```
0.0 0.0 0.0 1.0
0.0 1.0 0.0 0.5
1.0 1.0 1.0 2.0
```

5.0 2.5 2.0 5.0

Examples:

2\*sqrt(dy)

dx\*fac(5)

srand(0)

log(abs(dx))

exp(5)

interpolate(#1.0,2.0,2.0,2.5,3.0,2.0#t^2)

*Specific variables:*

Furthermore, the following variables can be used in the definition of functions. Note that some of these variables refer to the main body (index 1). In order to access the corresponding variable for other bodies, the index of the body must be given in parentheses ( examples: dx(2)[0,0,0], vx(3) ).

#### GENERAL VARIABLES

- time or t : Process time (unit: s).
- gravity or g : Magnitud of gravity (unit: m/s<sup>2</sup>).
- density : Fluid density (unit: kg/m·s<sup>2</sup>).
- ts or dt : time increment (unit: s).
- Init\_factor : returns the current initialization factor (if the initialization time option is used)

#### WAVE SPECTRUM VARIABLES

- Hs : Significant wave height (unit: m).
- Tm : Mean wave period (unit: s).
- W\_s : Wave frequency (unit: s<sup>-1</sup>)
- Gamma\_s : Wave direction (unit: °).

#### MESH GENERAL VARIABLES

These variables may be employed in the definition of boundary conditions varying in space thus resulting in functions that need to be evaluated at every boundary mesh node (for example to define the pressure distribution field within the pressurized free surface boundary condition).

- x, y, z : coordinates of mesh points.

#### FREE SURFACE VARIABLES

- eta[idNode] : Free surface elevation at the specified node
- eta\_prev[idNode] : Free surface elevation at the specified node (previous step)
- x[idNode] : x coordinate at the specified node idNode
- y[idNode] : y coordinate at the specified node idNode

#### BODY's VARIABLES

- vol(idx) or volume(idx) : Water volume displaced by body idx (unit: m<sup>3</sup>).
- disp : Mass of water displaced by body (unit: kg).
- Wet\_surface : Area of body surface under still water level (unit: m<sup>2</sup>).
- xc(idx) : X coordinate of the buoyancy center of body idx (unit: m).
- yc(idx) : Y coordinate of the buoyancy center of body idx (unit: m).
- zc(idx) : Z coordinate of the buoyancy center of body idx (unit: m).

xg(idx) : X coordinate of the buoyancy center of body idx (unit: m).

- yg(idx) : Y coordinate of the buoyancy center of body idx (unit: m).
- zg(idx) : Z coordinate of the buoyancy center of body idx (unit: m).
- k11(idx) : Surge hydrostatic restoring coefficient of body idx. (unit: N/m).
- k22(idx) : Sway hydrostatic restoring coefficient of body idx. (unit: N/m).
- k33(idx) : Heave hydrostatic restrng coefficient of body idx. (unit: N/m).
- k44(idx) : Roll hydrostatic restoring coefficient of body idx. (unit: N/rad).
- k55(idx) : Pitch hydrostatic restoring coefficient of body idx. (unit: N/rad).
- k66(idx) : Yaw hydrostatic restoring coefficient of body idx. (unit: N/rad).
- mass(idx) : Mass of body idx. (unit: kg).
- Ixx(idx) : Inertial moment of body idx respect to X direction and gravity center (unit: Kg·m<sup>2</sup>).
- Iyy(idx) : Inertial moment of body idx respect to Y direction and gravity center. (unit: Kg·m<sup>2</sup>).
- Izz(idx) : Inertial moment of body idx respect to Z direction and gravity center. (unit: Kg·m<sup>2</sup>).
- dx : Translation, in the x direction, of the main body gravity center. (unit: m).
- dy : Translation, in the y direction, of the main body gravity center. (unit: m).
- dz : Translation, in the z direction, of the main body gravity center. (unit: m).
- dx(bodyIdx,stepIdx) : Returns the translation along the x direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m).
- dy(bodyIdx,stepIdx) : Returns the translation along the y direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m).
- dz(bodyIdx,stepIdx) : Returns the translation along the z direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m).
- dx[xi ,yi ,zi]: Translation, in the x direction, of an arbitrary point with coordinates [xi ,yi ,zi] respect to the gravity center. (unit: m).
- dy[xi ,yi ,zi]: Translation, in the y direction, of an arbitrary point with coordinates [xi ,yi ,zi] respect to the gravity center. (unit: m).
- dz[xi ,yi ,zi]: Translation, in the z direction, of an arbitrary point with coordinates [xi ,yi ,zi] respect to the gravity center. (unit: m).
- ldsx : Translation, in the x direction in the local frame. (unit: m).
- ldsy : Translation, in the y direction in the local frame. (unit: m).
- ldsz : Translation, in the z direction in the local frame. (unit: m).
- rx : Rotation, around the x direction, of the body gravity center. (unit: rad).
- ry : Rotation, around the y direction, of the body gravity center. (unit: rad).

rz : Rotation, around the z direction, of the body gravity center. (unit: rad).

- rx(body,stepIdx) : Returns the rotation around the x direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad).
- ry(body,stepIdx) : Returns the rotation around the y direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad).
- rz(body,stepIdx) : Returns the rotation around the z direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad).
- lrx : Rotation, around the x direction in the local frame. (unit: rad).
- lry : Rotation, around the y direction in the local frame. (unit: rad).
- lrz : Rotation, around the z direction in the local frame. (unit: rad).
- vx : Body velocity in the x direction. (unit: m/s).
- vy : Body velocity in the y direction. (unit: m/s).
- vz : Body velocity in the z direction. (unit: m/s).
- vx(body,stepIdx) : Returns the velocity in the x direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m/s).
- vy(body,stepIdx) : Returns the velocity in the y direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m/s).
- vz(body,stepIdx) : Returns the velocity in the z direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m/s).
- wx : Body angular velocity around the x direction. (unit: rad/s).
- wy : Body angular velocity around the y direction. (unit: rad/s).
- wz : Body angular velocity around the z direction. (unit: rad/s).
- wx(body,stepIdx) : Returns the angular velocity around the x direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad/s).
- wy(body,stepIdx) : Returns the angular velocity around the y direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad/s).
- wz(body,stepIdx) : Returns the angular velocity around the z direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad/s).
- lvx : Body velocity in the x direction in the local frame. (unit: m/s).
- lvy : Body velocity in the y direction in the local frame. (unit: m/s).
- lvz : Body velocity in the z direction in the local frame. (unit: m/s).
- lwx : Body angular velocity around the x direction in the local frame. (unit: rad/s).
- lwy : Body angular velocity around the y direction in the local frame. (unit: rad/s).
- lwz : Body angular velocity around the z direction in the local frame. (unit: rad/s).
- ax : Body acceleration in the x direction. (unit: m/s<sup>2</sup>).
- ay : Body acceleration in the y direction. (unit: m/s<sup>2</sup>).
- az : Body acceleration in the z direction. (unit: m/s<sup>2</sup>).

ax(bodyIdx,stepIdx) : Returns the acceleration in the x direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m/s<sup>2</sup>).

- ay(bodyIdx,stepIdx) : Returns the acceleration in the y direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m/s<sup>2</sup>).
- az(bodyIdx,stepIdx) : Returns the acceleration in the z direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: m/s<sup>2</sup>).
- lax : Body acceleration in the x direction in the local frame. (unit: m/s<sup>2</sup>).
- lay : Body acceleration in the y direction in the local frame. (unit: m/s<sup>2</sup>).
- laz : Body acceleration in the z direction in the local frame. (unit: m/s<sup>2</sup>).
- arx : Body angular acceleration around the x direction. (unit: rad/s<sup>2</sup>).
- ary : Body angular acceleration around the y direction. (unit: rad/s<sup>2</sup>).
- arz : Body angular acceleration around the z direction. (unit: rad/s<sup>2</sup>).
- arx(body,stepIdx) : Returns the angular acceleration around the x direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad/s<sup>2</sup>).
- ary(body,stepIdx) : Returns the angular acceleration around the y direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad/s<sup>2</sup>).
- arz(body,stepIdx) : Returns the angular acceleration around the z direction of the body given by *bodyIdx* at a previous time step given by *stepIdx*. (unit: rad/s<sup>2</sup>).
- larx : Body angular acceleration around the x direction in the local frame. (unit: rad/s<sup>2</sup>).
- lary : Body angular acceleration around the y direction in the local frame. (unit: rad/s<sup>2</sup>).
- larz : Body angular acceleration around the z direction in the local frame. (unit: rad/s<sup>2</sup>).
- HPFx\_body : X component of the force due to the hydrostatic pressure over the body. (unit: N).
- HPFy\_body : Y component of the force due to the hydrostatic pressure over the body. (unit: N).
- HPFz\_body : Z component of the force due to the hydrostatic pressure over the body. (unit: N).
- HPMx\_body : X component of the moment due to the hydrostatic pressure over the body. (unit: N·m).
- HPMy\_body : Y component of the moment due to the hydrostatic pressure over the body. (unit: N·m).
- HPMz\_body : Z component of the moment due to the hydrostatic pressure over the body. (unit: N·m).
- PFx\_body : X component of the force due to dynamic pressure over body. (unit: N).
- PFy\_body : Y component of the force due to dynamic pressure over body. (unit: N).
- PFz\_body : Z component of the force due to dynamic pressure over body. (unit: N).
- PMx\_body : X component of the moment due to dynamic pressure over body. (unit: N·m).
- PMy\_body : Y component of the moment due to dynamic pressure over body. (unit: N·m).



pressure over body. (unit: N·m).

- **PMz\_body** : Z component of the moment due to dynamic pressure over body. (unit: N·m).
- **TFx\_body** : X component of the total force over body. (unit: N).
- **TFy\_body** : Y component of the total force over body. (unit: N).
- **TFz\_body** : Z component of the total force over body. (unit: N).
- **TMx\_body** : X component of the total moment over body. (unit: N·m).
- **TMy\_body** : Y component of the total moment over body. (unit: N·m).
- **TMz\_body** : Z component of the total moment over body. (unit: N·m).
- **Fx\_drift** : X component of the drift load due up to first order terms over body. (unit: N).
- **Fy\_drift** : Y component of the drift load due up to first order terms over body. (unit: N).
- **Fz\_drift** : Z component of the drift load due up to first order terms over body. (unit: N).
- **Mx\_drift** : X component of the drift moment due up to first order terms over body. (unit: N·m).
- **My\_drift** : Y component of the drift moment due up to first order terms over body. (unit: N·m).
- **Mz\_drift** : Z component of the drift moment due up to first order terms over body. (unit: N·m).
- **Fx\_Pdrift** : X component of the drift load due up to first order terms over body. (unit: N).
- **Fy\_Pdrift** : Y component of the drift load due up to first order terms over body. (unit: N).
- **Fz\_Pdrift** : Z component of the drift load due up to first order terms over body. (unit: N).
- **Mx\_Pdrift** : X component of the drift moment due up to first order terms over body. (unit: N·m).
- **My\_Pdrift** : Y component of the drift moment due up to first order terms over body. (unit: N·m).
- **Mz\_Pdrift** : Z component of the drift moment due up to first order terms over body. (unit: N·m).
- **RFx\_body** : X component of the reaction loads over body. (unit: N).
- **RFy\_body** : Y component of the reaction loads over body. (unit: N).
- **RFz\_body** : Z component of the reaction loads over body. (unit: N).
- **RMx\_body** : X component of the reaction moments over body. (unit: N·m).
- **RMx\_body** : Y component of the reaction moments over body. (unit: N·m).
- **RMz\_body** : Z component of the reaction moments over body. (unit: N·m).

