xdyn simulator

SIREHNA

1 Description

This program is a mechanical behavior simulator of undeformable solid bodies in a fluid environment. Its main object is the resolution of the motion equations of the body or bodies considered, in the time domain. These motion equations are built on the basis of external forces calculated by specific models included in the software, or which can be added to it. Each model is associated with a specific dataset .

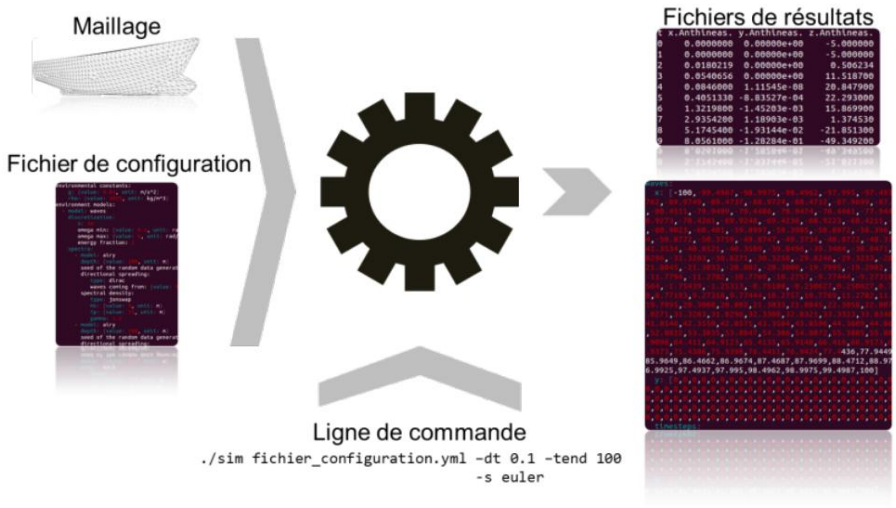
In principle, the simulator makes it possible to deal with any problem of solid body motion. The first force models implemented and provided essentially concern the behavior of ships on the swell.

Regarding the operating mode of the software, it is a command line tool, that is to say that it is launched from a terminal (or an MS DOS command prompt).

The simulator has been built to allow modular use on the command line. To do this, the different areas to configure are separated: the description of the physical problem is made in one or more input files and the description of the simulation (time step, solver, etc.) is made on the command line. In this way, one can easily run simulations of the same problem with different solvers and over different durations, without touching the problem configuration file(s).

The problem description file can be separated into as many files as you want, which allows for example to version the configuration of the physical environment separately from that of the solid(s).

As output, the simulator generates a result file in CSV format containing all the states of the solid(s). It can also generate a YAML file containing wave heights on a mesh. The following figure summarizes the inputs and outputs of the simulator:



Along with this user documentation, there is also implementation documentation describing the software architecture and detailing the APIs to facilitate code maintenance (fixes and feature additions).

2 Benchmarks and conventions

2.1 Benchmarks

In order to know and describe the attitude of one or more bodies in space, it is necessary to place them with respect to a reference frame.

2.1.1 Reference mark (NED)

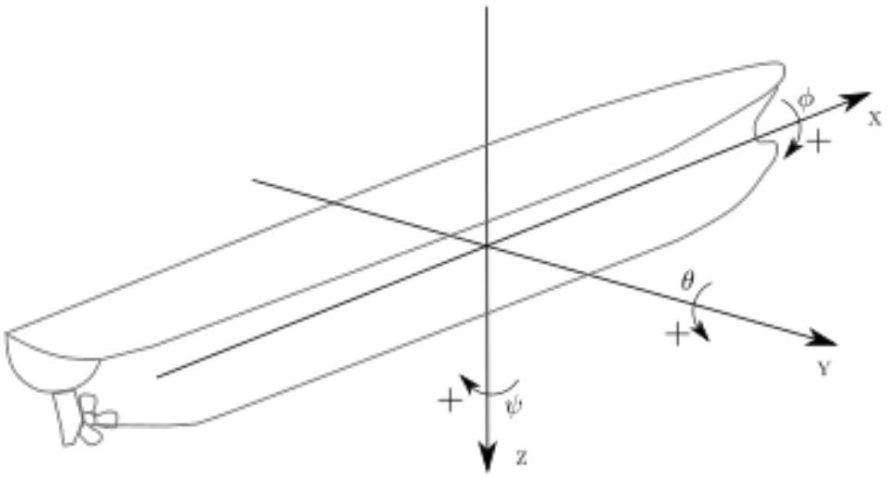
The NED (North-East-Down) marker is used as a reference marker, with an *'O'* reference point and a base pointing the North-East-Down directions. It is used to express the displacements of the bodies of the simulation.

2.1.2 Vessel fix (mobile or “body” or “resolution fix”)

The ship marker corresponds to the marker attached to the ship during the simulation. The reference point of this mark generally corresponds to the center of gravity of the ship. The axes of the ship marker are as follows:

- *'X'* forward

- *'Y* ' to starboard
- *'Z'* down

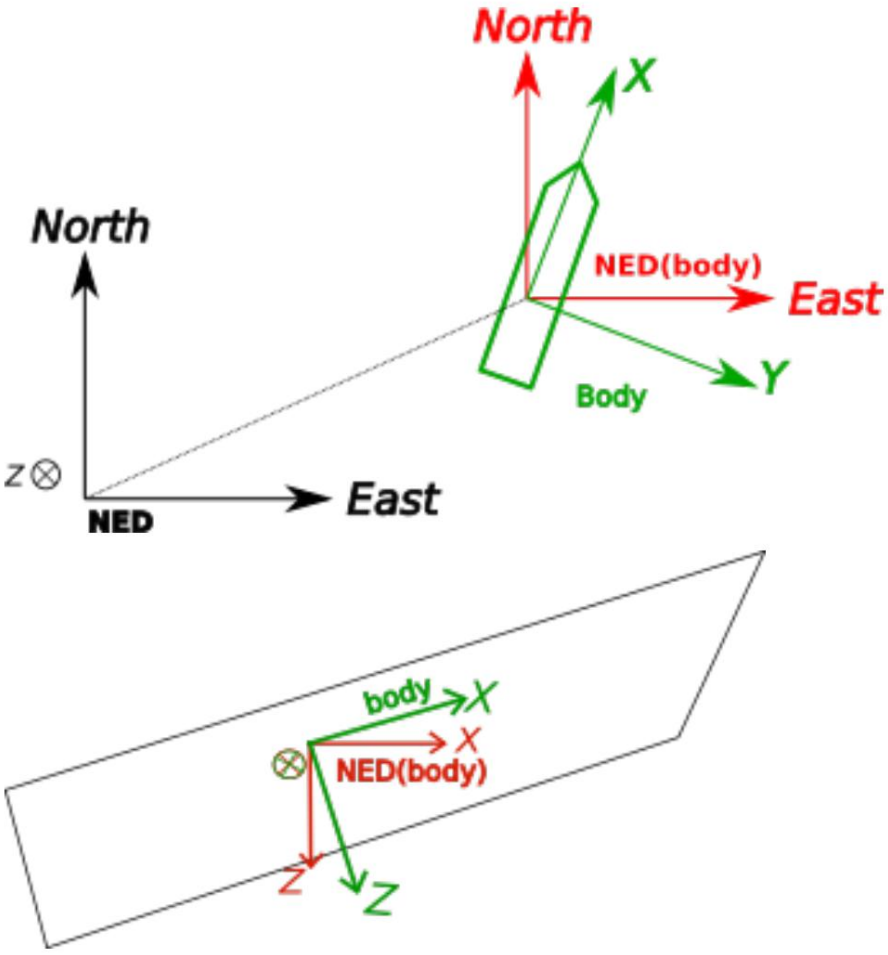


One solves the equations of the movement (fundamental principle of the dynamics) at the origin of this reference, that is to say that all the torsors of effort are moved to the point (0,0,0) of the reference “body ”. Similarly, the inertia and added mass matrices are moved to the resolution point.

2.1.3 Local NED benchmark

When exporting wave fields, neither the NED reference point nor the ship reference point are perfectly suited: indeed, if the mesh on which the swell is calculated is linked to NED, the ship will end up leaving this zone when it will move. If we calculate the wave heights in the ship frame, the area of the grid seen in the NED frame will vary according to the attitude of the ship and, ultimately, for a vertical ship ('ÿ = ÿ /2' for example), the grid projection is a segment.

We therefore define a “local” NED, that is to say a reference point centered at the same point as the ship reference point but simply translated in relation to NED:



This marker is named “NED(body)”. Thus, if the vessel is called “nav1”, the local NED fix will be “NED(nav1)”.

2.1.4 Ship attitude

The attitude of a body makes it possible to know its orientation with respect to a reference. The position is given by the triplet *'(X, Y,* Z)' and the orientation by a triplet of angles *'(ÿ, ÿ,* ÿ)'. The interpretation of this triplet in terms of rotations around the 'x', 'y', *'z'* axes depends on the chosen rotation convention. The orientation can also be expressed in a different way, notably with quaternions (this is also how it is expressed in the xdyn code).

2.2 Conventions d’orientations

This section presents the notations used to define the orientation of an element in space from a triplet of angles *'(ÿ, ÿ,* ÿ)'.

2.2.1 Defining an orientation

To define the composition of rotations giving the orientation of an element in space from a triplet of angles *'(ÿ, ÿ,* ÿ)', several elements must be defined:

- a convention of angles or axes. It allows you to define whether these are the angles or axes that evolve for the notation used.
- a composition of internal or external rotations. It defines whether the composition of the rotations is carried out in relation to a system of fixed axes or in relation to the system of newly modified axes.
- an order in which the rotations are applied. It allows to define completely the composition of rotations.

2.2.2 List of possible conventions

If we choose a convention of angles, then each angle of the triplet respectively defines a rotation around an axis 'X', *'Y* ' or 'Z'. Axes cannot be repeated. It is possible to define 6 angle conventions, which correspond to the permutation of the three axes: *'XY* Z' ,'XZY ' ,'Y *XZ' ,'Y ZX' ,'ZXY* ' ,'ZY X'. For example the rotation *'RY ZX'* applied to the triplet *'(ÿ, ÿ,* ÿ)' will be interpreted as a rotation of *'RY* (ÿ)', followed by the rotation *'RZ* (ÿ)', and ended by the rotation *'RX (ÿ)* '.

If we choose an axis convention, then we modify the order of the axes on which to successively apply the rotations. Repetitions of the axes are then possible, if they do not follow each other. For example, *'XY* X' will be valid, but not *'XXY* '. For example, a ZXY convention defines a composition of rotations.
It is possible to define 12 axis conventions: *'XY* X', *'XY* Z', *'XZX', ' XZY* ', *'Y XY* ', *'Y* XZ', *'Y* ZX', *'Y ZY* ', *' ZXY* ', *'ZXZ', 'ZY* X', *'ZY* Z'. For example the rotation *'RY ZX'* applied to the triplet *'(ÿ, ÿ,* ÿ)' will be interpreted as a rotation of *'RY* (ÿ)', followed by the rotation *'RZ* (ÿ)', and ended by the rotation *'RX (ÿ)* '.

With these angle and axis conventions, there are already 18 combinations. This number is doubled by the fact that the composition of rotations can be internal (intrinsic) or external (extrinsic). If the rotations are composed with respect to the fixed frame, we speak of external composition. If the rotations are composed with respect to the newly created markers, it is called internal composition. It is the latter that is used in the majority of cases. In total, it is thus possible to define 36 conventions.