#### Deprecated variables

- **dx\_prev** : Return body's displacement (previous step)
- **dy\_prev** : Return body's displacement (previous step)
- **dz\_prev** : Return body's displacement (previous step)
- **rx\_prev** : Rotation, around the x direction, of the body gravity center (previous step). (unit: rad).

**ry\_prev** : Rotation, around the x direction, of the body gravity center (previous step). (unit: rad).

- **rz\_prev** : Rotation, around the x direction, of the body gravity center (previous step). (unit: rad).
- **vx\_prev** : Body velocity in the x direction (previous step). (unit: m/s).
- **vy\_prev** : Body velocity in the y direction (previous step). (unit: m/s).
- **vz\_prev** : Body velocity in the z direction (previous step). (unit: m/s).
- **wx\_prev** : Body angular velocity around the x direction (previous step). (unit: rad/s).
- **wy\_prev** : Body angular velocity around the y direction (previous step). (unit: rad/s).
- **wz\_prev** : Body angular velocity around the z direction (previous step). (unit: rad/s).
- **ax\_prev** : Body acceleration in the x direction (previous step). (unit: m/s<sup>2</sup>).
- **ay\_prev** : Body acceleration in the y direction (previous step). (unit: m/s<sup>2</sup>).
- **az\_prev** : Body acceleration in the z direction (previous step). (unit: m/s<sup>2</sup>).
- **arx\_prev** : Body angular acceleration around the x direction in the local frame. (unit: rad/s<sup>2</sup>).
- **ary\_prev** : Body angular acceleration around the y direction in the local frame. (unit: rad/s<sup>2</sup>).
- **arz\_prev** : Body angular acceleration around the z direction in the local frame. (unit: rad/s<sup>2</sup>).

#### PRESSURIZED FREE SURFACE VARIABLES

- **Flux** : Vertical airflow displaced by pressurized free surfaces. (unit: m<sup>3</sup>/s).
- **Flux(Idx,stepIdx)** : Returns the vertical airflow displaced by the pressurized free surfaces indicated by Idx at a previous time step indicated by stepIdx. (unit: m<sup>3</sup>/s).
- **areap** : Area of pressurized free surface (unit: m<sup>2</sup>).
- **areapxm** : X component of the gravity center of the pressurized free surface (unit: m).
- **areapym** : Y component of the gravity center of the pressurized free surface (unit: m).
- **Pave** : Average pressure applied over pressurized free surface (unit: Pa).
- **P** : Total pressure applied over pressurized free surface (unit: Pa).
- **P(Idx,stepIdx)** : Returns the total pressure applied over the pressurized free surfaces indicated by Idx at a previous time step indicated by stepIdx. (unit: Pa).
- **vol\_P** : Volume occupied by the pressurized free surface elevation (unit: m<sup>3</sup>).
- **vol\_P(body,stepIdx)** : Returns the volume occupied by the pressurized free surfaces indicated by Idx at a previous time step indicated by stepIdx. (unit: m<sup>3</sup>).
- **Fx\_Pfs** : X component of the force due to the pressure over the pressurized free surface (unit: N).
- **Fy\_Pfs** : Y component of the force due to the pressure over the pressurized free surface (unit: N).
- **Fz\_Pfs** : Z component of the force due to the pressure over



the pressurized free surface (unit: N).

- $Mx\_Pfs$  : X component of the moment due to the pressure over the pressurized free surface (unit: N·m).
- $My\_Pfs$  : Y component of the moment due to the pressure over the pressurized free surface (unit: N·m).
- $Mz\_Pfs$  : Z component of the moment due to the pressure over the pressurized free surface (unit: N·m).

#### Deprecated variables

- $Flux\_prev$ : Vertical airflow displaced by pressurized free surfaces at previous time step. (unit:  $m^3/s$ ).
- $P\_prev$  : Total pressure applied over pressurized free surface at previous time step (unit: Pa).
- $vol\_P\_prev$  : Volume occupied by the pressurized free surface elevation at previous time step (unit:  $m^3$ ).

#### HEIGHT-LIMITED FREE SURFACE CONDITION VARIABLES

- $Fx\_Hfs$  : X component of the force due to the pressure over the height-limited free surface (unit: N).
- $Fy\_Hfs$  : Y component of the force due to the pressure over the height-limited free surface (unit: N).
- $Fz\_Hfs$  : Z component of the force due to the pressure over the height-limited free surface (unit: N).
- $Mx\_Hfs$  : X component of the moment due to the pressure over the height-limited free surface (unit: N·m).
- $My\_Hfs$  : Y component of the moment due to the pressure over the height-limited free surface (unit: N·m).
- $Mz\_Hfs$  : Z component of the moment due to the pressure over the height-limited free surface (unit: N·m).

### 3.17. Appendix B: Tcl extension

SeaFEM can be extended by using the Tcl scripting language. Tcl, or the "Tool Command Language", is a very simple, open-source-licensed programming language. Tcl provides basic language features such as variables, procedures, and control, and it runs on almost any modern OS, such as Unix, MacOS and Windows computers. But the key feature of Tcl is its extensibility.

You may find further information on Tcl at:

<http://wiki.tcl.tk/969>

SeaFEM distribution includes a basic installation of Tcl, that allows to efficiently implement new capabilities in SeaFEM. However full Tcl installation provides many tool-kits and libraries that can help in the implementation of above mentions SeaFEM extensions.

The full Tcl version can be downloaded from:

<http://www.activestate.com/activetcl/>

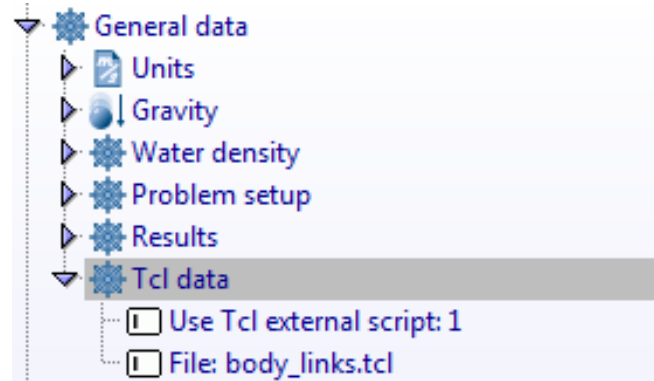
Finally, Ramdebbugger software can be used for editing and debugging Tcl code. Ramdebugger is free to use and can be downloaded from:

<http://www.compassis.com/ramdebugger>

#### 3.17.1. Initiating Tcl extension

SeaFEM Tcl extension is initiated by selecting the *Tcl extension* option available in the General data (Tcl data) page as shown in the following picture. If the check-box is selected, the Tcl extension of SeaFEM is activated. The entry must indicate a Tcl

script to be interpreted during execution.



The Tcl script used for the SeaFEM extension can implement some standard SeaFEM Tcl event procedures (see *Tcl interface procedures*). These procedures (listed below) are automatically called by SeaFEM during execution, when the Tcl interface is activated.

SeaFEM Tcl extension includes a basic library for vector operation and manipulation. Further information on the Tcl math library can be found in the Tdyn manual.

The functions for vector manipulation can operate with temporal vector created from Tcl code and with internal SeaFEM variable vectors. Internal SeaFEM vectors can be accessed from the Tcl extension using standard names. A list of the SeaFEM vectors that can be accesses from the Tcl interface are provided next:

<b>wxn</b> <b>d</b>	<b>vector of x coordinates of the mesh nodes.</b>
wyn d	vector of y coordinates of the mesh nodes.
wzn d	vector of z coordinates of the mesh nodes.
wph 1	vector of potential ( $\Phi$ ) nodal values (current value at t).
wph x	vector of nodal values of the x derivatives of the potential ( $\Phi$ ).
wph y	vector of nodal values of the y derivatives of the potential ( $\Phi$ ).
wph z	vector of nodal values of the z derivatives of the potential ( $\Phi$ ).
wph 2	vector of potential ( $\Phi$ ) nodal values (previous value at t-dt).
whfl	vector of nodal values of the free surface elevation height limit.
whf p	vector of nodal values of the pressure obtained due to free surface elevation height limit.

#### 3.17.2. Tcl interface procedures

The Tcl script used for the SeaFEM extension can implement some of the following Tcl event procedures (as well as other user-defined procedures). These procedures (listed below) are automatically called by SeaFEM during execution, when the Tcl interface is activated. Their syntax corresponds to the standard Tcl language.

- *TdynTcl\_InitiateProblem*

This procedure is called at the beginning of the execution, once all the data structures have been created.

### *TdynTcl\_FinishProblem*

This procedure is called at the end of the execution of the current problem.

- *TdynTcl\_StartSetProblem*

This procedure is called before the creation of the main structures of the problem.

- *TdynTcl\_EndSetProblem*

This procedure is called once the creation of the main structures of the problem is finished.

- *TdynTcl\_StartNewStep*

This procedure is called when a new time step is started.

- *TdynTcl\_FinishStep*

This procedure is called when a time step is finished.

- *TdynTcl\_StartNewIteration*

This procedure is called when a new iteration is started.

- *TdynTclFinishIteration*

This procedure is called when an iteration is finished.

- *TdynTcl\_CreateBodyLinks* or *TdynTcl\_DefineBodyData*

This procedure is called to initiate the body link conditions.

- *TdynTcl\_CreateMooring*

This procedure is called to create mooring lines.

- *TdynTcl\_InitiateHfs*

This procedure is called within the algorithm of definition of the free surface height (HFreeSurface condition). Within this procedure, the vector whfl should be initiated.

- *TdynTcl\_WriteResults time*

This procedure is called after writing the results of the step. The argument gives the current time for writing the result data. Note that this procedure is only called for the time steps specified in the input data for writing results.

- *TdynTcl\_StartTimeLoop*

This procedure is called just before to start the time loop of the problem.

Furthermore, most of Tcl commands included in the Tdyn Tcl interface can be also used in SeaFEM (see Tdyn user manual for further information). Additionally, several specific procedures are also available. These are described in the following:

- *TdynTcl\_Time*

Returns the current simulation time.

- *TdynTcl\_Dt*

Returns the current time increment.