2.2.3 Summary of the different agreements

The following two tables present the 36 possible conventions:

id	Order	Convention	Composition	Rotation matrix	Remarks
1	angle xyz		Extrinsic	*'RZ(ÿ).RY (ÿ).RX(ÿ)'*	
2	angle xzy 3		Extrinsic	*'RY* (ÿ).RZ(ÿ).RX(ÿ)'	
angle yxz 4 angle			Extrinsic	*'RZ(ÿ).RX(ÿ).RY* (ÿ)'	
yzx 5 angle zxy 6			Extrinsic	*'RX (ÿ).RZ(ÿ).RY* (ÿ)'	
angle zyx 7 angle x			Extrinsic	*'RY* (ÿ).RX(ÿ).RZ(ÿ)'	
y' z' ' 8 angle x z' y'			Extrinsic	*'RX(ÿ).RY (ÿ).RZ(ÿ)'*	
' 9 angle y x' z' ' 10			Intrinsic	*'RX(ÿ).RY (ÿ).RZ(ÿ)'*	
angle y z' x' ' 11 angle			Intrinsic	*'RX(ÿ).RZ(ÿ).RY (ÿ)'*	
z x' y' ' 12 angles z y'			Intrinsic	*'RY* (ÿ).RX(ÿ).RZ(ÿ)' '	
x' '			Intrinsic	*RY* (ÿ). RZ(ÿ).RX(ÿ)'	
			Intrinsic	*'RZ(ÿ).RX(ÿ).RY* (ÿ)'	
			Intrinsic	*'RZ(ÿ).RY (ÿ).RX(ÿ)'*	

id	Order	Convention	Composition	Rotation matrix	Remarks
13	axis x y x	14	Extrinsic	*'RX(ÿ).RY (ÿ).RX(ÿ)'*	Euler
axis x y z			Extrinsic	*'RZ(ÿ).RY (ÿ).RX(ÿ)'*	Gimbal - Tait - Bryan
15	axis x z x	16	Extrinsic	*'RX(ÿ).RZ(ÿ).RX(ÿ)'*	Euler
axis x z y			Extrinsic	*'RY* (ÿ).RZ(ÿ).RX(ÿ)'	Gimbal - Tait - Bryan
17	axis y x y	18	Extrinsic	*'RY* (ÿ).RX(ÿ).RY (ÿ)'	Euler
axis y x z			Extrinsic	*'RZ(ÿ).RX(ÿ).RY* (ÿ)'	Gimbal - Tait - Bryan
19	axis x 20 axis		Extrinsic	*'RX(ÿ).RZ(ÿ).RY* (ÿ)'	Euler
			Extrinsic	*'RY* (ÿ).RZ(ÿ).RY (ÿ)'	Gimbal - Tait - Bryan
21	axis z x y	22	Extrinsic	*'RY* (ÿ).RX(ÿ).RZ(ÿ)'	Euler
axis z x z			Extrinsic	*'RZ(ÿ).RX(ÿ).RZ(ÿ)'*	Gimbal - Tait - Bryan
23	axis z y x	24	Extrinsic	*'RX(ÿ).RY (ÿ).RZ(ÿ)'*	Euler
axis z y z			Extrinsic	*'RZ(ÿ).RY (ÿ).RZ(ÿ)'*	Gimbal - Tait - Bryan
25	axis x y' x' '	26	Intrinsic	*'RX(ÿ).RY (ÿ).RX(ÿ)'* Intrinsic	Euler
axis x y' z' '				*'RX(ÿ).RY (ÿ).RZ(ÿ)'*	Gimbal - Tait - Bryan
27	axis x z' x' '	28	Intrinsic	*'RX(ÿ).RZ(ÿ).RX(ÿ)'* Intrinsic	Euler
axis x z' y' '				*'RX(ÿ).RZ(ÿ).RY* (ÿ)'	Gimbal - Tait - Bryan
29	axis y x' y' '	30	Intrinsic	*'RY* (ÿ).RX(ÿ).RY (ÿ)'	Euler
axis y x' z' '			Intrinsic	*'RY* (ÿ).RX(ÿ).RZ(ÿ)'	Gimbal - Tait - Bryan
31	axis y z' x' '		Intrinsic	*'RY* (ÿ).RZ(ÿ).RX(ÿ)'	Euler

id Order Convention	Composition	Rotation matrix	Remarks
32 axis y z' y' '	Intrinsic	*'RY* (ÿ).RZ(ÿ).RY (ÿ)'	Gimbal - Tait - Bryan
33 axis z x' y' '	34 axis z x' z' '	Intrinsic *'RZ(ÿ).RX(ÿ).RY* (ÿ)' Intrinsic *'RZ(ÿ).RX(ÿ).RZ(ÿ)'*	Euler Gimbal - Tait - Bryan
35 axis z y' x' '	36 axis z y' z' '	Intrinsic *'RZ(ÿ).RY (ÿ).RX(ÿ)'* Intrinsic *'RZ(ÿ).RY (ÿ).RZ(ÿ)'*	Euler Gimbal - Tait - Bryan

where the rotation matrices around the three axes 'X', *'Y* ' and *'Z'* are written

$$'RX\ (a) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(ÿ) & ÿ \sin(ÿ) \\ 0 & \sin(ÿ) & \cos(ÿ) \end{bmatrix}$$

$$'RY\ (a) = \begin{bmatrix} \cos(ÿ) & 0 & \sin(ÿ) \\ 0 & 1 & ÿ \\ ÿ \sin(ÿ) & 0 & \cos(ÿ) \end{bmatrix}$$

$$'RZ\ (a) = \begin{bmatrix} \cos(ÿ) & ÿ \sin(ÿ) & 0 \\ \sin(ÿ) & \cos(ÿ) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.3 Conventions commonly used

Among the set of possible conventions, some are used more than others.

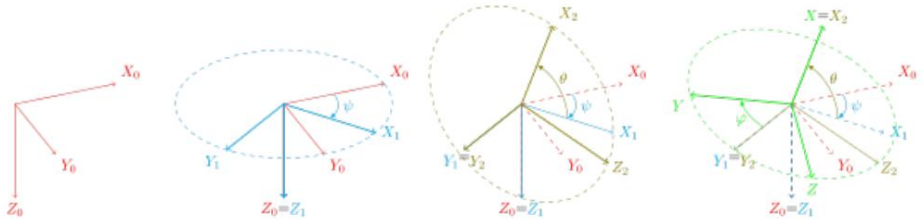
2.3.1 Convention of aeronautical angles

The convention of aeronautical angles (type 2 convention of the AFNOR standard) expressed by the triplet (Roll, Pitch, Yaw) regularly used is referenced id=12 in the table above.

It is understood as follows, we perform a rotation of the yaw angle around the 'Z' axis, followed by a rotation of the attitude angle around the new *'Y'* axis followed by a rotation of the roll angle around the new axis 'Xÿÿ '.

If we express this triplet as follows (Yaw, Roll, Pitch), we obtain id=33 in the table above.

The composition of rotations of these two conventions is shown in Fig.

next 

2.3.2 Convention ParaView

The orientation convention used in the ParaView software is identified by id=11 in the table above.

This composition of rotation is understood as a rotation *'ÿ'* around the axis 'Z', followed by a rotation *'ÿ'* around the new axis *'Xÿ* ' and finally a rotation *'ÿ'* around the new axis *'Y* ^{ÿÿ} '.

2.4 Quaternions

The use of Euler angles poses two main problems:

- Cardan blocking, on the one hand, which is manifested by the loss of a degree of freedom when the axes of two of the three rotations defining the transformation from the NED frame to the ship frame are coincident,
- The periodicity of the angles, d on the other hand, which introduces discontinuities in the states.

A common way to circumvent these problems is to use quaternions which, at the cost of adding an additional state, allow the rotations to be defined unambiguously and uniquely, whatever the angle convention adopted.

During calculations of sea resistance (with a frequency formulation), a linearized benchmark is often used. This benchmark, which can be involved during the link with the hydrodynamic databases resulting from the frequency, is calculated in the following way. By making the assumption that the movements are weak, one carries out a development of the rotations limited to the first order and thus they can be expressed independently compared to the principal axes related to the average position of the ship, in any order. This tag is not used in the current version of xdyn.

2.5 Ship States

The simulator is multi-body in the sense that several bodies can be simulated at the same time. Thus, several mechanically independent bodies can be modeled , with their hydrodynamic interactions, provided that the interaction model is implemented (which can come from a multibody HDB file). Currently, no interaction effort or kinematic binding are implemented.

Each body has states, making it possible to reconstitute its movement exactly. These states are:

- The position of the body relative to the origin of the NED projected in the frame body is noted *'p* n = *[x, y,* z] • The ^{T} and is expressed in meters.
attitude of the body is noted 'ÿ = *[ÿ, ÿ, ÿ]* ^{T} and is defined in the previous paragraph. In practice, we rather use quaternions *'q = [qr, qi , qj ,* qk]' internally in the simulator for the integration of the equations of motion, but the reasoning presented in this documentation is easily transposed. • The speed of translation of the body with respect to the fixed
reference NED, projected in the reference of the body (or "body") is noted ^{b}*'v = [u, v,* w] and ^{T} is expressed in m/s.

- The rotational speed vector of the "body" frame with respect to the frame = *[p, q,* r] NED, projected in the "body" frame, is denoted ^{b}*'ÿ* and is ex^{T}pressed in rad/s.

- The forces applied to the ship and projected into the "body" frame are noted : *'f* ^{b} = *[X, Y,* Z] • ^{T} '. They are expressed in N.
The moments applied to the ship and projected in the "body" frame are noted: *'mb = [K, M,* N] ^{T} '. They are expressed in Nm

2.6 Definition of simulated bodies

The bodies section of the YAML file contains a list of bodies, each beginning with a dash:

```
bodies:
    - name: ....
        ...
    - name: ....
        ...
```

Each body includes:

- a name (section name) •
possibly a mesh (section mesh) • the position of the
reference body relative to the mesh (section position of body frame relative to mesh) • its initial conditions (sections initial position
of body frame relative to NED and initial velocity of body frame relative to

DOWN)
- data necessary for the kinetic calculations (dynamics section) • the list of forces to which it is subjected (external forces sections). • optionally, forced states.

2.6.1 Complete example

```
bodies: # All bodies have NED as parent frame
  - name: TestShip
    mesh: test_ship.stl
    position of body frame relative to mesh: frame: mesh
        x: {value:
        9.355, unit: m} y: {value: 0, unit:
        m} z: {value: -3.21, unit: m}
        phi: {value: 0, unit: rad} theta:
        {value: 0, unit: rad} psi: {value: 0,
        unit: rad} initial position of body frame
        relative to NED: frame: NED


        x: {value: 0, unit: m} y:
        {value: 0, unit: m} z: {value:
        -5, unit: m} phi: {value: 0, unit:
        deg} theta: {value: -.0058, unit:
        rad} psi: {value: 0, unit: deg} initial velocity
        of body frame relative to NED:
    frame: TestShip u: {value: 0, unit: m/s} v: {value: 0, unit: m/s}
        w: {value: 0, unit: m/
        s} p: {value: 0, unit: rad/s} q:
        {value: 0, unit: rad/s} r: {value:
        0, unit: rad/s} dynamics:
        hydrodynamic forces calculation
        point in body frame:


            x: {value: 0.696, unit: m} y: {value:
            0, unit: m} z: {value: 1.418,
            unit: m} centre of inertia:

            frame: TestShip x:
            {value: 0.258, unit: m} y: {value:
            0, unit: m} z: {value: 0.432,
            unit: m}
        mass: {value: 253.31, unit: tonne} # Caution: 'ton' is the british ton which is 907  rigid body inertia matrix at the
        center of gravity and projected in the body frame:
            row 1: [253310,0,0,0,0,0] row 2:
            [0,253310,0,0,0,0] row 3:
            [0,0,253310,0,0,0] row 4:
            [0,0,0,1.522e6,0,0] row 5:
            [0,0,0,0,8.279e6,0]
```

```
            row 6: [0,0,0,0,0,7.676e6]
        added mass matrix at the center of gravity and projected in the body frame:
            row 1: [3.519e4,0,0,0,0,0] row 2:
            [0,3.023e5,0,0,0,0] row 3:
            [0,0,1.980e5,0,0,0] row 4:
            [0,0,0,3.189e5,0,0] row 5:
            [0,0,0,0,8.866e6,0] row 6:
            [0,0,0,0,0,6.676e6]
external forces:
  - model: gravity -
  model: non-linear hydrostatic (fast) blocked dof:

    from CSV:
      - state: u
        t: T
        value: PS
        interpolation: spline filename:
        test.csv
    from YAML:
      - state: p t:
        [4.2] value:
        [5]
        interpolation: piecewise constant
```

2.6.2 Naming the solid

The name of the solid is important since by defining a solid, one implicitly defines the reference mark which is attached to it (the “body” reference mark, cf. reference marks documentation).

One can then refer to it, in particular for the postprocessings.

2.6.3 Use of a mesh

For the forces integrated on the hull (for example, the non-linear hydrostatic forces and the Froude-Krylov forces), it is necessary to define a mesh.

2.6.3.1 Definition of the mesh file The mesh section is optional.
If one chooses to use it, it must contain an STL file name containing the surface mesh of the ship. STL ASCII and STL binary formats are supported. This path must be given relative to where the simulator is launched.

For example, if the simulator executable is in the A/B/C directory, the m.stl mesh is in A and we launch the executable from B, we will write:

```
mesh: ../m.stl
```

2.6.3.2 Mesh orientation convention Although the STL format contains the normal of each face, xdyn does not use this information. Instead , the normal is calculated by the “right hand rule” (according to the STL standard), i.e. the normal is oriented directly with respect to the local base formed by the vectors between the points ordered from the face.

Also, xdyn assumes that the normals are oriented towards the fluid. A warning is issued if a significant part of the faces is oriented towards the barycenter of the mesh, but this may be the case even when the normals are well oriented towards the fluid with complex geometries (several shells for example). It is thus up to the user to check that the normals are correctly oriented towards the fluid.

2.6.3.3 Passage of the reference mesh to the reference body The origin of the reference “body” (which is the reference in which the assessment of the forces is carried out) is specified compared to the reference of the mesh. In practice, *'x, y,* z' can define the position of the center of gravity in the mesh reference and *'ÿ, ÿ,* ÿ' define the rotation allowing to pass from the mesh reference to the body reference (according to the convention chosen in the section spins).

```
position of body frame relative to mesh: frame: mesh

    x: {value: 0, unit: m} y: {value:
    0, unit: m} z: {value: -10, unit:
    m} phi: {value: 1, unit: rad} theta:
    {value: 3, unit: rad} psi: {value: 2,
    unit: rad}
```

2.6.4 Champs dynamics

The dynamics section makes it possible to describe the inertia of the solid. It is made up of five subsections:

- hydrodynamic forces calculation point in body frame est le point calculation of hydrodynamic forces
- center of inertia (if the “body” reference is not at the center of mass) • mass containing the mass of the body considered • rigid body inertia matrix at the center of gravity and projected in the body frame defining the inertia matrix • added mass matrix at the center of gravity and projected in the body frame for added masses.

2.6.4.1 Position of the center of inertia The center of inertia section looks like this:

```
centre of inertia:
    frame: TestShip
```

x: {value: **0,** unit: **m} y:**
{value: **0,** unit: **m} z:** {value:
0, unit: **m}**

The frame field must contain either NED, or a solid name, or mesh(<solid name>) depending on the coordinate system chosen to express the coordinates of the center of gravity.

2.6.4.2 Definition of the mass of the solid The section mass contains the mass of the solid, used in particular for the gravity model:

mass: {value: 1000, unit: tonne}

Warning: if ton is specified, the system will use the British ton which is equal to 907.185 kg.

2.6.4.3 Matrix of inertia The matrix of inertia is not normalized and one does not carry out a change of reference.

rigid body inertia matrix at the center of gravity and projected in the body frame:
row **1: [253310,0,0,0,0,0]** row **2:**
[0,253310,0,0,0,0] row **3:**
[0,0,253310,0,0,0] row **4:**
[0,0,0,1.522e6,0,0] row **5:**
[0,0,0,0,8.279e6,0] row **6:**
[0,0,0,0,0,7.676e6]

The calculations are performed assuming that the terms *'aij* ' of the inertia matrix are in SI unit, i.e.:

• in 'kg' for the terms *'aij* ' with '1 ÿ *i, j* ÿ 3', • in 'kg × m
for the terms *'äij* ' with '4 ÿ *i, j* ÿ 6', • in 'kg× m' for the two
extra-diagonal 3x3 blocks representing the terms
crosses of the inertia matrix.

2.6.4.4 Added inertias The added mass matrix is defined in the same way:

added mass matrix at the center of gravity and projected in the body frame:
row **1:** [3.519e4,0,0,0,0,0] row **2:**
[0,3.023e5,0,0,0,0] row **3:**
[0,0,1.980e5,0,0,0] row **4:**
[0,0,0,3.189e5,0,0] row **5:**
[0,0,0,0,8.866e6,0] row **6:**
[0,0,0,0,0,6.676e6]

It is in the dynamics section and not in the external forces section (although it is an effort model, proportional to acceleration) because

that this model of effort is the subject of a particular treatment: it appears in the left member of the fundamental equation of the dynamics

M\ddot{X} = \sum F_i

for reasons of numerical stability (the effort depends on the accelerations which must precisely be calculated by the resolution of the fundamental equation of the dynamics). The matrix of added masses is however not equivalent to an additional mass, because the terms of Coriolis and centripetal which would correspond are not taken into account.

It is also possible to extrapolate the masses added at infinite pulsation from an HDB file. To do this, we write (to read from the test_ship.hdb file):

added mass matrix at the center of gravity and projected in the body frame:
from hdb: test_ship.hdb

One can also read this matrix from a PRECAL_R file. The section used is [added_mass_damping_matrix_inf_freq], present on condition of having activated the calcAmasDampCoefInfFreq key (section sim > parHYD > calcAmasDampCoefInfFreq of the PRECAL_R input XML file):

added mass matrix at the center of gravity and projected in the body frame: from raodb:
ONRT_SIMMAN.raodb.ini

If the added masses come from a file (PRECAL_R or AQUA+/DIODORE), the frame and row keys must not be specified (the program stops with an error if this is done in order to avoid any ambiguity) . As for STL files, the path to the file (HDB or PRECAL_R) is relative to where the executable is launched.

In the HDB files, the section corresponding to the added masses is [Added_mass_Radiation_Damping], subsections [ADDED_MASS_LINE_*] where * is 1 to 6 and indicates the line of the matrix. The value used by xdyn is the mass matrix added to the minimum period defined in the HDB file (no extrapolation is made).

In the PRECAL_R files the matrix of added masses with infinite pulsation appears in the section [added_mass_damping_matrix_inf_freq]. PRECAL_R directly calculating asymptotic values with infinite pulsations, no interpolation or extrapolation is necessary.

2.6.5 Forcing degrees of freedom

It is possible to force the values of the following degrees of freedom: U, V, W, P, Q, R for each body. Although it is theoretically possible to force attitude and position (x, y, z, phi, theta, psi), xdyn does not allow this because it would also involve forcing velocities (u, v, w, p , q, r). An additional complexity appears when one wishes to force the Euler angles (for example a heading 'ÿ')

since this involves solving a minimization problem to find the quaternions giving the forced Euler angles. Forcing positions, attitudes and velocities at the same time can lead to incoherent situations that should be detected, while forcing the velocities u, v, w, p, q, r does not present these difficulties.

To force the degrees of freedom, we add the following (optional) section to the body section:

```
blocked dof:
    from CSV:
      - state: u
        t: T
        value: PS
        interpolation: spline filename:
        test.csv
    from YAML:
      - state: p t:
        [4.2] value:
        [5] interpolation:
        piecewise constant
```

Either the states are given directly in the YAML file, or they are read from a CSV file.

- state: name of the state to force. u, v, w, p, q or r.

If the state values are in the YAML:

- value: Value of the forced state for each instant (in the example above, ' 4.2). ÿ • t: instants at u=5 for t which the state is defined • interpolation: type of interpolation to be performed. constant piecewise, linear ou spline.

If the values are read from a CSV file:

- filename: name of the file containing the forced values. The file path is relative to where the simulator is launched.
- value: name of the column containing the values to read • t: name of the column containing the time • interpolation: type of interpolation to perform. constant piecewise, linear ou spline.

For values of t outside the interval [tmin,tmax], the state is assumed to be free (not forced).

When a degree of freedom is forced, its derivative is also forced before the call to the solver. Effort models are called with the forced states and the states are forced again after the solver call, so as not to embed the forced states.

It is possible to recover in the outputs the difference between the real force and the force making it possible to preserve the forcings, in other words it is possible to recover

$$(M+M_a)\dot{X_{\textrm{blocked}}} - \sum F_i$$

To do this, we use in the 'output' section the following keys (if the body is called 'TestShip):

- Fx(blocked states,TestShip,TestShip) • Fy(blocked states,TestShip,TestShip) • Fz(blocked states,TestShip,TestShip) • Mx(blocked states,TestShip,TestShip) • My(blocked states,TestShip,TestShip) • Mz(blocked states,TestShip,TestShip)

It should be noted that these forces are expressed in the reference BODY.

2.7 HDB files resulting from a frequency calculation

2.7.1 Format HDB

The HDB (Hydrodynamic DataBase) format is the standard format of the École Centrale The AQUA+ software (developed and used internally by de Nantes and Diodore software. SIREHNA, parts of which have been included in the NEMOH software) also has an output to create an HDB file. These files can be used by xdyn to calculate:

- added masses (see previous paragraph) • radiation damping • diffraction forces, calculated from transfer functions (RAO)

2.7.2 Format PRECAL_R

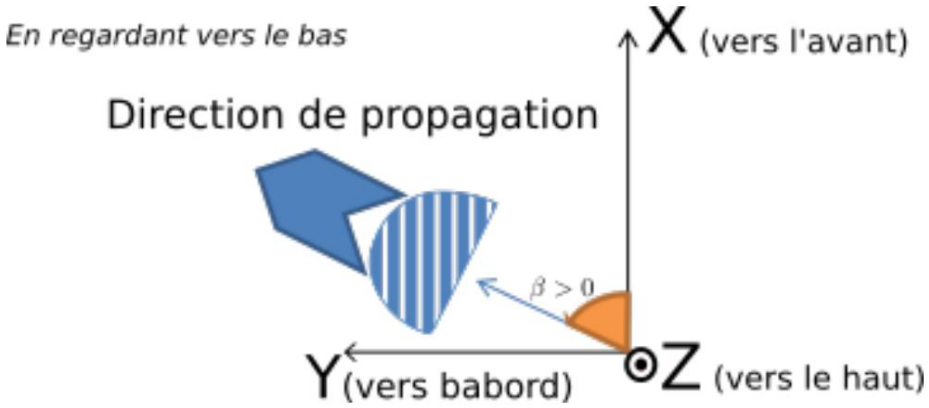
xdyn can read the outputs of PRECAL_R and extract:

- the added masses • the write-off depreciation

PRECAL_R is software developed by Marin as part of the Cooperative Research Ships (CRS) group. It solves seakeeping problems (diffraction and radiation) by a potential method like AQUA+. It can be used to calculate the movements and forces due to waves on any hull. It is, among other things, able to take stabilization into account, offers stochastic equilinearization and allows coupling with the structure. PRECAL_R will eventually evolve to SEACAL which will improve the taking into account of the speed of advance, while retaining the same structure of hydrodynamic data exportable, among others, to xdyn.

2.7.3 HDB File Conventions

HDB files do not specify their calculation marker or expression points.
Rather than letting the user specify it (with the risk of error that entails) or detecting it automatically, it is assumed for xdyn that the HDB files were generated with the conventions of the AQUA+ software (“z up” convention ”). The frame in which all the matrices of all the HDB files read by xdyn are expressed is therefore: x longitudinal, y towards port and z upwards.



The RAOs depend on the conventions (reference, origin of the phases, origin vs. propagation, sign of the angles). However, the convention used does not appear in the HDB format. As the CAD we currently use come exclusively from the AQUA+ software, in the current version xdyn assumes that the convention used is that of AQUA+.

The difference between the convention of xdyn and that of AQUA+ is that the *'z'* axis is ascending for AQUA+ and descending for xdyn. The angle of the swell represents in both cases a direction of propagation (and not of origin). The potential of wave speeds is, in xdyn as in AQUA+:

‘ÿ(x, *y, z,* t) = ÿ $\frac{gÿa}{oh}\frac{\cosh(k\cdot(hÿz))}{\cosh(k\cdot h)}$ cos(k · *(x* · cos(ÿ) + *y* · sin(ÿ)) ÿ *ÿ* · *t* + ÿ)'

All data from HDB files are given in the *z-up convention :* therefore, a change of frame must be made (rotation of *'ÿ'* around the 'X' axis) to put them in the xdyn frame *(z down).* The basic change matrix is:

R_X(\pi)=\left[\begin{array}{ccc} 1 & 0 &0\\0&-1&0\\0&0&-1\end{array}\right]

- For vectors

\left[\begin{array}{c}x_d\\y_d\\z_d\\\end{array}\right]_{\mbox{xdyn}}=\left[\begin{array}{c

- For the matrices (added masses, depreciation. . .) If *“M = ((mij))”* indicates a matrix expressed in the reference AQUA+ and *“Md”* the same matrix expressed in the reference xdyn, one a:

M_d = \left[\begin{array}{cc}R_X(\pi)&S(AB)R_X(\pi)\\0&R_X(\pi)\end{array}\right]^\top M \l

2.7.4 PRECAL_R file convention

The conventions are the same as for the HDB files (expression of the forces at the center of gravity, Z mark upwards, definition of the wave propagation angle).