- *TdynTcl\_Insert\_Interpolator\_Mesh* *interpolator\_name*  
*initial/final mesh\_id*

This procedure can be used to insert an internal SeaFEM mesh (*mesh\_id*) in a interpolator structure (*interpolator\_name*). The

available meshes are: fluid, body, frees, outlet, inlet, pfrees and hfrees. See [Examples of scripts defining a Tcl extension](#) for further information.

- *TdynTcl\_Read\_Interpolator\_Mesh\_ForHfs* *interpolator\_name*  
*initial/final mesh\_file vector\_id*

This procedure can be used to insert a mesh in GiD format (*mesh\_file*) in a interpolator structure (*interpolator\_name*) for imposing a free surface height condition. See [Examples of scripts defining a Tcl extension](#) for further information.

- *TdynTcl\_Create\_Mooring\_Segment* *body type xi yi zi xe ye ze w L*  
*A E [SN d1 d2]*

This procedure can be used to create a mooring line. The different options to create mooring lines are presented in the section [Appendix E: Mooring definition by Tcl](#). The arguments passed to this procedure are listed in what follows. Note that arguments within brackets only apply for certain types of mooring types (see argument's description below).

*body* : index of the body which the mooring is linked to (from 1 to n).

*type* : this parameter determines the type of mooring segment to be used. The possible values of this parameter are:

*type* = 1 - quasi-static elastic bar (spring able to work in both, tension and compression regimes).

*type* = 2 - quasi-static catenary

*type* = 3 - quasi-static cable (spring able to work only in tension)

*type* = 4 - quasi-static elastic catenary

*type* = 6 - dynamic cable

*xi* [m] : x coordinate of the initial point of the mooring segment (this must be the point closer to the body)

*yi* [m] : y coordinate of the initial point of the mooring segment (this must be the point closer to the body)

*zi* [m] : z coordinate of the initial point of the mooring segment (this must be the point closer to the body)

*xe* [m] : x coordinate of the end point of the mooring segment

*ye* [m] : y coordinate of the end point of the mooring segment

*ze* [m] : z coordinate of the end point of the mooring segment

*w* [N/m] : effective weight (actual weight minus bouyancy) per unit length

*L* [m] : length of the segment

*A* [m<sup>2</sup>] : cross section area of the cable

*E* [Pa] : Young modulus

*S* : this is the seabed parameter. This argument has only effect for type elastic catenary segments (type 4). If it is provided for other mooring types, it will be actually ignored. The possible values of this parameter are:

*S* = 0 - the catenary is fixed to a seabed point

*S* = 1 - considers a sandy seabed contact

*S* = 2 - considers a sliding seabed wire

*S* = 3 - considers a sliding seabed chain

*N* : This argument only applies in the case of dynamic cables (type 6). It indicates the number of line elements for cable geometric discretization.

*d1, d2* : User defined damping ratios.

- *TdynTcl\_Create\_Mooring\_Link*

This procedure creates a link between different mooring lines. The different options to create links are presented in the section [Appendix E: Mooring definition by Tcl](#).

- *TdynTcl\_Set\_Mooring\_Displacement seg1 fun1 fun2 fun3*

This procedure can be used to specify the displacement of one end of the mooring line. The arguments are:

*seg1* : identifier variable corresponding to the segment

*fun1* : index of the function describing the time-dependent displacement in OX of the end point of the mooring segment

*fun2* : index of the function describing the time-dependent displacement in OY of the end point of the mooring segment

*fun3* : index of the function describing the time-dependent displacement in OZ of the end point of the mooring segment

- *TdynTcl\_Configure\_Mooring\_Segment Gk Gc Gu Cd Cf Cm alpha\_bs gamma\_bs bci steps*

This procedure can be used to configure already existing mooring segments. It is only available for mooring segments of cable type. The arguments are:

*Gk* : seabed interaction parameter that specifies the ground normal stiffness per unit length (Pa/m)

*Gc* : seabed interaction parameter that specifies the ratio of critical damping of ground

*Gu* : seabed interaction parameter that specifies the friction coefficient

*Cd* : drag force parameter that specifies the value of the tangential drag coefficient

*Cf* : drag force parameter that specifies the value of the normal drag coefficient

*Cm* : added mass coefficient

*alpha\_bs* : coefficient of the Bossak-Newmark method

*gamma\_bs* : dissipation term for the treatment of boundary conditions

*bci* : this parameter defines the boundary conditions to be applied for cable analysis

*bci* = 1 - acceleration is imposed

*bci* = 0 - the node is free

*steps* : number of substeps to be used in the iterative solution process of the dynamic cable

- *TdynTcl\_Configure\_Mooring\_Segment id load\_curve [list e1 F1 e2 F2 ... en Fn]*

This procedure can be used to define a load-extension curve for a given dynamic cable. The arguments are:

*id* : index that identifies the cable to which the load-curve is going to be applied

*load\_curve* : a literal command that must be provided to indicate that this is a load-curve mooring configuration command (different from the generic mooring configuration command explained above)

*[list e1 F1 e2 F2 ... en Fn]* : a list tcl command used to provide pairs of extension-load (e-F) points that define the actual load curve.

- *TdynTcl\_CreateBodyLinks*

*TdynTcl\_CreateBodyLink* can be used to create correlations between degrees of freedom from different bodies. This is internally implemented by using a Lagrange's multipliers approach. This essentially consists on adding a matrix of restrictions to the bodies dynamics, each row of that matrix corresponding to the equation defining a link. The arguments passed to this function are triads of values that consecutively define the various coefficients of the link equation. To this aim, each triad consists on an integer identifying the body, another integer identifying the degree of freedom and a real value setting the corresponding coefficient value. An additional argument is given at the end to specify an optional independent coefficient that could exist for a given link equation.

The different options to create body links are presented in the section [Appendix D: Multi-body analysis](#).

- *TdynTcl\_Give\_Motions\_For\_Mesh body\_id time mesh\_file*

This procedure calculates the displacements, velocities and acceleration of the nodes of a mesh (*mesh\_file* in GiD format) calculated from the motion data of the body *body\_id*. The evaluation is done for time *time*. See [Examples of scripts defining a Tcl extension](#) for further information.

- *TdynTcl\_Add\_Mass\_Matrix body\_id [list x y z] mass\_matrix*

This procedure can be called first within the *TdynTcl\_InitiateProblem* procedure to set the mass matrix of the body. The mass matrix must be a 6x6 matrix (given as a list of 36 elements) containing the mass and inertia elements of the different degrees of freedom of the body (surge, sway, heave, roll, pitch and yaw) referred to a point given by the coordinates (*x y,z*) that are also passed as a list argument to the procedure. This procedure overwrites the definition of mass and inertia inserted in the user interface. This procedure can be called within *TdynTcl\_StartNewStep* to update the mass matrix of the body from that time step. In that case, the procedure *TdynTcl\_Build\_Mass\_Matrix* must be called afterwards. The arguments are:

*body\_id* : index of the body to define the mass matrix (from 1 to n).

*x y z* : real values (given as a list) corresponding to the coordinates of the reference point for which the mass matrix is provided.

*mass\_matrix* : list of 36 elements containing the mass and inertia elements of the different degrees of freedom of the body (surge, sway, heave, roll, pitch and yaw). Such a mass matrix refers to the point specified by the *x y z* arguments.

See [Examples of scripts defining a Tcl extension](#) for a usage example.

- *TdynTcl\_Add\_Damping\_Matrix body\_id [list x y z] damping\_matrix*

This procedure can be called first within the

*TdynTcl\_InitiateProblem* procedure to set the damping matrix of the body. The damping matrix must be a 6x6 matrix (given as a list of 36 elements) containing the damping terms corresponding to the different degrees of freedom of the body (surge, sway, heave, roll, pitch and yaw) referred to a point given by the coordinates (x,y,z) that are also passed as a list argument to the procedure. This procedure can be called within *TdynTcl\_StartNewStep* to update the damping matrix of the body from that time step. The arguments are:

*body\_id* : index of the body to define the stiffness matrix (from 1 to n).

*x y z* : real values (given as a list) corresponding to the coordinates of the reference point for which the mass matrix is provided.

*damping\_matrix* : list of 36 elements containing the damping terms corresponding to the different degrees of freedom of the body (surge, sway, heave, roll, pitch and yaw).

- *TdynTcl\_Add\_Stiffness\_Matrix body\_id [list x y z] stiffness\_matrix*

This procedure can be called first within the *TdynTcl\_InitiateProblem* procedure to set the stiffness matrix of the body. The stiffness matrix must be a 6x6 matrix (given as a list of 36 elements) containing the stiffness terms corresponding to the different degrees of freedom of the body (surge, sway, heave, roll, pitch and yaw) referred to a point given by the coordinates (x,y,z) that are also passed as a list argument to the procedure. This procedure can be called within *TdynTcl\_StartNewStep* to update the stiffness matrix of the body from that time step. The arguments are:

*body\_id* : index of the body to define the stiffness matrix (from 1 to n).

*x y z* : real values (given as a list) corresponding to the coordinates of the reference point for which the mass matrix is provided.

*stiffness\_matrix* : list of 36 elements containing the stiffness terms corresponding to the different degrees of freedom of the body (surge, sway, heave, roll, pitch and yaw).

- *TdynTcl\_Give\_Pressure\_For\_Mesh body\_id time mesh\_File*

This procedure can be called to get the total pressure over the surface of a given body. Such pressure data is returned in the form of a list of pressure values for every node of the mesh provided. These pressure values are calculated by interpolating from the mesh of the required body. The arguments are:

*body\_id* : index of the body from which we want to obtain the total pressure.

*time* : time to be reported together with the pressure information

*mesh\_File* : name of the file containing the mesh (in GiD format) to which we want to interpolate the pressure data of the given body.

- *TdynTcl\_Set\_Pfreesurface\_Pressure Idx press*

This procedure can be called to specify the total pressure *press* to be applied over the free surface of the PFreeSurface condition identified by *Idx*. The arguments are:

*Idx*: index of the PFS condition whose pressure is going to be enforced.

*press*: total pressure value to be enforced over the free surface

of the given PFS condition.

### 3.17.3. Examples of scripts defining a Tcl extension

The scripts below show examples of procedures defining a SeaFEM Tcl extension. In order to execute these procedures, they have to be saved to a file and the file has to be inserted in the Tcl data page of General Data.

#### Tcl script example 1: Write a notice in info file

The following procedure (*TdynTcl\_StartNewStep*) is executed at the beginning of every time step. It just writes a message to the standard SeaFEM output (**Calculate > View process info...**).

```
proc TdynTcl_StartNewStep {} {
    # Reading SeaFEM internal time
    set t [TdynTcl_Time]

    # Writing a message in SeaFEM info window
    TdynTcl_Message "Executing TdynTcl_StartNewStep: time $t"
    notice
}
```

#### Tcl script example 2: Loading Tk package

The following script loads Tk package. Tk is Tcl a library, including basic elements for building a graphical user interface. Once this library is loaded, graphic elements can be created from the Tdyn Tcl interface. In this example, a text window is created and then every time step, the text "Step \$Current\_Step\$" is printed in that window.