Whatever velocity potential is chosen as the solution to the equation of Laplace describing a monochromatic swell (several potentials can be a solution), the phase shifts between the excitation due to the swell and the responses of the ship calculated by the potential codes will be the same, as illustrated in the figure below:

“‘python echo=False, results=‘raw’, name=‘tutorial_01_plot_results’ import matplotlib.pyplot as plt import numpy as np from numpy import cos, pi, sin

Tp = 4 omega = *2pi/Tp phi = pi/4 t = np.arange(0, 2* Tp, 0.1) # start,stop,step aqua_plus_wave_elevation = -sin(omegat) *aqua_plus_response = -2* sin(omegat + *phi) precal_r_wave_elevation = cos(omegat)*

precal_r_wave_response = *2cos(omegat* + phi) plt.plot(t, aqua_plus_wave_elevation, ‘r-’) plt.plot(t, aqua_plus_response, ‘r–’) plt.plot(t, precal_r_wave_elevation, ‘b-’) plt.plot(t, precal_r_wave_response, ‘b–’) plt.xlabel(‘Temps (s)’) plt.title(‘Déphasage entre PRECAL_R et AQUA+’) plt.legend([‘Élévations AQUA+’, ‘Réponse AQUA+’, ‘Élévations PRECAL_R’, ‘Réponse PRECAL_R’]) plt.arrow(2.5, 1, 0.5, 0, shape=‘left’, color=‘black’, linestyle=‘-’) plt.axvline(x=2.5, color=‘red’, linestyle=‘–’, lw=1) plt.arrow(3.5, 1, 0.5, 0, shape=‘left’, color=‘black’, linestyle=‘-’) plt.axvline(x=3.5, color=‘blue’, linestyle=‘–’, lw=1) plt.text(2.65

Consequently, xdyn uses the results of AQUA+ and PRECAL_R in the same way to rebuild the forces coming from RAO (cf [detail of the calculation of

In addition, all the matrices read since a file HDB or PRECAL_R (added masses and damping of radiation) undergo the change of reference described in the following paragraph.

Transport of inertia and damping matrices read from the HDB file

The formula of Huygens makes it possible to translate the inertias of the center of gravity towards an unspecified point. One is interested here in the displacement and the change of reference of a matrix of generalized inertia (own inertias and added inertias).
Compared to the formula of Huygens, the difference comes from the fact that the inertia is not identical on all the axes (mathematically, the matrix can thus be any symmetric, positive and definite bilinear form).

Either

```math
S(\lambda)=\left[\begin{array}{ccc} 0&-\lambda_3&\lambda_2\\\lambda_3&0&-\lambda_1\\-\lambd
```

the matrix of the vector product by *'ÿ* ^{ÿ} = [ÿ1, ÿ2, ÿ3]' : 'ÿa ÿ Rÿ , $ÿ \times a$ = S(ÿ)a'

To transport a 6x6 matrix in generalized coordinates (mass, added mass or damping) from point *'A'* to point 'B', use:

M_B=\left[\begin{array}{cc}I&S(AB)\\0&I\end{array}\right]^\top M_A \left[\begin{array}{cc}I

This formula is a generalization of the Huygens formula.

By combining with a base change (from base *'a'* to base 'b') by the rotation matrix ' *aRb'* from *'b'* to *'a'* we obtain the more general expression:

{}^bM_B=\left[\begin{array}{cc}{}^a R_b&S(AB){}^a R_b\\0&{}^a R_b\end{array}\right]^\top {}

This formula makes it possible to carry out at the same time the transport of a matrix of generalized inertia 6x6 from a point *'A'* to a point *'B'* and the change of its reference mark of expression from *'a'* towards 'b' .

The inertia matrix is most often expressed at the center of gravity. However, there is no guarantee of this in HDB files. In the general case, it is therefore necessary to carry out a transport, but it is assumed in xdyn that there is no transport to be carried out and therefore that the only operation for the conversion is the passage of a z mark upwards (HDB) to a down z mark (xdyn). The above formula then simplifies to:

```
{}
^bM_B=\left[ \begin{array}
{cccccc}
1&0&0&0&0&0\\
0&-1&0&0&0&0\\
0&0&-1&0&0&0\\
0&0&0&1&0&0\\
0&0&0&0&-1&0\\
0&0&0&0&0&-1\
\ \end{array} \right]^\top {}
    ^aM_A
\left[ \begin{array}{cccccc}
1&0&0&0&0&0\\
0&-1&0&0&0&0\\
0&0&-1&0&0&0\\
0&0&0&1&0&0\\
0&0&0&0&-1&0\\
0&0&0&0&0&-1\\
  \end{array}
```

$$\right]$$

By taking the notation

$$M_{\mbox{HDB}} =$$

$$\left[\begin{array}{cccccc}
m_{11} & m_{12} & m_{13} & m_{14} & m_{15} & m_{16}\\ m_{21} & m_{22} &
m_{23} & m_{24} & m_{25} & m_{26} \\ m_{31} & m_{32} & m_{33} & m_{34} &
m_{35} & m_{36} \\ m_{41} & m_{42} & m_{43} & m_{44} & m_{45} & m_{46}\\
m_{51} & m_{52} & m_{53} & m_{54} & m_{55} & m_{56} \\ m_{61} & m_{62} &
m_{63} & m_{64} & m_{65} & m_{66} \\ \end{array} \right]$$

we obtain :

$$M_{\mbox{xdyn}} =$$

$$\left[\begin{array}{cccccc}
m_{11} & -m_{12} & -m_{13} & m_{14} & -m_{15} & -m_{16}\\ -m_{21} & m_{22} &
m_{23} & -m_{24} & m_{25} & m_{26} \\ -m_{31} & m_{32} & m_{33} & -m_{34} &
m_{35} & m_{36} \\ m_{41} & -m_{42} & -m_{43} & m_{44} & -m_{45} & -m_{46}\\
-m_{51} & m_{52} & m_{53} & -m_{54} & m_{55} & m_{56} \\ -m_{61} & m_{62} &
m_{63} & -m_{64} & m_{65} & m_{66} \\ \end{array} \right]$$

For the torsors, we obtain the following change:

$$\tau_{\mbox{HDB}} =$$

$$\left[\begin{array}{c}
F_X\\
F_Y\\
F_Z\\
M_X\\
M_Y\\
M_Z\\
\end{array}
\right]$$

$$\tau_{\mbox{xdyn}} =$$

$$\left[\begin{array}{c}
F_X\\
-F_Y\\

-F_Z\\
M_X\\
-M_Y\\
-M_Z\\
\end{array} \right]

Cf. *SimBody Theory Manual,* Release 3.1, March, 2013, page 137, §12.3.1, Rigid body shift of rigid body spatial inertia

2.7.5 Structure of HDB files

HDB files are outputs generated by Fortran 77 code. No formal specification of this format has been found. To be able to import them into xdyn, their EBNF grammar was deduced from the analysis of several files (reverse-engineering). This grammar, which makes it possible to parse all the HDB files that have been provided to us so far, can be represented as follows (without guarantee of completeness):

```
ast                                  = string-key | value-
                                      key | vector-section
                                      | matrix-section | list-of-matrix-
                                      sections | list-of-matrix-
                                      sections-with-id; *" "

str                                  = character . +extended-char . , +extended-char;
header                               = "[" = . str . "]"; ,
string-key  value-                   header  =  . str . -eol;
key  values                          header  =  . double  , -eol; +double  ,
                                     double_ ,  = header  -eol; +(double  eol)  , -eol; ,
vector-section                       . eol . +(*(eol, values))  , -eol; +matrix-section  ,
matrix-section                       = header . eol . -eol; , *(eol, values)  , -eol; +(header  , +(*(eol,
list-of-matrix-sections              = header . eol . values) double  eol
section-with-id                      = header . double . eol
list-of-matrix-sections-with-id = header . eol . . . .
character                            = "A" | "B" | "C" | "D" | "E" | "F" | "G"
                                      | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                                      | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
                                      | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k"
                                      | "l" | "m" | "n" | "O" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "_";
extended-char                         = character | "-" | "+"
```

HDB files therefore consist of a list of elements, these elements being able to be of type string-key, value-key, vector-section,

matrix-section, list-of-matrix-sections or list-of-matrix-sections-with-id. xdyn does not take into account the order in which these elements are (but no guarantee can be provided for any other tools using HDBs).

2.7.5.1 Exemple de “string-key”

```
[SOFT] AQUA+
```

2.7.5.2 Example of “value-key”

```
[FORWARD_SPEED] 0.00
```

2.7.5.3 Example of “vector-section”

```
[List_calculated_periods] 4.00
  64.00
 125.00
```

2.7.5.4 Example of “matrix-section”

```
[Mass_Inertia_matrix]
 8.635000E+06 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+000.000000E+00
 0.000000E+00 8.635000E+06 0.000000E+00 0.000000E+00 0.000000E+000.000000E+00
 0.000000E+00 0.000000E+00 8.635000E+06 0.000000E+00 0.000000E+000.000000E+00
 0.000000E+00 0.000000E+00 0.000000E+00 4.149000E+08 0.000000E+000.000000E+00
 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00 1.088000E+100.000000E+00
 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00 1.088000E+10
```

2.7.5.5 Exemple de “list-of-matrix-sections”

```
[Added_mass_Radiation_Damping]
[ADDED_MASS_LINE_1]
    4.00 8.770898E+04 0.000000E+00 2.122348E+05 0.000000E+00 1.844677E+07 0.000000E+00
   64.00 2.356385E+05 0.000000E+00 3.063730E+05 0.000000E+00 6.103379E+07 0.000000E+00
  125.00 2.334437E+05 0.000000E+00 3.058729E+05 0.000000E+00 6.044752E+07 0.000000E+00
[ADDED_MASS_LINE_2]
    4.00 0.000000E+00 2.406631E+06 0.000000E+00 -1.298266E+06 0.000000E+00 2.894931E+07 64.00 0.000000E+00
   7.130066E+06 0.000000E+00 -3.516670E+06 0.000000E+00 9.536960E+07 125.00 0.000000E+00 7.100760E+06
 0.000000E+00 -3.493418E+06 0.000000E+00 9.499156E+07
```

2.7.5.6 Exemple de “list-of-matrix-sections-with-id”

```
[FROUDE-KRYLOV_FORCES_AND_MOMENTS]
[INCIDENCE_EFM_MOD_001] 0.000000
   4.00 6.452085E+04 0.000000E+00 3.775068E+05 0.000000E+00 2.630894E+07 0.000000E+00
  64.00 8.296234E+04 0.000000E+00 2.098381E+07 0.000000E+00 1.280202E+08 0.000000E+00
125.00 2.179682E+04 0.000000E+00 2.105635E+07 0.000000E+00 1.258710E+08 0.000000E+00
```

[INCIDENCE_EFM_MOD_001] 30.00000
4.00 5.988292E+04 3.453055E+04 5.220861E+05 6.375514E+05 2.533077E+07 1.149621E+06
64.00 7.185571E+04 4.148640E+04 2.098683E+07 7.927488E+04 1.274491E+08 3.394846E+04
125.00 1.887675E+04 1.089865E+04 2.105658E+07 2.081770E+04 1.258309E+08 2.938749E+03
[INCIDENCE_EFM_PH_001] 0.000000
4.00 5.079494E-01 1.570796E+00 1.171722E+00 1.570796E+00 1.426003E+00 1.570796E+00
64.00 -3.141362E+00 1.570796E+00 -1.576680E+00 1.570796E+00 -1.768797E+00 1.570796E+00
125.00 -3.141476E+00 1.570796E+00 -1.572334E+00 1.570796E+00 -1.623406E+00 1.570796E+00

2.7.6 Sections used by xdyn

The HDB sections currently used by xdyn are:

- Added_mass_Radiation_Damping for radiation damping and added mass matrices,
- DIFFRACTION_FORCES_AND_MOMENTS for RAO of diffrac forces tion.

2.8 Benchmark of hydrodynamic calculation

The forces of viscous damping and resistance to advancement are calculated in a reference called **reference of hydrodynamic calculation,** which is a reference translated compared to the reference body. The center of this frame is a point defined (in the body frame) as follows:

- Its abscissa *'x'* is that of the center of the surface resulting from the projection of the mesh on the plane *'(x,* z)'
- Its ordinate *'y'* is zero • Its altitude *'z'* is that of the center of the surface resulting from the projection of the mesh on the plane *'(x,* y)'

This point is, in general, distinct from the center of gravity and the center of volume. It is defined in the dynamics/hydrodynamic forces calculation point in body frame section of the YAML file:

```
hydrodynamic forces calculation point in body frame:
    x: {value: 0.696, unit: m} y: {value:
    0, unit: m} z: {value: 1.418,
    unit: m}
```

the ' *localTbody'* the transformation allowing to convert coordinates One notes in reference body in coordinates of the same point expressed in the reference of hydrodynamic calculation. ' *localTNED'* is that allowing to convert coordinates in the NED reference into coordinates of the same point expressed in the reference of hydrodynamic calculation.

This reference should be distinguished from that used in the hydrodynamic database (HDB files), used for the expression of the damping matrices

of radiation, the RAO of effort (for the calculation of the diffraction efforts) and the added masses.

2.9 Rotations and environmental constants

2.9.1 Rotations

The rotation convention makes it possible to find the rotation matrix of one frame with respect to another, given a triplet *'(ÿ, ÿ,* ÿ)'. The convention used is described in the rotations section:

rotations convention: [psi,theta',phi'']

This line is interpreted as follows: given a triplet *(ÿ, ÿ,* ÿ), we construct the rotation matrices by first performing a rotation of angle *'ÿ'* around the Z axis, then a angle rotation *'ÿ'* around the Y axis of the reference frame previously transformed, then an angle rotation *'ÿ'* around the X axis of the reference frame thus obtained.

If we had noted:

rotations convention: [z,y',x'']

we would first have a rotation of angle *'ÿ'* around the Z axis, then a rotation of angle *'ÿ'* around the new axis Y, then a rotation *'ÿ'* around the new axis X.

Apostrophes are used to indicate compositions of rotations relative to the new axis system, and therefore internal composition. Thus [x,y',z''] will designate a rotation around X, followed by a rotation around the new Y axis, called Y' and ended by a rotation around the new Z axis, called Z' '. The double apostrophe refers to the second mark used for rotation composition.

The convention rotations list always has three elements. The second element is always different from the first. The third is either different from the other two or equal to the first.

The convention of aeronautical angles, frequently (and incorrectly) denoted “Euler angles” (yaw 'ÿ', pitch 'ÿ', roll 'ÿ'), is defined as follows

rotations convention: [psi, theta', phi'']

The orientation convention used in the ParaView software is given by:

rotations convention: [psi,phi',theta'']

For more details on the conventions of angles and axes, refer to the detailed documentation.

The definition of a marker in xdyn is done from a known marker (NED or body). We define the change of coordinates to go from a known marker to a new marker as follows in the YAML file:

- frame the name of the known frame,
- x ,y ,z: the triplet of position where each position is defined by the dictionary of the keys value and unit, •

phi ,theta ,psi, the triplet of orientation whose interpretation in terms of rotation matrices depends on the chosen orientation convention

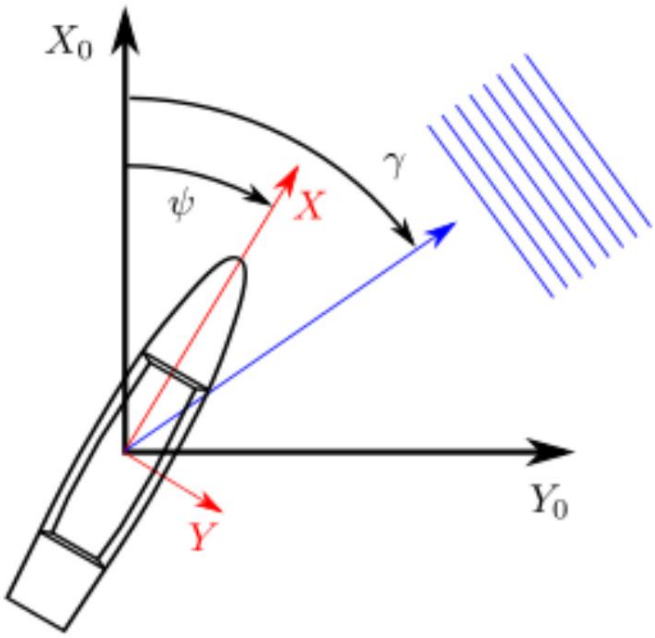
```
frame: NED
x: {value: 0, unit: m} y: {value:
0, unit: m} z: {value: 0, unit:
m} phi: {value: 0, unit: rad}
theta: {value: 0, unit: rad} psi: {value:
0, unit: rad}
```

This description is in particular used to define the change of coordinate data between the coordinate system of the mesh and the reference system of the body (by the key position of body frame relative to mesh) and the initial position of the body to be simulated (key initial position of body frame relative to NED).

2.10 Wave convention

xdyn uses the *'Z'* down convention for amplitude. The azimuth or the direction of the swell that we note *'ÿ'* represents the direction of propagation of the swell.

Thus a null direction indicates a south to north propagation of the swell.
A direction of 90° represents a propagation from west to east.



The different positions of the vessel in relation to the swell are described as follows:

• 'ÿ ÿ ÿ = 0ÿ : ship with its back to the swell, swell from the rear • '0ÿ < ÿ ÿ ÿ < 90ÿ
: Swell coming from port aft • 'ÿ ÿ ÿ = 90ÿ • '90ÿ <
ÿ ÿ ÿ < 180ÿ • 'ÿ ÿ ÿ = : Swell coming from port side
180ÿ ': ship heading into the swell, : Swell coming from starboard aft
swell from the front • '180ÿ < ÿ ÿ ÿ < 270ÿ
: Swell coming from starboard bow • 'ÿ ÿ ÿ = 270ÿ
‘ : Swell coming from starboard • '270ÿ
< ÿ ÿ ÿ < 360ÿ of environment : Swell coming from forward port side # Models

Environmental models are the wave and wind (and eventually current...) models used by xdyn. Currently, only wave and wind models are implemented. Their parameterization is in the environment section of the input YAML file. It can be empty (for example, during the simple simulation of tutorial 1).

Wave models are involved in the calculation of non-linear hydrostatic forces (through the elevation of the free surface), the Froude-Krylov forces (through the dynamic pressure), the calculation of diffraction forces (via the frequency) and the rudder model (taking into account orbital velocities).

Wind models are used in particular for model sails and other wind propulsion systems. At present, only constant wind models are implemented, but in the long term turbulence models on spectral representation will be added.

2.11 Environmental constants

The acceleration of gravity (denoted by g), the volumetric density of water (rho) and its naked viscosity are constants which intervene in several physical models. Therefore, rather than being filled in at the level of each model and thus risking inconsistencies, they appear in the environmental constants section which has the following form:

```
environmental constants:
    g: {value: 9.81, unit: m/s^2} rho: {value:
    1025, unit: kg/m^3} nu: {value: 1.18e-6, unit:
    m^2/s} air rho: {value: 1.225, unit: kg/m^3}
```

In addition, the air density can be added with the air rho key, which is necessary for all aerodynamic models. This data is optional in the absence of aerodynamic models.

These four constants are the set of all environmental constants currently used by xdyn's models.

As explained in a previous section, physical dimensions are not checked and simply converted to SI units. If xdyn encounters an unknown unit, it produces an error message like:

```
unknow unit : hhm
```

2.12 Wave models

2.12.1 Simulation sans houle

To simulate a perfectly flat free surface, we operate as follows:

```
environment:
  - model: no waves
    constant sea elevation in NED frame: {value: 0, unit: m}
```

model: no waves indicates that we want a horizontal free surface and constant sea elevation in NED frame represents the elevation of the free surface in the NED frame.

In this case, the excitation forces (Froude-Krylov and radiation) will be zero.

2.12.2 Houle d'Airy

A swell can be defined as being a sum of several directional spectra , that is to say a power spectrum and a spatial dispersion. To derive the general expression of a wave composed of several spectra, we start with the case of a monochromatic and monodirectional wave.

2.12.2.1 Expression of the wave velocity potential Let *'V (x, y, z,* t) = *(u, v,* w)' be the velocity of the fluid at the coordinate point *'(x, y,* z)' (in the NED frame) and at time 't'.

Water is assumed to be inviscid, incompressible, homogeneous and isotropic and an irrotational flow is considered. Assuming irrotational flow implies (by Poincaré's lemma) that the velocity derives from a potential that we call *'ÿ : (x, y, z,* t) ÿ *ÿ(x, y, z,* t)'. By definition, the velocity at any point of the flow is therefore given by:

$$V(x,y,z,t) = \textrm{grad}{\phi(x,y,z,t)}$$

The pressure *'p'* satisfies Bernoulli's equation:

$$p + \rho g z -\rho\frac{\partial\phi}{\partial t} + \frac{\rho}{2} V\cdot V = C(t)$$

where *'C : t* ÿ *C(t)'* is an arbitrary function of time, so the equation is particularly valid for 'C(t) = p0' (atmospheric pressure at the surface):

$$p_0 + \rho g z -\rho\frac{\partial\phi}{\partial t} + \frac{\rho}{2} V\cdot V = p_0$$

either

$$g z -\frac{\partial\phi}{\partial t} + \frac{1}{2} V\cdot V = 0$$

The term *'ÿgz'* represents the hydrostatic pressure and the term ÿ ÿ ÿÿ ÿt ' is here dynamic pressure.

This is the first free surface condition.

We can define the function 'F(x, *y, z,* t) = *z* ÿ *ÿ(x, y,* t)'

For a particle on the free surface, 'F(x, *y, z,* t) = 0' which implies that its particle derivative is zero:

$$\frac{DF}{Dt} = \frac{\partial F}{\partial t} + V\cdot \nabla F = 0$$

either

ÿÿ ÿÿ ÿ ÿ + ÿt ÿx ÿx ÿÿ ÿÿ ÿy ÿx ÿy = 0 on *'z = ÿ(x, y,* t)'. *ÿz*

This is the second free surface condition.

By linearizing these two free surface conditions, we obtain:

$$\frac{\partial \eta}{\partial t} = \frac{\partial\phi}{\partial z}$$

g\eta - \frac{\partial\phi}{\partial t} = 0

Moreover, since water is assumed to be incompressible, 'ÿ *V* = = 0' ÿ2ÿ ÿ 2ÿ ÿ2ÿ ÿ + ÿx2 ÿy2 ÿz2

This is a Laplace equation whose solution is obtained by the method of separation of variables. Several potentials can be solution. For example

- 'ÿ(x, *y, z,* t) = ÿ gÿa / oh · cosh(k·(hÿz)) cos(k · *(x* · cos(ÿ) + *y* · sin(ÿ)) ÿ ÿ · *t* + ÿ)' cosh(k·h)
- 'ÿ(x, *y, z,* t) = gÿa o · cosh(k·(hÿ z)) sin(k · *(x* · cos(ÿ) + *y* · sin(ÿ)) ÿ ÿ · *t* + ÿ)' cosh(k·h)

Here we choose:

\phi(x,y,z,t) = -\frac{g\eta_a}{\omega}\frac{\cosh(k\cdot(hz))} {\cosh(k\cdoth)}\cos (k\cdot(x\cdot \cos(\gamma)+ y\cdot \sin(\gamma))-\omega\cdot t+\phi)

which is the potential used by the AQUA+ software.

- *'h'* is the depth of the fluid (height from the ground to the free surface), • *'ÿa'* is the amplitude of the swell (in m), • *'x, y,* z' are the coordinates of the point considered, expressed in the mark DOWN,
- *'k'* is the wave number, reflecting the spatial periodicity.

2.12.2.2 Relation between the number of wave and the pulsation There exists a relation between *'k'* and 'ÿ', called relation of dispersion, and which is written:

\omega^2 = g\cdot k \cdot \tanh(k\cdot h)

where *'h'* denotes the water depth and *'g'* the acceleration due to gravity.

At infinite depth ('k h > 3'), this relation tends to:

\omega^2 \sim g\cdot k

2.12.2.3 Wave height The wave height is derived from the second free surface condition:

'ÿ(x, *y,* t) = +1 ÿi ·– ÿÿ(x,y,z=0,t) ÿt / g = ÿ nf req i=1 A(ÿi)sin(ki · *(x* · cos(ÿ) + *y* · sin(ÿ)) ÿ *t* + ÿi)'

where *'ÿ'* designates the direction of origin of the swell, defined in the swell convention section.