In the following code, the full Tcl installation is assumed to be in the directory "C:\Program Files\Tcl\lib". The full Tcl installation can be downloaded from <http://www.activestate.com/activetcl/>.

```
# Define directory of the Tcl installation
lappend ::auto_path {C:\Program Files\Tcl\lib}

TdynTcl_Message [package require Tk] notice

# Creates a text window
pack [text .t -width 70 -height 20]

# Prints information in the text window

proc TdynTcl_StartNewStep {} {
    set time [TdynTcl_Time]

    .t ins end "Time $time\n" ; .t see end ; update
}
```

#### Tcl script example 3: Initiate Hfs

The following script initiate the free surface height of a surface (with HFreeSurface condition applied) interpolating the z coordinate from a mesh file in GiD format (mesh.msh).

```
proc TdynTcl_InitiateHfs {} {
    #Initiate elevation
    TdynTcl_Message "Initiate z!!!" notice
    set inter2 [TdynTcl_Create_Interpolator]
```

```
set tmpvec [::mather::mkvector 1 0.0]

TdynTcl_Insert_Interpolator_Mesh $inter2 initial hfreess

TdynTcl_Read_Interpolator_Mesh_ForHfs $inter2 final "mesh.msh"
$tmpvec

TdynTcl_OnInitial_Interpolator $inter2 $tmpvec whfl

TdynTcl_Release_Interpolator $inter2

vmdelete $tmpvec

}
```

#### Tcl script example 4: Communicate SeaFEM with Ramseries using HFreeSurface condition

The following script communicates in calculation time SeaFEM with Ramseries. SeaFEM sends to Ramseries (every time step) the pressure calculated on a HFreeSurface condition. Ramseries uses this information to apply a pressure load on the structure and send back the displacements of the structure, used in SeaFEM to update HFreeSurface condition (height elevation).

```
proc TdynTcl_InitiateHfs {} {

#Initiate elevation

TdynTcl_Message "Initiate z!!!" notice

set inter2 [TdynTcl_Create_Interpolator]

set tmpvec [::mather::mkvector 1 0.0]

TdynTcl_Insert_Interpolator_Mesh $inter2 initial hfreess

TdynTcl_Read_Interpolator_Mesh_ForHfs $inter2 final "mesh.msh"
$tmpvec

TdynTcl_OnInitial_Interpolator $inter2 $tmpvec whfl

TdynTcl_Release_Interpolator $inter2

vmdelete $tmpvec

}

proc TdynTcl_InitiateProblem {} {

global myvec myvec2 myvec3 myrec nnode inter

# nnRam must be set to the number of nodes of the structural mesh
(fingers.msh)

set nnRam 2619

# Create the vectors

set myvec [::mather::mkvector $nnRam 0.0]

set myrec [::mather::mkvector $nnRam 0.0]

set myvec2 [::mather::mkvector $nnRam 0.0]

set myvec3 ""

# Init coupling

mather_initcoupling 0 2010 100000

TdynTcl_Message "Connected!!!" notice

# Insert mesh for interpolation (mesh.msh is the structural mesh)

mather_insertcouplingmesh "mesh.msh"
```

```
# Init interpolator structure to pass information from the updated
HFs mesh to fingers.msh

set inter [TdynTcl_Create_Interpolator]

TdynTcl_Insert_Interpolator_Mesh $inter initial hfreess

TdynTcl_Read_Interpolator_Mesh $inter final "mesh.msh"

}

proc TdynTcl_StartNewStep {} {

global myvec myvec2 myvec3 myrec nnode inter

set step [TdynTcl_Time]

TdynTcl_Message "-----> Interpolate data of pressure to the
structural mesh" notice

TdynTcl_OnFinal_Interpolator $inter whfp $myvec

TdynTcl_Message "-----> Starts to send pressure vector for time
$step" notice

# Note: Only first component (pressure) of the traction vector is sent

mather_sendcouplingvector $myvec 0 $step

TdynTcl_Message "-----> Vector sent for time $step" notice

}

proc TdynTcl_FinishStep {} {

#return

global myvec myvec2 myvec3 myvec4 myrec nnode inter

set step [TdynTcl_Time]

TdynTcl_Message "-----> Starts to retrieve displacement increment
for time $step" notice

# Only z component is used

set data [mather_retrievecouplingheader]

mather_retrievecouplingvector $myrec 0 $step

set data [mather_retrievecouplingheader]

mather_retrievecouplingvector $myrec 0 $step

set data [mather_retrievecouplingheader]

mather_retrievecouplingvector $myrec 0 $step

TdynTcl_Message "-----> Info received for time $step" notice

# Convergence check

mather_sendcouplingconvergence $step 1

set conv [mather_retrievecouplingconvergence]

TdynTcl_Message "-----> Convergence check for time $step: $conv"
notice

# Interpolate and update the displacement

if { $myvec3 eq "" } {

set myvec3 [::mather::mkvector [::mather::vector_info length whfl]
0.0]
```



```
set myvec4 [::mather::mkvector [::mather::vector_info length whfl]
0.0]

}

TdynTcl_OnInitial_Interpolator $inter $myrec $myvec3

vmexpr whfl+=$myvec3

TdynTcl_Message "-----> TdynTcl_FinishStep" notice

}
```

### Tcl script example 5: Debugging a Tcl script with Ramdebugger

RamDebugger is a graphical debugger for Tcl-TK. With RamDebugger, it is possible to make Local Debugging, where the debugger starts the program to be debugged, and Remote debugging, where the program to debug is already started and RamDebugger connects to it. The latter option will be used in this case.

#### Remark:

In Windows, it is necessary to load the package comm (not the standard, but the one modified in RamDebugger), in order to debug the program remotely.

To debug the following example:

1. open it with Ramdebugger and set a breakpoint in the line "TdynTcl\_Message "Set a breakpoint in this line" notice".
2. Then execute Tdyn, and wait for a few seconds, until the execution freezes.
3. Go to Ramdebugger and select

#### File ► Debug on ► Tdyn

4. Tdyn execution will restant until the breakpoint is find.

#### Remark:

If Tdyn is not included in the list of "Remote TCL debugging" programs, update the list by selecting

#### File ► Debug on ► Update remotes

```
# Load package commR

# Change the directory below with the one where Ramdebugger is
intalled

lappend ::auto_path {C:/Utils/RamDebugger7.0/addons}

TdynTcl_Message [package require commR] notice

# This register Tdyn for debugging

comm::register Tdyn 1

# Add breakpoint beyond this point.

# breakpoints only work inside a proc.

proc TdynTcl_FinishStep {} {

    TdynTcl_Message "Set a breakpoint in this line" notice
    TdynTcl_Message "Click the arrow button to go to this line"
    notice

}

# Program gets stopped here waiting for the debugger to connect

commR::wait_for_debugger
```

### Tcl script example 6: Calculating rigid body motions for a mesh

The following script calculates rigid body displacements, velocities and accelerations for the nodes of a given mesh (GiD ASCII format) and writes the results in an ASCII file (GiD format).

```
proc TdynTcl_InitiateProblem {} {

    global resfile mshfile

    set mshfile "C:/Users/julio/Desktop/Current/spar_mesh.msh"

    set resfile "C:/Temp/spar_acc.res"

    set fileid [open $resfile w+]

    puts $fileid "GiD Post Results File 1.0"

    close $fileid

}

proc TdynTcl_WriteResults { time } {

    global resfile mshfile

    # Arguments for TdynTcl_Give_Motions_For_Mesh are body_index,
    time and mesh_file_name

    set acclist [TdynTcl_Give_Motions_For_Mesh 1 $time $mshfile]

    set fileid [open $resfile a]

    puts $fileid "Result \"Displacement (m)\" Body $time Vector
    OnNodes"

    puts $fileid "Values"

    set i 1

    foreach inode $acclist {

        puts $fileid "$i [lrange $inode 0 2]"

        incr i

    }

    puts $fileid "End Values"

    puts $fileid "Result \"Velocity (m/s)\" Body $time Vector
    OnNodes"

    puts $fileid "Values"

    set i 1

    foreach inode $acclist {

        puts $fileid "$i [lrange $inode 3 5]"

        incr i

    }

    puts $fileid "End Values"

    puts $fileid "Result \"Acceleration (m/s2)\" Body $time Vector
    OnNodes"

    puts $fileid "Values"

    set i 1

    foreach inode $acclist {

        puts $fileid "$i [lrange $inode 6 8]"

    }

}
```

```
incr i
}
puts $fileid "End Values"
close $fileid
}
```

### Tcl script example 7: Create mooring lines

The following script creates 3 quasi-static catenary mooring lines set at 120° to each other.

```
proc TdynTcl_CreateMooring { } {
# Mooring line 1: type 2: catenary

# arguments: body type xi[m] yi[m] zi[m] xe[m] ye[m] ze[m] w[N/m]
L[m] A[m2] E[Pa]

set cat1 [TdynTcl_Create_Mooring_Segment 1 2 -380.0 0.0 -210.0
-4.15 0.0 -60.0 1350 -1 0.02 2.05e11]

# Mooring line 2: type 2: catenary

# arguments: type xi[m] yi[m] zi[m] xe[m] ye[m] ze[m] w[N/m] L[m]
A[m2] E[Pa]

set cat2 [TdynTcl_Create_Mooring_Segment 1 2 190.0 329.1 -210.0
2.1 3.6 -60.0 1350 -1 0.02 2.05e11]

# Mooring line 3: type 2: catenary

# arguments: type xi[m] yi[m] zi[m] xe[m] ye[m] ze[m] w[N/m] L[m]
A[m2] E[Pa]

set cat3 [TdynTcl_Create_Mooring_Segment 1 2 190.0 -329.1 -210.0
2.1 -3.6 -60.0 1350 -1 0.02 2.05e11]

TdynTcl_Message "TdynTcl_CreateMooring finished!!!" notice
}
```

The following script creates a multi-segment delta line:

```
proc TdynTcl_CreateMooring { } {
set type 3
set area 1.13E-006
set w 0.076
set E 2.10E+011
# segments

set segA [create_mooring_segment $type -3.58 0.00 -2.07 -4.49 0.0
-4.78 $w 2.9370 $area $E 0]

set segB [create_mooring_segment $type -1.28 0.00 -1.89 -3.58 0.0
-2.07 $w 2.5590 $area $E 0]

set segC [create_mooring_segment $type -0.17 0.00 -0.64 -1.28 0.0
-1.89 $w 1.7620 $area $E 0]

set segD [create_mooring_segment $type -0.32 0.00 -2.99 -1.28 0.0
-1.89 $w 1.5590 $area $E 0]

# links
create_mooring_link $segB $segA 230.0
create_mooring_link $segC $segB $segD 0.0
}
```