The wave elevation *'ÿ'* at a point *'(x,* y)' is a time signal *'ÿ(t)'.* By definition of the plane *'z =* 0' (mean wave elevation), this signal *'ÿ'* is centred. It is also assumed to be stationary, implying that its autocorrelation function *'R'* depends only on *'ÿ* ':

R(\tau)=\mathbf{E} (\eta(t)\eta(t+\tau))

The power spectral density *'G'* of *'ÿ'* is equal to the Fourier transform of its auto-correlation function 'R'. As *'R'* is even and real, so *is 'G'* and we usually only consider the positive part (one-sided) by defining the spectral density *'S'* as:

$$S(\omega)=\left\{\begin{array}{cc}2G(\omega),&\omega\geq 0\\0,&\omega<0\end{array}\right.$$

On a :

$$R(\tau) = \frac{1}{2} \sum_{i=1}^{nfreq} A(\omega_i)^2 \cos(\omega_i \tau)$$

but we also have:

$$R(\tau) = \int_0^\infty S(\omega) \cos(\omega\tau)d\omega$$

whence, by identification:

$$A(\omega_i)^2=2 S(\omega_i)d\omega_i$$

We can generalize this formulation by involving the directional spreading *'D(ÿ)'* of the storm :

$$A(\omega,\gamma)^2 = 2 S(\omega)d\omega D(\gamma) d\gamma$$

We finally get:

$$\eta(x,y,t) = -\sum_{i=1}^{nfreq} \sqrt{2 S(\omega_i)d\omega D(\gamma) d\gamma} \sin(k_i\cd y\cdot \sin(\gamma))-\omega_i\cdot t+\phi_{i})$$

2.12.2.4 Dynamic pressure The expression of the dynamic pressure (pressure fields of the incident swell), used by the Froude Krylov model, is defined as the total pressure minus the hydrostatic pressure *'ÿgz'* and its expression is deduced from the first linearized free surface condition:

$$p_{\textrm{dyn}} = -\rho \frac{\partial \Phi(x,y,z,t)}{\partial t}$$

either

$$p_{\textrm{dyn}} = \rho\cdot g \sum_{i=1}^{nfreq}A(\omega_i,\gamma) \frac{\cosh(k_i\cdot(h-z))}{\cosh(k_i\cdot h)}\sin(k_i\cdot(x\cdot \cos(\gamma)+ y\cdot \sin(\gamma))-\omega_i\cdot t+\phi_{i})$$

- *'g'* indicates the acceleration of gravity (9.81 *'m/s2 '*) • *'ÿ'* is the voluminal density of the fluid (in *'kg/m3 '*)

When the depth *'h'* is very large in front of 'z', the hyperbolic cosines are equivalent to exponentials:

$$\mathop {\lim }\limits_{x \to \infty } \cosh x = \frac{{e^x }}{2}$$

we therefore obtain:

$$p_{\textrm{dyn}}\mathop \approx \limits_{x \to \infty } \rho\cdot g \sum_{i=1}^{nfreq} A(\omega_i,\gamma)$$

$$e^{-k_i\cdot z}\sin(k_i\cdot(x\cdot\cos(\gamma)+ y\cdot \sin(\gamma))-\omega_i\cdot t+\phi_{i})$$

2.12.2.5 Total pressure It can be shown that the total pressure (sum of the hydrostatic pressure and the dynamic pressure) is positive.

The reasoning is simply expressed for a single frequency and a single direction, but it is easily generalizable.

For a single frequency and a single direction, we have:

$$p_{\textrm{tot}} = \rho g z - \rho g \frac{\cosh(k(h-z))}{\cosh(k h)}\eta$$

Under the free surface, i.e. for *'z* > 0', we have *'h* ÿ *z* < h' so

$$0<\frac{\cosh(k(h-z))}{\cosh(k h)}\leq 1$$

SO

$$\rho g z - \rho g \frac{\cosh(k(h-z))}{\cosh(k h)}\eta \geq \rho g \left(z - \eta\right)$$

however for the calculation, we always consider the submerged part of the hull so *'z* ÿ ÿ', hence

$$p_{\textrm{tot}} \geq 0$$

Above the free surface, we have *'ÿ* ÿ *z* ÿ 0' so *'h* ÿ *z* ÿ h' and

$$\frac{\cosh(k(hz))}{\cosh(kh)}\geq 1$$

Or

$$p_{\textrm{tot}} = \rho g \cdot \left(z - \frac{\cosh(k(h-z))}{\cosh(k h)} \eta\right)\geq$$

As

$$1- \frac{\cosh(k(h-z))}{\cosh(k h)} \leq 0$$

and *'ÿ* ÿ *z* ÿ 0' (we are under water),

$$\eta \left(1- \frac{\cosh(k(h-z))}{\cosh(k h)}\right)\geq 0$$

and so

$$p_{\textrm{tot}}\geq 0$$

2.12.2.6 Irregular wave Wave velocity potential has so far been expressed for a single frequency and a single direction. It can be generalized for several frequencies and several directions.

in notation

$$a_{i,j} = A(\omega_i, \gamma_j) = \sqrt{2 S(\omega_i)d\omega D(\gamma_j)d\gamma}$$

the irregular wave potential is written:

$$\phi(x,y,z,t) = -\sum_{i=1}^{nfreq}\sum_{j=1}^{ndir} a_{i,j} \cdot \frac{g}{\omega_i}\frac{ {\cosh(k_i\cdot h)} \cos(k_i\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})$$

We deduce the expression of the elevation *'ÿ' :*

$$\eta(x,y,t) = \frac{1}{g}\frac{\partial\phi}{\partial t} = - \sum_{i=1}^{nfreq} \sum_{j=1 }^{ndir} a_{i,j}\sin(k_i\cdot(x\cdot \cos(\gamma_j) + y\cdot \sin(\gamma_j))- \omega_i\cdot t+\phi_{i,j })$$

as well as the expression of the dynamic pressure 'pdyn' :

$$p_{\textrm{dyn}}(x,y,z,t) = \rho\cdot g \sum_{i=1} ^{nfreq}\sum_{j=1}^{ndir} a_{i,j} \frac{\cosh(k_i\cdot(h-z))}{\cosh(k_i\cdot h)}\sin(k_i\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})$$

2.12.2.7 Orbital velocity

2.12.2.7.1 At finite depth The orbital velocity *'V (x, y, z,* t) = *(u, v,* w)' of the swell is defined by:

$$u = \frac{\partial \phi}{\partial x} = g \sum_{i=1} ^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} \frac{\cosh(k_i\cdot(h-z))} {\cosh(k_i\cdot h)}\cdot\cos(\gamma_j) \sin(k_i\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})$$

$$v = \frac{\partial \phi}{\partial y} = g \sum_{i=1} ^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} \frac{\cosh(k_i\cdot(h-z))} {\cosh(k_i\cdot h)}\cdot\sin(\gamma_j) \sin(k_i\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})$$

$$w = \frac{\partial \phi}{\partial z} = g \sum_{i=1} ^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} \frac{\sinh(k_i\cdot(h-z))} {\cosh(k_i\cdot h)} \cos(k_i\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})$$

2.12.2.7.2 In infinite depth When *'ki · h* > 3', the hyperbolic cosines can be considered as equivalent to exponentials (relative error less than '2.5×10ÿ3 '). We can therefore use the following approximation:

$$u = g \sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} e^{-k_i z}$$

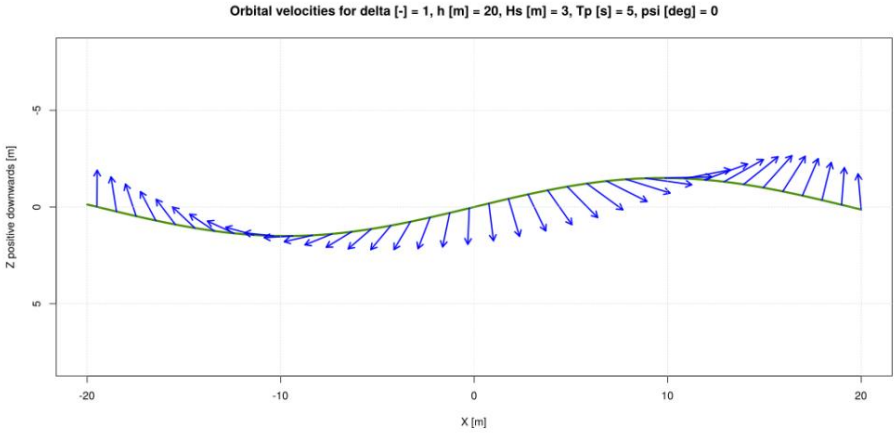
\cdot\cos(\gamma_j)
\sin(k\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})

v = g
\sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} e^{-k_i z}
\cdot\sin(\
gamma_j)
\sin(k\cdot(x\cdot
\cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})

w = g
\sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} e^{-k_i z}

\cos(k\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})

2.12.2.7.3 Evolution of the orbital velocity on the free surface The expression for the elevation of the free surface contains a sine term. The *'u'* and *'w'* orbital velocities contain cosine terms and are therefore out of phase with elevation. The *'v'* component of orbital velocity, on the other hand, is in phase with elevation. The following diagram represents the wave elevation (in green) with the orbital velocity vectors (in blue) at several points of the free surface in the (X,Z) plane:



2.12.2.8 Parametrization of wave models Airy's directional wave spectra are parameterized as follows:

- model: airy
  depth: {value: 100, unit: **m}** seed of
  the random data generator: 0 stretching: delta:
  0

```
    h: {unit: m, value: 100}
directional spreading: type:
    dirac waves
    propagating to: {value: 90, unit: deg}
spectral density: type:
    jonswap Hs:
    {value: 5, unit: m}
    Tp: {value: 15, unit: s} gamma:
    1.2
```

• model: currently can only be airy. • stretching: see the paragraph below. • depth : depth (distance between bottom and surface). 0 for the “infinite depth” approximation. Used for calculation of wave number and therefore for calculation of free surface elevation, dynamic pressures and orbital velocities. • seed of the random data generator : seed used for the generation of random phases. If we give none as value all the random phases will be null (used mainly for the tests). • directional spreading: directional spreading. See below. • spectral density: power spectral density. See below.

2.12.3 Power spectral densities

The formulation of swell spectra has been developed in a semi-empirical way since the 1950s. Depending on the spectrum, the sea state can be completely formed (at the limit, the spectrum has only one parameter) or a combination of the swell and the six-parameter wind sea.

The choice of spectrum therefore depends on both the place considered and the sea state. This choice is of great importance for forecasting the movements of the platforms; the ship's response will vary according to the type of sea state. It should be noted that the sea state cannot be varied during a same simulation.

2.12.3.1 Dirac The simplest power spectral density is also the least realistic because it corresponds to a monochromatic swell, i.e. to a single sinusoidal function:

$$\omega_0\in\mathbb{R}^+,\forall\omega\in\mathbb{R}^+, S(\omega) = \left\{\begin{array}{l}0, \textrm{si }\omega\ness \omega_0\\1, \textrm{si } \omega=\omega_0\end{array}\right.$$

'ÿ0 = 2ÿf' is the pulsation (in Rad/s) of the swell.

The corresponding time signal looks like this:

The configuration of this spectrum is:

```
spectral density: type:
    dirac
    Hs: {value: 5, unit: m} omega0:
    {value: 15, unit: rad/s}
```

The wave height is given by Hs and its pulsation by omega0. The wave amplitude will be equal to Hs/2.

This spectrum is essentially of interest for establishing transfer functions or comparing responses on a controlled swell, but it is not representative of real conditions.

2.12.3.2 Bretschneider The analytical function most often used to represent partially or fully developed sea states was proposed in 1959 by Bretschneider. Initially, the dependence on the wave period was highlighted and expressed in the form:

$$S(T)=\alpha\cdot T^3\cdot e^{-\beta T^4}$$

Today, we prefer a frequency formulation:

$$S\left(\omega=\frac{2\pi}{T}\right) = \frac{A}{\omega^5}\cdot e^{-\frac{B}{\omega^4}}$$

The moment of order 0 makes it possible to obtain a relation between *'A'* and *'B' :*

$$m_0 = \int_0^\infty S(\omega) d\omega = \left[-e^{-\frac{B}{\omega^4}}\right]^\infty_0=\fra$$

from where

$$A=4m_0B$$

In addition, the first derivative of the spectrum is canceled for a period *'ÿp' :*

$$\frac{d}{d\omega}S(\omega) = \frac{A}{\omega^6}e^{-\frac{B}{\omega^4}}\left(\frac{4B}{\ome$$

from where

$$B=\frac{5}{4}\omega_p^4$$

And

$$\omega_p=\left(\frac{4}{5}B\right)^{1/4}$$

In addition, we observe empirically that the wave heights are often distributed according to a Rayleigh law (law of the norm of a vector whose

two components follow a normal law) of variance *'ÿ* hypothesis, ^{2} = 4m0' and, under this the wave height *'HS'* corresponding to two standard deviations is:

$$H_S = 2\sigma = 4\sqrt{m_0}$$

Also, *'ÿp* = $\frac{2\,p}{City}$

We can thus express *'A'* and *'B'* in terms of *'HS'* and *'Tp' :*

$$A=\frac{5\pi^4H_S^2}{T_p^ 4}$$

$$B=\frac{20\pi^4}{T_p^4}$$

This spectrum looks like this:

Its parameterization in xdyn is carried out using the following YAML:

```
spectral density: type:
    bretschneider
    Hs: {value: 5, unit: m}
    Tp: {value: 15, unit: s}
```

2.12.3.3 Pierson-Moskowitz In the late 1940s, several weather ships were stationed in the Pacific Ocean and the North Atlantic. These ships recorded the weather daily in the form of a weather code. In 1961, Tucker created a method to obtain quantitative estimates from these records by correlating the frequency of occurrence of different weather codes with weather observations from several shore stations. In 1964 Willard Pierson and Lionel Moskowitz at New York University prepared a report for the US Naval Oceanographic Office analyzing a large number of North Atlantic records. Only the recordings for fully developed seas having been considered, it is a spectrum adapted to such sea states. It is written in the form:

$$S(\omega) = \frac{\alpha\cdot g^2}{\omega^5} \exp{\left(-\beta\left[\frac{g}{U_{19.5}\omega$$

where *'ÿ = 8.1* 10ÿ3 ' denotes the Phillips constant, *'g'* the acceleration of earth's gravity , 'U19.5' the wind speed at 19.5 meters above sea level and *'ÿ '* is 0.74.

This spectrum is a special case of the Bretschneider spectrum by taking

$$A=8.1\cdot 10^{-3}\cdot g^2$$

And

$$B=0.74\left(\frac{g}{U_{19.5}}\right)^4$$

Subsequently, the Pierson and Moskowitz data were re-analyzed to establish the following empirical relationship between wind speed and modal wave pulsation:

$$0.74 \left(\frac{g}{U_{19.5}}\right)^4=\frac{5}{4}\omega_0^4$$

from where

$$\omega_p=0.877\frac{g}{U_{19.5}}$$

An expression of the peak period *'ÿp'* only as a function of *'HS'* can be obtained by using the following relations established for the spectrum of board cutter :

- *'m0 =* $\frac{A}{4B}$.
- *'ÿp =* $\sqrt[4]{5B}^{1/4}$.

Assuming that the wave heights follow a Rayleigh distribution, we a:

$$H_S=4\sqrt{m_0}=4\sqrt{\frac{A}{4B}}$$

from where

$$B=\frac{4 A}{H_S^2}$$

$$\omega_p=\left(\frac{4}{5}B\right)^{1/4}=\left(\frac{4A}{5H_S^2}\right)^{1/4}=\left(\frac{1$$

either

$$\omega_p=0.4\sqrt{\frac{g}{H_S}}$$

This spectrum was the reference spectrum for many years but is only valid for fully developed sea states and seas resulting from moderate winds on very large fetches. For the more frequent conditions of strong winds on short fetches, especially in the North Sea, spectra with at least two parameters are more suitable.

This spectrum looks like this:

Its parameterization in xdyn is carried out using the following YAML:

```
spectral density: type:
      pierson-moskowitz
```

```
Hs: {value: 5, unit: m}
Tp: {value: 15, unit: s}
```

2.12.3.4 JONSWAP The JONSWAP spectrum (Joint North Sea Wave Project) was proposed in 1973 by Hasselmann et al. after having stripped measurements made during the formation of storms in the North Sea. More than 2000 spectra were thus measured and a least squares method was used to obtain a spectral formulation. It is valid for limited fetches and uniform wind speeds. The importance of this spectrum comes from the fact that it takes into account the development of the waves on a limited fetch on the one hand, and, on the other hand, the attenuation of the waves in shallow waters. This spectrum is often used by the offshore industry in the North Sea.

$$S(\omega)=(1-0.287 \log(\gamma))\frac{5}{16}\frac{\alpha}{\omega}H_S^2 e^{-1.25\left(\frac{$$

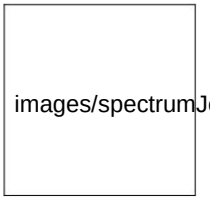
with

$$r=e^{-0.5\left(\frac{\omega-\omega_0}{\sigma\omega_0}\right)^2}$$

And

$$\sigma=\left\{\begin{array}{l}0.07,\omega\leq\omega_0\\0.09,\omega>\omega_0\end{array}\righ$$

This spectrum looks like this:



images/spectrumJonswap.png

Its parameterization in xdyn is carried out using the following YAML:

```
spectral density: type:
    jonswap Hs:
    {value: 5, unit: m}
    Tp: {value: 15, unit: s} gamma:
    1.2
```

2.12.4 Directional spreads

2.12.4.1 Dirac When this spread is chosen, the swell is monodirectional .

```
directional spreading:
    type: dirac
    waves propagating to: {value: 90, unit: deg}
```

The direction of propagation is given by waves propagating to, in the NED frame (0° corresponds to waves propagating from South to North,

45° to waves propagating from South-West to North-East, -90° to waves propagating from East to West). There are no particular bounds for this angle (apart from the maximum float size).

2.12.4.2 Cos2s The spread is given by:

\gamma\mapsto \cos^{2s}\left({\gamma-\gamma_0}\right)

where 'ÿ0' is the direction of propagation, in the NED frame (0° corresponds to waves propagating from South to North, 45° to waves propagating from South-West to North-East, -90° to waves propagating from east to west). There are no particular bounds for this angle (apart from the maximum float size).

This spread is set as follows:

```
directional spreading: type:
      cos2s s: 2

      waves propagating to: {value: 90, unit: deg}
```

The direction of propagation 'ÿ0' is given by waves propagating to.

2.12.5 Wave stretching

2.12.5.1 Description The Airy formulation is not valid above the *'z* =0' plane . This means that one cannot use it in a non-linear formulation where one seeks to know the dynamic pressures and the orbital velocities for *'z* < 0'. Stretching models are a workaround which consists in taking as reference not the *'z* =0' plane but the deformed free surface.

2.12.5.2 Use in xdyn As a reminder, the parameterization of the wave model is carried out by a YAML of the type:

```
- model: airy
  depth: {value: 100, unit: m} seed of
  the random data generator: 0 stretching: delta:
  0 h: {unit: m,
      value:
      100} directional spreading:
  type: dirac waves propagating
      to: {value: 90,
      unit: deg} spectral density: type: jonswap Hs: {value: 5,
  unit: m}


      Tp: {value: 15, unit: s} gamma:
      1.2
```

In xdyn, stretching is populated in the stretching section of swell models. The only stretching model implemented is delta-stretching and its derivatives (absence of stretching, linear extrapolation and Wheeler model).
The stretching section contains the h and delta parameters of the delta stretching model:

- absence of stretching: possible if one is on a linear modeling.
  Enter h: {value: 0, unit: m} and delta: 1. Not recommended when using models using orbital velocities (for reasons mentioned below),

- linear extrapolation, by setting *'h'* to the water depth depth and 'ÿ = 1', • Wheeler model, if *'h'* is the depth depth and 'ÿ = 0', • delta stretching for any other value .

2.12.5.3 Theoretical justifications Under the hypotheses of the linear irregular swell model detailed above, the orbital speed of the water particles with respect to the NED reference frame (projected on the *'X'* axis of the BODY reference frame) is written:

u = g
\sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j}

\frac{\cosh(k\cdot(h-z))}{\cosh(k\cdot h)}\cdot\cos(\gamma_j) \sin(k\cdot(x\cdot
\cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})

which, in infinite depth, is written:

u = g
\sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} e^{-k_i z}
\cdot\cos(\
gamma_j)
\sin(k\cdot(x\cdot
\cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})

This expression is based on the linear theory, which supposes *a priori* the flat free surface, including for the calculation of the free surface deformation, which is a quantity like the others (pressures, velocities, potential), result of the resolution of the issue.

The mathematical formulation of the problem which leads to the above expressions is not valid for z>0 (linearization of the free surface condition). A difficulty arises when one wants to exploit this formula in a non-linear modeling, ie by really modeling the deformation of the free surface. Indeed, the value of the term *'e ÿkiz* ' is less than 1 for the points below the mean level (surface *'z* = 0'), but it increases rapidly for the points located above this plane, and this all the more so as the wave number *'k'* is large, while it decreases below mean sea level. Thus, for two close points on the free (non-horizontal) surface one at *'z* > 0' and the other at *'z* < 0', the orbital speed will be very different: the contributions of the

high frequency components of the swell will be strongly amplified for the point at *'z* > 0' and strongly attenuated for the point at *'z* < 0'. Particles above mean sea level (particularly on the wave crests) will thus be seen as oscillating at high frequencies while those in the trough of the waves will oscillate more slowly: mean sea level therefore acts as a border between the amplification and the attenuation of the high frequencies, which is not physical (but coherent with the initial linearized modeling).

The linear quantities are thus not defined in the deformed zones. To overcome this drawback, so-called “stretching” models can be used, which make it possible to readjust the orbital velocities at the water-air interface (the top or the trough of the waves) in one of the ways described below. Some of these methods are equivalent to stretching the *'z'* axis (hence the name stretching). What follows is a non-exhaustive presentation of some stretching models (linear extrapolation, Wheeler model and delta-stretching).

2.12.5.3.1 Linear stretching without extrapolation Besides the absence of stretching, the simplest model comes down to locking the orbital velocity above sea level *'z* = 0':

$$\forall z\leq 0, u(x,y,z,t) = u(x,y,0,t)$$

A break in the not very physical speed profile is thus obtained. This model is not implemented in xdyn.

2.12.5.3.2 Stretching by linear extrapolation This model amounts to extending the velocity model by a tangent:

$$u(x,y,z,t) \sim u(x,y,0,t) - z\cdot \frac{\partial u}{\partial z} (x,y,0,t)$$

This model can be used in xdyn by setting h to the water depth depth and delta: 1.

2.12.5.3.3 Stretching of Wheeler The orbital speed is written:

$$u = g \sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} f(z)$$

$$\cdot\cos(\gamma_j) \sin(k\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})$$

with

$$f(z)=\frac{\cosh(k\cdot(h-z))}{\cosh(k\cdot h)}$$

After stretching, we want to find the orbital speed values given by *'f'* at the surface (in *'z* = ÿ'xi, *'ÿ'* designating the water height given by the wave model) and at the bottom (in *'z* = h'):

We are therefore looking for a function *'g'* such that:

g(z=\eta)=f(0)

g(z=h)=f(h)

We can construct such a function by taking

g(z) = f(z'(z))

with *'z ' (ÿ)* = 0' and *'z ' (h)* = h'.

We can for example choose a function *'z ÿ z* ÿ ' linear:

z'(z)=\frac{h}{h-\eta}(z-\eta)

which gives the speed profile (projected here on the *'X'* axis of the body marker):

u = g
\sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} \frac{\cosh\left(k\cdot h\frac{h-z}
{h-\eta(x,y,t)}\right)}{\cosh(k\cdot h)} \cdot\cos(\gamma_j) \sin(k\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-
\omega_i\cdot t+\phi_{i,j})

The orbital speed on the other axes is given by similar formulas.

In this model, mass is not conserved because the Laplacian of the velocity potential is not zero. There is therefore no theoretical justification for this stretching model. Its use derives more from its practical interest: it is observed experimentally that the orbital velocities calculated without stretching are further from the experimental results than those calculated with stretching.

In the case of the Wheeler model, test campaigns show that the orbital velocities calculated in the ridges are somewhat underestimated compared to the measurements.

Since this model is a special form of the delta-stretching model, it can be used in xdyn by setting *'h'* to the water depth depth and delta: 0.

2.12.5.3.4 Chakrabarti stretching In this model, we only act on the water depth in the denominator of the function *'f'*

f(z)=\frac{\cosh(k\cdot(h-z))}{\cosh(k\cdot h)}

We replace ' cosh(k · h)' by ' cosh(k · *(h +* ÿ*(x, y,* t))'. On the *'X'* axis of the body reference , for example, we thus obtain the profile:

u = g
\sum_{i=1}^{nfreq}\sum_{j=1}^{ndir}\frac{k_i}{\omega_i} a_{i,j} \frac{\cosh(k\cdot(h-z))}
{\cosh(k\cdot (h+\eta(x,y,t)))} \cdot\cos(\gamma_j)

\sin(k\cdot(x\cdot \cos(\gamma_j)+ y\cdot \sin(\gamma_j))-\omega_i\cdot t+\phi_{i,j})

The speed on the other axes is given by similar formulas.

It is found experimentally that, like the Wheeler model, the Chakrabarti model underestimates the orbital velocities in the ridges.

This model not being a derivative of the delta-stretching model, it is not accessible in xdyn.

2.12.5.3.5 Delta-stretching This is a generalization of the Wheeler model which makes it possible to pass continuously from the latter to the linear extrapolation model. By playing on its parameters, we can find three stretching models (no stretching, linear extrapolation and Wheeler model) and that is why it was chosen as a reference model in xdyn.

Just like the Wheeler model, we want to find the orbital speed at the surface at the trough and at the crest of the waves, that is to say at *'z* = ÿ'. The authors of this model, Rodenbusch and Forristal, add two parameters to Wheeler's model:

- A parameter 'hÿ' which controls the height of water on which the le stretching
- A parameter 'ÿ' between 0 and 1 (0 for the Wheeler model, 1 for the linear extrapolation)

'with varies from 'hÿ' to *'ÿÿ'* when *'z'* varies from 'hÿ' to 'ÿ'.