The following script creates eight pre-stressed spring lines:

```
proc TdynTcl_CreateMooring { } {
# Mooring lines: type 1: spring

# arguments: body type xi[m] yi[m] zi[m] xe[m] ye[m] ze[m] w[N/m]
L[m] A[m2] E[Pa]

set cat1 [TdynTcl_Create_Mooring_Line 1 1 -26.0 -2.6 -150.0 -26.0
-2.6 -47.0 386.7 102.52 0.00503 2.05e11]

set cat2 [TdynTcl_Create_Mooring_Line 1 1 -26.0 2.6 -150.0 -26.0 2.6
-47.0 386.7 102.52 0.00503 2.05e11]

set cat3 [TdynTcl_Create_Mooring_Line 1 1 26.0 -2.6 -150.0 26.0 -2.6
-47.0 386.7 102.52 0.00503 2.05e11]

set cat4 [TdynTcl_Create_Mooring_Line 1 1 26.0 2.6 -150.0 26.0 2.6
-47.0 386.7 102.52 0.00503 2.05e11]

set cat5 [TdynTcl_Create_Mooring_Line 1 1 -2.6 26.0 -150.0 -2.6 26.0
-47.0 386.7 102.52 0.00503 2.05e11]

set cat6 [TdynTcl_Create_Mooring_Line 1 1 2.6 26.0 -150.0 2.6 26.0
-47.0 386.7 102.52 0.00503 2.05e11]

set cat7 [TdynTcl_Create_Mooring_Line 1 1 -2.6 -26.0 -150.0 -2.6
-26.0 -47.0 386.7 102.52 0.00503 2.05e11]

set cat8 [TdynTcl_Create_Mooring_Line 1 1 2.6 -26.0 -150.0 2.6 -26.0
-47.0 386.7 102.52 0.00503 2.05e11]

TdynTcl_Message "TdynTcl_CreateMooring finished!!!" notice
}
```

The following script creates a mooring line (spring), defining the displacement of the end point by functions.

```
proc TdynTcl_CreateMooring { } {
set seg1 [TdynTcl_Create_Mooring_Segment 1 1 0.0 0.0 0.0 100.0 0.0
0.0 0.0 100.0 0.1 1.0e6]

set fun1 [::mather::create_function waves "1.0*sin(0.5*t);"]
set fun2 [::mather::create_function waves "0.0;"]
set fun3 [::mather::create_function waves "0.0;"]

TdynTcl_Set_Mooring_Displacement $seg1 $fun1 $fun2 $fun3

TdynTcl_Message "TdynTcl_CreateMooringLine finished!!!" notice
}
```

### Tcl script example 8: Creating functions

This script creates a function '0.9\*density\*volume;' at the beginning of the execution. Then a procedure My\_Mass is created to evaluate that function. The procedure can be accessed from SeaFEM function fields, by using 'tcl(My\_Mass)'.

```
proc TdynTcl_InitiateProblem { } {
global myfunc

set myfunc [::mather::create_function waves "0.9*density*volume;"]
}

proc My_Mass { } {
global myfunc
```

```
set ret [::mather::evaluate_function $myfunc]

TdynTcl_Message "My_Mass = $ret" notice

return $ret
}
```

### Tcl script example 9: Creating a body link

In this example, a link is provided between the surge degree of freedom of body 1 ( $x_1$ ) and the surge and pitch degrees of freedom ( $x_2$  and  $ry_2$ ) of body 2. In this equation coefficients concerning  $x_1$  and  $x_2$  are 1.0 and -1.0 respectively, while the coefficient concerning  $ry_2$  is the z-distance between the gravity center of the two bodies being equal to 2.0. No independent term exists for this link.

```
proc TdynTcl_CreateBodyLinks {} {

#  $x_1 - x_2 + (z_{g2} - z_{g1}) * ry_2 = 0$ 

set blnk2 [create_body_link 1 1 1.0 2 1 -1.0 2 5 +2.0 0.0]

TdynTcl_Message "Body link 1 finished!!!" notice

}
```

### Tcl script example 10: Interpolate to an external mesh the pressure acting on a given body

The following script interpolates into an external mesh representing a cylinder, the pressure acting on the main body of the simulation (body number 1). The operation is performed every time step and the results are written into a text file using GiD results format.

```
proc TdynTcl_InitiateProblem {} {

global mshfile resfile

set mshfile "Cylinder.msh"

set resfile "Cylinder.flavia.res"

set fileid [open $resfile w+]

puts $fileid "GiD Post Results File 1.0"

close $fileid

}

proc TdynTcl_FinishStep {} {

global mshfile resfile

set time [TdynTcl_Time]

set acclist [TdynTcl_Give_Pressure_For_Mesh 1 $time $mshfile]

set fileid [open $resfile a]

puts $fileid "Result \"Pressure (Pa)\" Body $time Scalar OnNodes"

puts $fileid "Values"

set i 1

foreach inode $acclist {

puts $fileid [lrange $inode 0 1]

incr i

}
```

```
puts $fileid "End Values"

close $fileid

}
```

### Tcl script example 11: Defining a mass matrix

The following script illustrates how to define the mass matrix of a body taking a generic point as the reference point for that matrix. Note that the mass matrix can be directly introduced within the graphic user interface of SeaFEM only in the case the matrix is known for the center of gravity. Hence, the tcl script procedure is useful when the mass matrix is known from a generic reference point that may differ from the center of gravity. If this is the case, the mass matrix in the graphic user interface is left empty and the following tcl script is used to set up the mass matrix for the generic reference point. SeaFEM takes care of transforming the mass matrix from the reference point specified in the tcl script to the center of gravity or any other generic point specified for the sake of output in the user interface. In the present example, the generic point from which the mass matrix is specified is the point of coordinates (15, -25, -7). The first argument in the *TdynTcl\_Add\_Mass\_Matrix* procedure indicates that the specified mass matrix concerns body number 1.

```
proc TdynTcl_InitiateProblem {} {

TdynTcl_Add_Mass_Matrix 1 [list 15 -25 -7] [list \
1607.95 0 0 0 11255.65 -40198.75 \
0 1607.95 0 -11255.65 0 -24119.25 \
0 0 1607.95 40198.75 24119.25 0 \
0 -11255.65 40198.75 1085366 602981 168835 \
11255.65 0 24119.25 602981 442186.3 -281391.3 \
-40198.75 -24119.25 0 168835 -281391.3 1368365 ]

TdynTcl_Message "TdynTcl_InitiateProblem finished!!!" notice

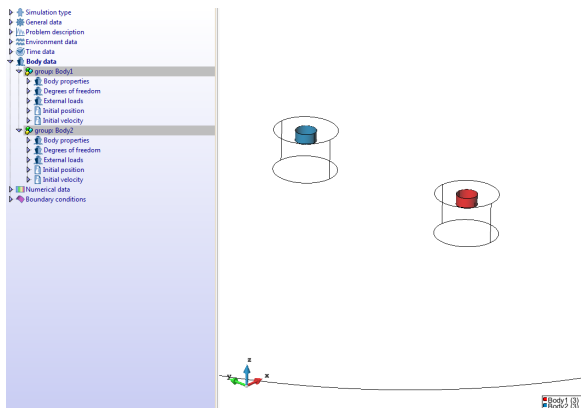
}
```

## 3.18. Appendix C: RAO analysis

When carrying out RAO analysis using the white noise spectrum, it must be taken into account that the SeaFEM RAO's calculation is based on a discrete Fourier transform decomposition. Because of this, the total calculation time and the sampling time are automatically calculated and fixed internally by the solver based on the maximum and the minimum frequencies (periods) specified by the user in the graphic user interface. Hence, even if a "Simulation time" and/or a "Time step" are specified by the user, they will be ignored and fixed internally by SeaFEM. Only the "Initialization time" parameter will be maintained since it can be useful to avoid energetic transient. This is because a sudden initialization (which can occur when no initialization time is specified) might lead to large accumulation of energy by the body, which would require longer simulation to dissipate all that unrealistic body energy.

## 3.19. Appendix D: Multi-body analysis

SeaFEM can perform analysis of multi-body configurations. In newer versions of SeaFEM, all bodies can be defined through the graphic user interface. To this aim, as many body data conditions as needed can be defined and assigned to the corresponding geometric entities. In the figure below for instance, two independent body data conditions were assigned to body1 and body2 groups that correspond to the 2 cylinders shown.



Hence, each group has assigned its own body properties, degrees of freedom, external loads and initial positions and velocities. Nevertheless, if any property within the condition is specified using a function, the variables used still require the body index to be indicated between parenthesis. For example, if body mass is defined using a function of the form  $M = vol * density$  the actual body to which the variable  $vol$  applies must be indicated between parenthesis. Hence, the mass of the first body would actually read  $M = vol(1) * density$  and so on. Keep in mind that body indices are assigned in the same order the bodies are defined in the data tree.

The results (movements, forces, etc.) of all bodies, are saved in the `model` folder, in the files `'$model_name$.BodyKinematics.res'` and `'$model_name$.BodyLoads.res'`. Furthermore, they can be visualised in the 'SeaFem graphs' option of the Postprocess menu.

### Body links

Within SeaFEM, it is also possible to define links between different bodies. This can be done in two ways. First, action-reaction forces exerted between two given bodies can be introduced as external loads acting on the corresponding bodies. These functions use the standard syntax shown in section [Appendix A: function editor](#), but it is extended for multi-bodies analysis, by adding the index of the body between parenthesis at the end of the function. The following are some examples of this extension:

- `mass(2)`: mass of the body of index 2.
- `dx[1.0,1.0,1.0](1)`: displacement of the point 1.0,1.0,1.0 of the body of index 1.
- `yg(3)`: Y coordinate of the center of gravity of the body of index 3.

Secondly, links can also be created that correlate degrees of freedom from different bodies. This is internally implemented by using a Lagrange's multipliers approach. This essentially consists on adding a matrix of restrictions to the bodies dynamics system, each row of that matrix corresponding to the equation defining a link. At the user's level, it is necessary to define these kind of links by using a tcl script procedure. In particular, a Tcl procedure named `TdynTcl_CreateBodyLinks` must be reimplemented by the user in the tcl script. Within these procedure, several types of joints between bodies (each implying different number of body links) can be defined following the syntax detailed below. In addition, body links can be optionally updated at each iteration to take into account large displacements and rotations. To this aim, a Tcl procedure named `TdynTcl_StartNewIteration` must be reimplemented by the user in the tcl script. The syntax of the tcl functions used to update the joints (links) is identical to their creation counterpart

just adding a first argument to identify the constraint equation index to be updated.

The type of body joints currently available within SeaFEM is listed below:

- **Rotation constraint**
- **Line constraint**
- **Plane constraint**
- **Rotation link**
- **Translation link**
- **Rigid body joint**
- **Ball joint**
- **Revolute joint**
- **Cylindrical joint**
- **Prismatic joint**
- **Translational joint**

Next, the above body constraints and mechanical joints are described in detail alongside with the arguments required to define each one of the body link relations.

#### • Rotation constraint

This type of constraint is used to prevent body rotation around an arbitrary given direction.

`create_rotation_constraint #body #a #b #c`

`update_rotation_constraint #ilink #body #a #b #c`

Argument `#body1` must be an integer value (greater than 0) and is used to identify the body affected by the rotation constraint. Such an index must respond to the order of creation of the bodies in the interface. Arguments `#a #b #c` are the direction cosines

of the axis about which the rotation of the body is prevented.

#### • Line constraint

This type of constraint is used to enforce body movement along a given direction (the body is allowed to move only

along a given direction).

`create_line_constraint #body #a #b #c`

`update_line_constraint #ilink #body #a #b #c`

Argument `#body1` must be an integer value (greater than 0) and is used to identify the body affected by the line constraint. Such an index must respond to the order of creation of the bodies in the interface. Arguments `#a #b #c` are the direction cosines

of the direction along which the body can move.

#### • Plane constraint

This type of constraint is used to enforce body movement over a given plane. The body is allowed to move only over a plane whose normal is given and going through the initial position of the body's reference point.

`create_plane_constraint #body #a #b #c`

`update_plane_constraint #ilink #a #b #c`

Argument `#body1` must be an integer (greater than 0) and is used to identify the body to which the plane constraint applies.

This body index must be consistent with the order of creation of the bodies in the user interface. Arguments *#a #b #c* are the direction cosines of the plane's normal direction.

#### • Rotation link

This type of joint is used to enforce the rotation degree of freedom of two different bodies to be the same.

*create\_rotation\_link #body1 #body2 #rotdof*

*update\_rotation\_link #ilink #body1 #body2 #rotdof*

Arguments *#body1* and *#body2* must be integer values greater than 0 and are used to identify the two bodies being joined by the rotation link. These indexes correspond to the order of creation of the bodies in the interface. Argument *#rotdof* must be an integer value between 1 and 3 and is used to identify the constrained rotational degree of freedom (1 = Roll, 2 = Pitch, 3 = Yaw). Argument *#ilink* identifies the constraint equation index to be updated. It usually corresponds to an integer variable previously initialized by the return value of the *create\_link* function.

#### • Translation link

This type of link is used to create a constraint between translational degrees of freedom of 2 bodies.

*create\_translation\_link #body1 #body2 #transdof*

*update\_translation\_link #ilink #body1 #body2 #transdof*

Arguments *#body1* and *#body2* must be integer values (greater than 0) and are used to identify the two linked bodies. These indexes correspond to the order of creation the different bodies in the interface. Argument *#transdof* must be an integer value between 1 and 3 and is used to identify the constrained translational degree of freedom (1 = Surge, 2 = Sway, 3 = Heave). Argument *#ilink* identifies the constraint equation index to be updated. It usually corresponds to an integer variable previously initialized by the return value of the *create\_link* function.

#### • Rigid body joint

This type of joint can be used to enforce two different bodies to move as a single rigid body. The advantage of modelling the rigid body as two separate components instead of a unique body, is that forces and moments exerted by one component over the other are then accessible in the form of body joint reactions.

*create\_rigid\_body\_full\_link #body1 #body2*

*update\_rigid\_body\_full\_link #ilink #body1 #body2*

Arguments *#body1* and *#body2* must be integer values greater than 0 and are used to identify the two components of the rigid body set. These indices must correspond to the order the bodies were created in the user interface. Argument *#ilink* identifies the constraint equation index to be updated. It usually corresponds to an integer variable previously initialized by the return value of the *create\_link* function.

#### • Ball joint

This type of link is used to create a ball joint constraint between 2 bodies. This means that the two bodies being joined share a common point (usually the ball joint position point) about which both bodies can rotate freely.

*create\_ball\_joint\_link #body1 #body2 #x #y #z*

*update\_ball\_joint\_link #ilink #body1 #body2 #x #y #z*

Arguments *#body1* and *#body2* must be integer values greater than 0 and are used to identify the two linked bodies. These indexes correspond to the order of creation of the different bodies in the interface. Arguments *#x*, *#y* and *#z* must be real values that define the initial global position of the ball joint. Argument *#ilink* identifies the constraint equation index to be updated. It usually corresponds to an integer variable previously initialized by the return value of the *create\_link* function.

#### • Revolute joint

This type of joint is used to create a revolute constraint between 2 bodies. This type of junction implies that the two bodies can freely rotate about a common axis (actually the bodies share two points aligned on a given axis), while the remaining degrees of freedom (translation and rotation) are linked.

*create\_revolute\_joint\_link #body1 #body2 #x1 #y1 #z1 #x2 #y2 #z2*

*update\_revolute\_joint\_link #ilink #body1 #body2 #x1 #y1 #z1 #x2 #y2 #z2*

Arguments *#body1* and *#body2* must be integer values greater than 0 and are used to identify the two bodies joined through the revolute joint. These indexes correspond to the order of creation of the different bodies in the interface. Arguments *#x1*, *#y1*, *#z1* and *#x2*, *#y2*, *#z2* must be real values defining the two points shared by the bodies. Actually these points define the common axis of revolution.

#### • Cylindrical joint

This type of link is used to create a cylindrical joint between 2 bodies. A cylindrical joint implies that the two bodies can undergo relative translation along a given axis and relative rotation about the same axis. All the remaining degrees of freedom are linked.

*create\_cylindrical\_joint\_link #body1 #body2 #a #b #c*

*update\_cylindrical\_joint\_link #ilink #body1 #body2 #a #b #c*

Arguments *#body1* and *#body2* must be integer values greater than 0 and are used to identify the two bodies linked by the cylindrical joint. These indexes correspond to the order of creation of the different bodies in the interface. Arguments *#a*, *#b* and *#c* must be real values that define the vector/axis that characterizes the cylindrical joint. Argument *#ilink* identifies the constraint equation index to be updated. It usually corresponds to an integer variable previously initialized by the return value of the *create\_link* function.

#### • Prismatic joint (deprecated)

This type of link is used to create a prismatic joint constraint between 2 bodies. A prismatic joint means that all rotational degrees of freedom are linked (this is, both bodies undergo identical roll, pitch and yaw rotations) and that both bodies can undergo relative translation only along a given axis. Such an axis is limited to be aligned along one of the global coordinate system axes (i.e. X, Y, or Z global axes). This type of link is deprecated and superseded by the Translational joint described below. Nevertheless, Prismatic joint are still available to ensure backward compatibility.

*create\_prismatic\_joint\_link #body1 #body2 #axis*

*update\_prismatic\_joint\_link #ilink #body1 #body2 #axis*

Arguments *#body1* and *#body2* must be integer values greater



than 0 and are used to identify the two joined bodies. These indexes correspond to the order of creation the different bodies in the interface. Argument *#axis* must be an integer value greater than 1 that determines the axis along which the two bodies can undergo relative displacement. Argument *#ilink* identifies the constraint equation index to be updated. It usually corresponds to an integer variable previously initialized by the return value of the *create\_link* function.

#### • Translational joint

This type of link generalizes the prismatic joint. It is used to create a prismatic joint constraint between 2 bodies so that all rotational degrees of freedom are linked (this is, both bodies undergo identical roll, pitch and yaw rotations) while both bodies can undergo relative translation along a given generic axis.

```
create_prismatic_joint_link #body1 #body2 #a #b #c
```

```
update_prismatic_joint_link #ilink #body1 #body2 #a #b #c
```

Arguments *#body1* and *#body2* must be integer values greater than 0 and are used to identify the two joined bodies. These indexes correspond to the order of creation the different bodies in the interface. Arguments *#a #b #c* must be real values that define the vector/axis that characterizes the translational joint. Argument *#ilink* identifies the constraint equation index to be updated. It usually corresponds to an integer variable previously initialized by the return value of the *create\_link* function.

An example of tcl script procedure reads as follows:

```
proc TdynTcl_CreateBodyLinks {} {
    global link1 link2

    set link1 [create_ball_joint_link 1 2 0.0 0.0 -1.0]
    set link2 [create_rotation_link 1 2 0]
}

proc TdynTcl_StartNewIteration {} {
    global link1 link2

    update_ball_joint_link $link1 1 2 0.0 0.0 -1.0
    update_rotation_link $link2 1 2 0
}
```

In this example, first a ball joint link is created between bodies 1 and 2 (the corresponding equation index is stored in the global variable *link1*). The ball joint is located at the position given by the initial global coordinates 0.0, 0.0, -1.0. Secondly, an additional constraint between bodies 1 and 2 is created so that both bodies rigidly rotate in roll direction (link 2).

Generic links can also be created that correlate any number of degrees of freedom from different bodies. The implementation of this kind of links essentially consists on calling the *TdynTcl\_CreateBodyLink* function through the Tcl script. The arguments passed to this function are triads of values that consecutively define the various coefficients of the particular link equation. To this aim, each triad consists on an integer identifying the body, another integer identifying the degree of freedom and a real value setting the corresponding coefficient value. An additional argument can be provided at the end to specify an optional independent coefficient that could exist for a given link equation. An example of tcl script procedure for this

kind of generic link would read as follows:

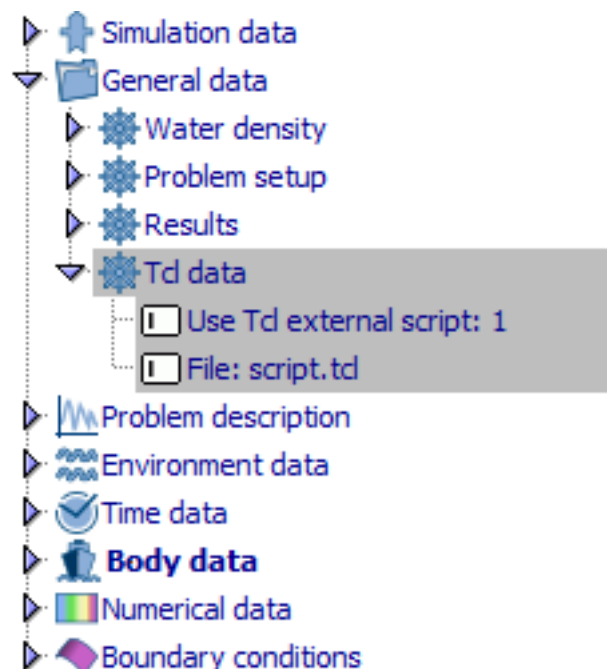
```
proc TdynTcl_CreateBodyLinks {} {
    # x_1 - x_2 + (zg2 - zg1)*ry_2 = 0
    TdynTcl_CreateBodyLink 1 1 1.0 2 1 -1.0 2 5 2.0 0.0
}
```

In this example, a link is provided between the surge degree of freedom of body 1 and the surge degree of freedom of body 2. In this equation, coefficients concerning *x\_1* and *x\_2* are 1.0 and -1.0 respectively, while the coefficient concerning *ry\_2* is the z-distance between the gravity center of the two bodies (*zg2 - zg1*) being equal to 2.0. No independent term exists for this link.

It is also possible to define links between bodies using the different mooring options of SeaFEM. See [Appendix E: Mooring definition by Tcl](#) section for further information.

### 3.20. Appendix E: Mooring definition by Tcl

Mooring definition in SeaFEM can be done by using the GUI options available (see [Mooring data](#) section for further information). However, mooring systems can also be defined through a Tcl script procedure, that gives access to advanced mooring definition options. The Tcl procedure must be implemented in a text file. Such a file, is further provided as an input argument of the Tcl data section of the data tree, once the option to use an external Tcl script is activated.



Tcl data section of the SeaFEM data tree

Tcl data can be accessed and modified at any time through the menu option:

#### General data ► Tcl data

The procedure to be implemented in the tcl script is named *TdynTcl\_CreateMooringLine* and simply consists on the definition of all mooring segments and further definition of the segment links. Segments definition is performed by calling the SeaFEM internal function *create\_mooring\_segment*. Arguments passed to this function are described below. Links between

segments are specified by calling the SeaFEM internal function `create_mooring_link`. Arguments passed to this function are also described below.

The syntax to call **TdynTcl\_Create\_Mooring\_Segment** and the arguments passed to the function are as follows:

syntax : `TdynTcl_Create_Mooring_Segment $body $type $xi $yi $zi $xe $ye $ze $w $L $A $E [$S $N $d1 $d2]`

alternative syntax: `create_mooring_segment $body $type $xi $yi $zi $xe $ye $ze $w $L $A $E [$S $N $d1 $d2]`

The arguments between brackets only apply to certain types of mooring segments.

- **body** : index of the body which the mooring is linked to (from 1 to n). If no body is specified, the current mooring segment is assumed to be attached to the main body
- **type** : this parameter determines the type of mooring segment to be used. The possible values of this parameter are:
  - type = 4 - catenary
  - type = 1 - spring
  - type = 3 - spring only traction
  - type = 6 - dynamic cable
- **xi** [m] : x coordinate of the initial point of the mooring segment (this must be the point closer to the body)
- **yi** [m] : y coordinate of the initial point of the mooring segment (this must be the point closer to the body)
- **zi** [m] : z coordinate of the initial point of the mooring segment (this must be the point closer to the body)
- **xe** [m] : x coordinate of the end point of the mooring segment
- **ye** [m] : y coordinate of the end point of the mooring segment
- **ze** [m] : z coordinate of the end point of the mooring segment
- **w** [N/m] : effective weight (actual weight minus bouyancy) per unit length
- **L** [m] : length of the segment
- **A** [m<sup>2</sup>] : cross section area of the cable
- **E** [Pa] : Young modulus
- **S** : this is the seabed parameter. This argument has only effect for catenary segments and dynamic cables. If it is provided for other mooring types, it will be actually ignored.
- **S = 0** - No seabed contact. The segment has at both ends a 'Connection point', or a 'Connection point' and a 'Fairlead point'.
- **S = 1,...,7** - the segment has an 'Anchor point'. The possible values of this parameter 'S' are as follows:

S = 1 - Frictionless seabed (no seabed friction)

S = 2 - Chain with sandy seabed (chain + sandy seabed contact exist)

S = 3 - Chain with mud/sand seabed (chain + mud/sand seabed contact exist)

S = 4 - Chain with mud/clay seabed (chain + mud/clay seabed contact exist)

S = 5 - Wire rope with sandy seabed (wire rope + sandy seabed contact exist)

S = 6 - Wire rope with mud/sand seabed (wire rope + mud/sand seabed contact exist)

S = 7 - Wire rope with mud/clay seabed (wire rope + mud/clay seabed contact exist)

- **N** : This argument only applies in the case of dynamic cables (type 6). It indicates the number of line elements for cable geometric discretization.
- **d1, d2** : User defined damping ratios.

For each segment, initial and end points must be introduced in a specific order, being the initial point the closest one to the body. The function simply returns an identifier for the created segment, that can be used to define links among the different mooring segments. These links are defined using the function `create_mooring_link`, as defined in the following.

By default, `create_mooring_segment` assumes that the initial point is linked to the given body, and the end point is fixed at the seabed. However, this assumption can be overwritten by defining links with other mooring segments or bodies. As commented above, the function `create_mooring_link` can be used to define links among the different segments of a multi-segment mooring line. This function can create different type of links as explained below.

The syntax to call **create\_mooring\_link** and the arguments passed to the function are as follows:

syntax : `create_mooring_link $seg1 $seg2 $f`

alternative syntax: `TdynTcl_Create_Mooring_Link $seg1 $seg2 $f`

- **seg1** : identifier variable corresponding to the first segment of the link (this must be the segment closer to the body)
- **seg2** : identifier variable corresponding to the second segment of the link
- **f** [N] : buoyancy force to be applied at the link node

The operator **TdynTcl\_Create\_Mooring\_Link** can also be used to define links between three lines (joint to the same node). In that case, the syntax of the function is as follows:

syntax : `TdynTcl_Create_Mooring_Link $seg1 $seg2 $seg3 $f`

alternative syntax: `create_mooring_link $seg1 $seg2 $seg3 $f`

- **seg1** : identifier variable corresponding to the first segment of the link (the segment closer to the body)
- **seg2** : identifier variable corresponding to the second segment of the link
- **seg3** : identifier variable corresponding to the third segment of the link
- **f** [N] : buoyancy force to be applied at the link node

The function **TdynTcl\_Create\_Mooring\_Link** can also be used to define links between two bodies. In that case, the syntax of the function is as follows:

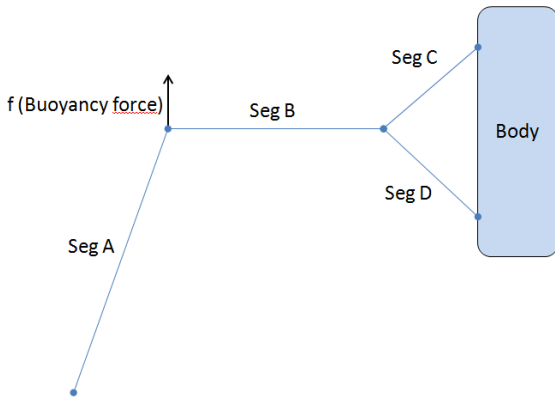
syntax : `TdynTcl_Create_Mooring_Link $seg1 &body`

alternative syntax: `create_mooring_link $seg1 &body`

- **seg1** : identifier variable corresponding to the segment of the link

**body** : index of the body which the mooring is linked to (from 1 to n).

An example of a TCL script for defining a multi-catenary mooring system is shown below. It corresponds to the mooring system configuration sketched in the following figure:



```
proc TdynTcl_CreateMooring {} {
    set type 3
    set area 1.13E-006
    set w 0.076
    set E 2.10E+011
    # segments
    set segA [create_mooring_segment $type -3.58 0.00 -2.07 -4.49
    0.0 -4.78 $w 2.9370 $area $E 0]
    set segB [create_mooring_segment $type -1.28 0.00 -1.89 -3.58
    0.0 -2.07 $w 2.5590 $area $E 0]
    set segC [create_mooring_segment $type -0.17 0.00 -0.64 -1.28
    0.0 -1.89 $w 1.7620 $area $E 0]
    set segD [create_mooring_segment $type -0.32 0.00 -2.99 -1.28
    0.0 -1.89 $w 1.5590 $area $E 0]
    # links
    create_mooring_link $segB $segA 230.0
    create_mooring_link $segC $segB $segD 0.0
}
```

The function **TdynTcl\_Set\_Mooring\_Displacement** can be used to define the displacement of the end point of a mooring segment.

syntax : TdynTcl\_Set\_Mooring\_Displacement \$seg1 \$fun1 \$fun2 \$fun3

alternative syntax: set\_mooring\_displacement \$seg1 \$fun1 \$fun2 \$fun3

- **seg1** : identifier variable corresponding to the segment
- **fun1** : index of the function describing the time-dependent displacement in OX of the end point of the mooring segment
- **fun2** : index of the function describing the time-dependent displacement in OY of the end point of the mooring segment

**fun3** : index of the function describing the time-dependent displacement in OZ of the end point of the mooring segment

The function **TdynTcl\_Configure\_Mooring\_Segment** can be used to configure advanced options of the cable segments.

syntax : TdynTcl\_Configure\_Mooring\_Segment \$seg1 \$Gk \$Gc \$Gu \$Ms \$Md \$Cd \$Cf \$Cm \$alpha \$bci

alternative syntax: configure\_moorin\_segment \$seg1 \$Gk \$Gc \$Gu \$Ms \$Md \$Cd \$Cf \$Cm \$alpha \$bci

- **seg1** : identifier variable corresponding to the segment
- **Gk** : ground normal stiffness per unit length (Pa/m) for seabed interaction
- **Gc** : ratio of critical damping of ground (-) for seabed interaction
- **Gu** : horizontal damping coefficient (-) for seabed interaction
- **Ms** : static horizontal friction coefficient (-) for seabed interaction
- **Md** : dynamic horizontal friction coefficient (-) for seabed interaction
- **Cd** : tangential drag coefficient of the cable
- **Cf** : normal drag coefficient of the cable
- **Cm** : added mass coefficient of the cable
- **alpha** : Alpha coefficient of the Bossak-Newmark integration scheme
- **bci** : boundary condition for the initial node: 1 if acceleration is imposed, -1 if the node is fixed, 0 if it is free

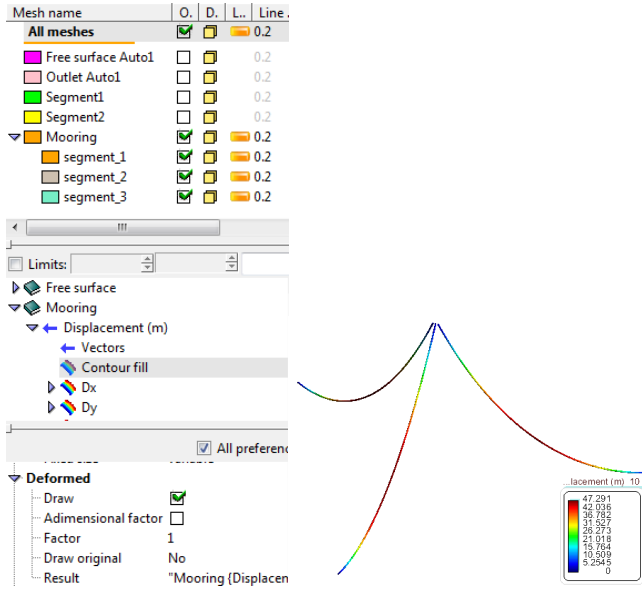
### 3.20.1. Visualization of mooring results

When running a SeaFEM analysis that includes mooring lines definition, specific result files are generated concerning the mooring system:

- **MooringLoads.res** - It contains the forces and moments exerted by the mooring system on the attached body.
- **MooringResults.msh** - It contains an automatically generated mesh of the mooring system.
- **MooringResults.res** - It contains the displacement results of each mooring segment.
- **projectName.MooringData.res** - It contains the tension force components at both ends of each mooring segment.

If **MooringResults.msh** and **MooringResults.res** files exist, they are automatically loaded into the postprocessor graphic user interface. All mooring segments appear grouped within a single Mesh group called "Mooring". Visualization options of each mooring segment can be manipulated individually as can be done with any other mesh. Mooring displacement results can also be animated by drawing the corresponding deformed result as shown in the figure below. To this aim, follow the steps listed in what follows:

- 5.- In the meshes window, check all mooring segments that you want to be plotted.
- 6.- Select the "Mooring > Displacement > Contour fill" option in the results window.
- 7.- Choose the "Deformed > Result > Mooring {Displacement}" option in the preferences window.
- 8.- Finally, activate the "Deformed > Draw" option in the preferences window.



### 3.21. Appendix F: Morison's forces effect

When viscous effects may be advanced to have a significant effect on the dynamic behavior of an offshore structure, Morison's equation can be used to correct the forces evaluated by the diffraction-radiation solver. For this purpose, an auxiliary framework structure, associated to a body must be defined. Based on the information provided by the user, SeaFEM evaluates Morison's forces per unit length acting on this framework structure. After integration along the different elements, the resultant forces are incorporated to the dynamic solver of the rigid body to which the idealized framework structure has been associated.

To this aim, a tcl procedure must be used to define the different elements of the framework structure associated to a body. This procedure allows specifying the reference line of the element as well as the added mass, lift, drag and friction coefficients of the Morison's equation. This function can be called several times to define as many elements as required.

The syntax to define the Tcl procedure to create new elements of the framework structure read as follows:

```
TdynTcl_Add_Morison_Element b t x1 y1 z1 x2 y2 z2 D S C_M C_D C_V C_F C_L
```

where:

- $b$  is and integer index that identifies the body to which the Morison element pertains,
- $t$  is the element type (0 defines a standard element for which the floatability forces are evaluated, and 1 defines a virtual element for which floatability forces are neglected).
- $x1, y1, z1, x2, y2, z2$  are the coordinates of the end points of the cylinder,
- $D$  is the section characteristic linear dimension (the diameter in the case of a cylinder),
- $S$  is the cross section area,
- $C_M$  is the added mass correction coefficient,
- $C_D$  is the non-linear drag coefficient,
- $C_V$  is the linear drag coefficient,
- $C_F$  is the friction coefficient,
- and  $C_L$  is the lift coefficient.

*TdynTcl\_Add\_Morison\_Element* procedure returns the index of the created element of the framework structure.

Based on the information provided in the successive calls to *TdynTcl\_Add\_Morison\_Element* command, SeaFEM evaluates Morison's forces per unit length acting on the framework structure defined by the user. After integration, the resultant forces are incorporated to the dynamics of the rigid body to which the framework structure has been associated. The various (per unit length) contributions to the force are calculated as follows:

$$\begin{aligned} F_M &= (1 - \delta_w)(1 + C_M) \cdot \rho \cdot S \cdot (l \cdot a_w \cdot l) - \rho \cdot S \cdot C_M (l \cdot a_b \cdot l) \\ F_D &= 0.5 \cdot C_D \cdot \rho \cdot D \cdot |l \cdot v \cdot l| \cdot (l \cdot v \cdot l) \\ F_V &= 0.5 \cdot C_V \cdot \rho \cdot D \cdot (l \cdot v \cdot l) \\ F_F &= 0.5 \cdot C_F \cdot \rho \cdot \pi \cdot D \cdot |l \cdot v| \cdot (l \cdot v) \cdot l \\ F_L &= 0.5 \cdot C_L \cdot \rho \cdot D \cdot |l \cdot v| \cdot (l \cdot v) \end{aligned}$$

where  $l$  is the unit vector locally oriented along the element,  $a_w$  is the fluid acceleration of the incident wave,  $a_b$  is the acceleration vector of any point of the body,  $v$  is the relative velocity vector, and  $F_M$ ,  $F_D$ ,  $F_V$ ,  $F_F$  and  $F_L$  are the inertial correction, drag, linear drag, friction and lift components of the force per unit length respectively.

*Remark:* the first term in the right hand side of the  $F_M$  equation includes the Froude-Kriloff force (i.e. the undisturbed wave pressure force) and the diffraction inertial force, while the second term represents the radiation inertial force.

It is important to keep in mind that the added mass contribution calculated using the Morison's equation will be added to the added mass effect that is already accounted for by the diffraction-radiation SeaFEM solver. Hence, the added mass contribution of the Morison's equation must be only used if, for some reason, the user wishes to correct the added mass effect calculated by SeaFEM.

It is emphasized that SeaFEM will always calculate the inertia term by integrating the pressure field on the body surface. This will result in the evaluation of the Froude-Krylov force plus diffraction force. On the other hand, it is well known that viscous effects can reduce the amplitude of the diffraction force, and in most of the cases SeaFEM calculation will result in overprediction of this value. Hence, the added mass contribution of the Morison's equation can be used to improve the computational prediction of this force, by setting a suitable (usually negative) value of  $C_M$ , based on experimental information.

As an example, the following code can be used to add to body number 1 the Morison forces acting on a cylinder whose axis is oriented in the vertical  $z$ -direction. In this simplified example, only added mass and non-linear drag effects are considered, while linear drag, friction and lift corrections are kept null.

```
proc TdynTcl_StartSetProblem {} {
    set idx [TdynTcl_Add_Morison_Element 1 0 0.0 0.0 0.0 0.0 0.0 -0.5 1
    0.7854 -0.3 1.4 0.0 0.0 0.0]
}
```

Finally, it is possible to update the values of the different coefficients of an element of the auxiliary framework structure. The syntax of the Tcl procedure that can be used for this purpose read as follows:

```
TdynTcl_Update_Morison_Element idx C_M C_D C_V C_F C_L
```

where:

idx index of the previously created element of the framework structure (returned by *TdynTcl\_Add\_Morison\_Element*),

- $C_M$  is the updated added mass correction coefficient,
- $C_D$  is the updated non-linear drag coefficient,
- $C_V$  is the updated linear drag coefficient,
- $C_F$  is the updated friction coefficient,
- and  $C_L$  is the updated lift coefficient.

As an example, the following code can be used to nullify the Morison's correction of the framework element of index 1, after 1.0 s of simulation.

As an example, the following code can be used to nullify the Morison's correction of the framework element of index 1, after 1.0 s of simulation.

```
proc TdynTcl_StartNewStep {} {
    if {[TdynTcl_Time]>1.0} {
        TdynTcl_UpdateAdd_Morison_Element 1
        0.0 0.0 0.0 0.0 0.0
    }
}
```

As stated above, the forces evaluated on the auxiliary framework structure are added to the dynamic solver of the different associated bodies. Furthermore, these forces are outputted to the ASCII file 'MorisonLoads.res'. This file contains 24-n columns, where n is the number of bodies in the analysis. For every body, the following 24 values are written every time step:

- $M_x$ : X component of the added mass force acting on the center of gravity of the body.
- $M_y$ : Y component of the added mass force acting on the center of gravity of the body.
- $M_z$ : Z component of the added mass force acting on the center of gravity of the body.
- $MM_x$ : X component of the added mass moment acting on the center of gravity of the body.
- $MM_y$ : Y component of the added mass moment acting on the center of gravity of the body.
- $MM_z$ : Z component of the added mass moment acting on the center of gravity of the body.
- $D_x$ : X component of the total drag force acting on the center of gravity of the body.
- $D_y$ : Y component of the total drag force acting on the center of gravity of the body.
- $D_z$ : Z component of the total drag force acting on the center of gravity of the body.
- $DM_x$ : X component of the total drag moment acting on the center of gravity of the body.
- $DM_y$ : Y component of the total drag moment acting on the center of gravity of the body.
- $DM_z$ : Z component of the total drag moment acting on the center of gravity of the body.

$F_x$ : X component of the friction force acting on the center of gravity of the body.

- $F_y$ : Y component of the friction force acting on the center of gravity of the body.
- $F_z$ : Z component of the friction force acting on the center of gravity of the body.
- $FM_x$ : X component of the friction moment acting on the center of gravity of the body.
- $FM_y$ : Y component of the friction moment acting on the center of gravity of the body.
- $FM_z$ : Z component of the friction moment acting on the center of gravity of the body.
- $L_x$ : X component of the lift force acting on the center of gravity of the body.
- $L_y$ : Y component of the lift force acting on the center of gravity of the body.
- $L_z$ : Z component of the lift force acting on the center of gravity of the body.
- $LM_x$ : X component of the lift moment acting on the center of gravity of the body.
- $LM_y$ : Y component of the lift moment acting on the center of gravity of the body.
- $LM_z$ : Z component of the lift moment acting on the center of gravity of the body.

### 3.22. Appendix G: Analysis advanced configuration

Several advanced (or rarely used) options of the analysis can be configured through the Tcl extension of SeaFEM. As usual, the necessary tcl script procedure must be implemented in a text file, which is further provided as an input argument in the Tcl data section of the data tree once the option to use an external Tcl script is activated.

#### General data ► Tcl data

The specification of advanced options is performed by reimplementing the *TdynTcl\_InitiateProblem* procedure, and simply consists on calling the SeaFEM internal function *configure\_analysis* with two arguments indicating the advanced option's name and the corresponding value. Available configuration options are described in the following list:

- **Mooring\_Elements\_Number**: sets the number of elements for the output of dynamic mooring results. The second argument must be a positive integer number.
- **Mooring\_Initial\_Damping**: activates or deactivates the initial damping for dynamic mooring. The second argument must be 0 (deactivated) or 1 (activated).
- **Mooring\_Solver\_Type**: sets the type of nonlinear solver for dynamic mooring lines. The remaining arguments must be a couple of real values.
- **Morison\_Elements\_Number**: sets the number of elements for the output of Morison element results. The second argument must be a positive integer number.
- **Relax\_Type**: switches on and off the Aitkens relaxation method for wave-structure coupling analysis. The second argument must be 0 (deactivated) or 1 (activated).
- **Solve\_Dif\_Rad**: switches on and off the diffraction-radiation solver. The second argument must be 0 (switch off) or 1 (switch on).
- **Moving\_frame**: updates the coordinates reference frame



according to the main body horizontal displacement. The second argument must be 0 (deactivated) or 1 (activated).

**Example:** the following Tcl script can be used to deactivate the solution of the diffraction-radiation problem (so that only incident waves are taken into account) and to update the coordinates reference frame in accordance to the horizontal displacement of the main body.

```
proc TdynTcl_InitiateProblem { } {  
  configure_analysis Solve_Dif_Rad 0  
  configure_analysis Moving_frame 1  
}
```