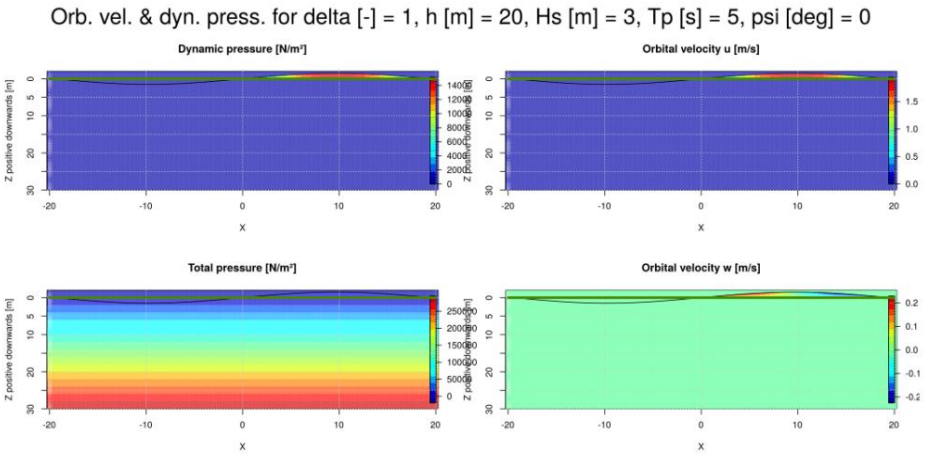
We therefore take:

- For *'z* > hÿ', *'z* = z'
- For *'z* < hÿ', *'z* *(z* ÿ hÿ) *ÿÿhÿ* ÿÿhÿ = + hÿ‘
- For *'hÿ* = 0' and 'ÿ = 1', there is no stretching.
- If 'hÿ' is the depth depth and 'ÿ = 0', we find the model of Wheeler.
- With 'hÿ' equaling depth and 'ÿ = 1' we obtain the linear extrapolation.

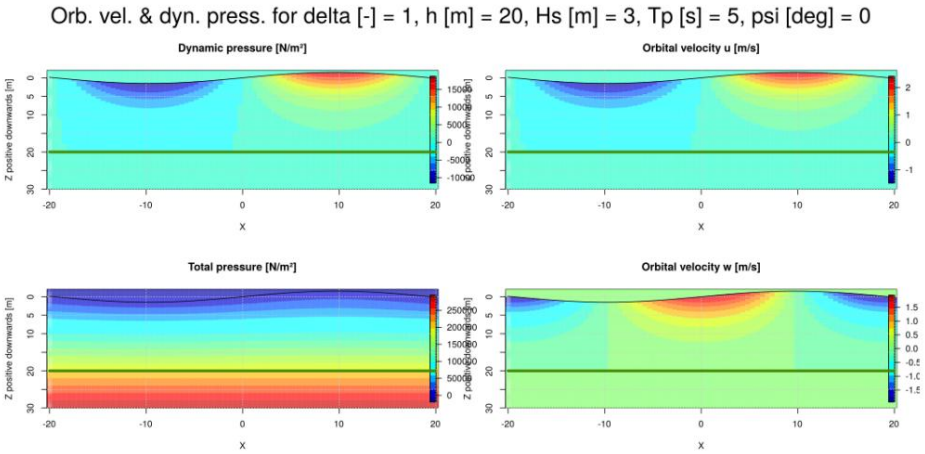
2.12.5.3.6 Choice of the stretching model As the stretching models do not really have any theoretical justification, the only way to choose them is to compare them directly with the measured velocity profiles. The test campaigns carried out until now have not made it possible to categorically choose one stretching model over another. This comes both from the difficulty of carrying out the experiment under controlled conditions and of obtaining a reliable measurement, but also from the importance of non-linear phenomena, absent from stretching models.

The three graphs below show the influence of the stretching model on the dynamic pressure (and also show that it is not taken into account in the calculation of the orbital speed).

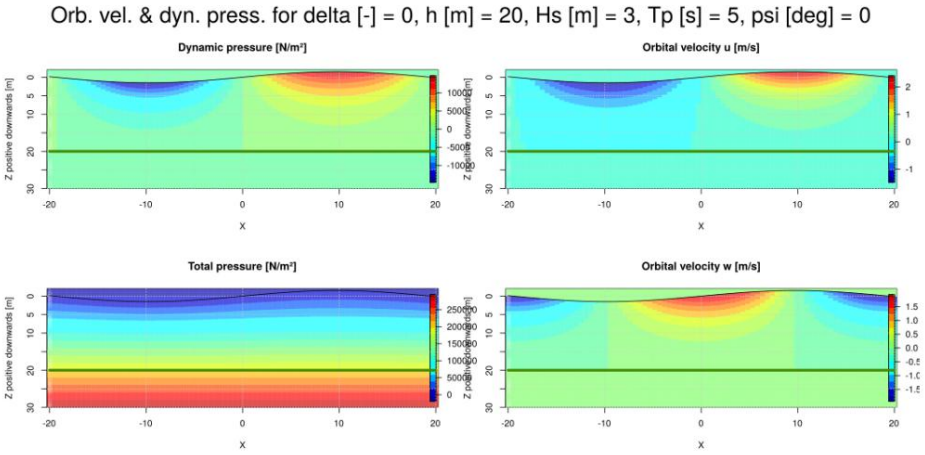
Sans stretching



Linear extrapolation



Stretching de Wheeler



2.12.6 Discretization of spectra and spreads

The spreads and spectra presented above are continuous. In order to carry out the IT implementation, they must be discretized. If the pulsations are distributed uniformly over an interval, a temporal periodicity of the swell is introduced (this amounts in fact to performing an inverse Fourier transform , which by construction gives a periodic result). In order to be more representative of real sea states, one may wish to break this periodicity by discretizing the pulsations randomly. An aperiodic signal is thus obtained .

The performance of the implementation of the wave models is crucial: indeed , the dynamic pressure and the static pressure being integrated on all the facets of the mesh (in the case of a non-linear model), these models are evaluated by many times per calculation step. As the number of components summed to calculate the dynamic elevations and pressures is potentially large, only the products 'S(ÿi)D(ÿj)' contributing significantly to the total energy are selected .

To do this, these products are classified in descending order and the first *'n'* are selected so that their sum represents a predetermined fraction of the total power. In this way, the computation times are considerably reduced , while keeping a good representativeness of the physics of the problem.
However, this technique is not always applicable, depending on the answer one is interested in. Indeed, a small body in the swell can have a response very affected by low-energy components (in relative terms), for example for wet or impact problems. Similarly, the local responses of a large body (slamming, efforts on appendages, etc.) can be affected by low-energy components.

The discretization is parameterized as follows:

```
discretization:
    nfreq: 128 ndir:
    8
    omega min: {value: 0.1, unit: rad/s} omega max: {value:
    6, unit: rad/s} energy fraction: 0.999 equal energy
    bins: true
```

• nfreq: number of frequencies • ndir:
number of directions • omega min:
minimum pulsation (included) • omega max: maximum
pulsation (included) • energy fraction : the power spectrum
and directional spread products $'Si \cdot Dj$ are listed in descending order. We calculate the cumulative sum and we stop when the accumulated energy is equal to energy fraction of the total energy.

• equal energy bins : optional parameter that can take the values true or false (by default). When it is false (or absent), the discretization of the pulses is done regularly (the step between two omegas is constant). When it is true, the pulses are discretized so that the integral (by the trapezium method) of the spectrum between two successive pulses is constant.

Warning : for reasons of compatibility with previous versions of xdyn, it is possible to use the key n instead of nfreq and ndir : in this case, the number of frequencies and the number of directions will be equal to n. However, n and nfreq or n and ndir cannot be used together . The number of lines generated is equal to the product nfreq × ndir.

2.12.7 Outputs

We can output the wave heights calculated on a mesh (defined in a fixed or mobile frame). In fact, you can even choose to do only a swell simulation, without a body, as described in tutorial 3.

We define a mesh (Cartesian) on which the swell will be calculated (in the environment/model/output section). For example :

```
output:
    frame of reference: NED
    mesh:
        xmin: {value: 1, unit: m} xmax: {value:
        5, unit: m}
        nx: 5
        ymin: {value: 1, unit: m} ymax: {value:
        2, unit: m} ny: 2
```

• frame of reference : name of the reference in which the coordinates of the points of the mesh are expressed. • xmin, xmax, nx :
definition of the discretization of the x axis. The values range from xmin (inclusive) to xmax (inclusive) and there are nx values in total. • ymin, ymax, ny: as for x.

In the previous example, the coordinates are given in the NED coordinate system. The mesh has 10 points: (1,1),(1,2),(2,1),(2,2),(3,1),(3,2),(4,1),(4 ,2),(5,1),(5,2).

Outputs are written to the file and format specified in the output section already defined at the root of the YAML file.

Two different results are obtained, depending on whether the frame in which they are expressed is mobile or fixed with respect to the frame NED. Indeed, if the frame is fixed, there is no need to repeat the x and y coordinates.

In the case of a fixed reference, we obtain an output of the form:

```
waves:
  x: [1,2,3,4,5,1,2,3,4,5] y:
  [1,1,1,1,1,2,2,2,2,2] timesteps: - t:
  0

    - z: [-4.60386,-4.60388,-4.6039,-4.60392,-4.60393,-4.6553,-4.65531,-4.65533,-4.65535,-4 - t: 1

    - from: [-3.60794,-3.60793,-3.60793,-3.60792,-3.60791,-3.68851,-3.6885,-3.6885,-3.68849,-3
```

x and y designate the coordinates (expressed in meters) in the chosen reference (here it is the NED) of the points of the mesh. t is the time at which the wave heights were calculated. z is the wave height, ie the distance between a point with coordinates (x,y,0) and the same point located on the free surface.
A positive value denotes a swell below z=0 (trough) and a negative value a value above z=0 (hump).

If the exit mark is mobile, we obtain instead a result of the form:

```
waves:
  timesteps: - t:
    0
      x: [1,2,3,4,5,1,2,3,4,5] y:
      [1,1,1,1,1,2,2,2,2,2]
    - z: [-4.60386,-4.60388,-4.6039,-4.60392,-4.60393,-4.6553,-4.65531,-4.65533,-4.65535,-4 - t: 1

      x: [2,4,5,6,7,2,4,5,6,7] y:
      [1,1,1,1,1,2,2,2,2,2]
    - from: [-3.60794,-3.60793,-3.60793,-3.60792,-3.60791,-3.68851,-3.6885,-3.6885,-3.68849,-3
```

2.12.8 Using a remote wave model

xdyn allows to use wave models on a remote server. The interest is that we can thus implement swell models that are not implemented in the xdyn code. These models can be implemented in any computer language you want (Python, Java, Go. . .) and can be used like xdyn's “internal” wave models.

2.12.8.1 Technology used The technology used to do this is called “gRPC” and allows services to be quickly defined in C++, Java, Python, Ruby, Node.js, C#, Go, PHP or Objective-C. Interface code is generated automatically for clients written in each of these languages (in the case of xdyn, C++). We thus obtain:

- A single interface definition validated by a compiler
- Generated client/server interfaces (no risk of error on this point)
- Interoperability between systems written in different languages
- A fast communication protocol.

Thus, the external swell models which respect this interface can be used, besides by xdyn, by client applications written in Python, in C++, in Java. . . For example, one could implement a wave sensor model in Java using an external wave model written in, say, Python.

2.12.8.2 Interface for wave models The interface definition file for wave models is available at the following address:

https://gitlab.sirehna.com/sirehna/waves_gRPC/-/blob/master/wave_grpc .therefore

This file is necessary if you want to implement a remote wave model (a wave server) callable by xdyn, but it is not necessary to use from xdyn an existing wave model satisfying this interface.

2.12.8.3 Parameterization in xdyn A parameterless model running on a server accessible at the address http://localhost:50001 is parameterized as suit :

```
- model: grpc url:
    http://localhost:50001
```

If the swell model contains parameters, these must appear next in the xdyn YAML file and they are transmitted directly to the server without being interpreted by xdyn. For example, a unidirectional Airy model with a JONSWAP spectrum could be parameterized as follows:

```
- model: grpc url:
    http://localhost:50001
    Hs: 5
```

Tp: 15
gamma: 1.2
seed of the random data generator: 0 waves
propagating to: 90

2.12.8.4 Example of use Tutorial 9 details the implementation of the simulation.

2.12.9 References

• *Environmental Conditions and Environmental Loads,* April 2014, DNV RP-C205, Det Norske Veritas AS, page 47
• *Hydrodynamics of Offshore Structures,* 2002, Bernard Molin, Editions TECHNIP, ISBN 2-7108-0815-3, page 70, 78 for stretching • *Sea Loads on Ships And Offshore Structures,* 1990, OM Faltinsen, Cam bridge Ocean Technology Series, ISBN 0-521-37285-2, pages 27 • *Seakeeping: Ship Behavior in Rough Weather,* 1989, ARJM Lloyd, Ellis Horwood Series in Marine Technology, ISBN 0-7458-0230-3, page 75 • *Offshore Hydromechanics,* 2001, JMJ Journée and WW Massie, Delft University of Technology, sections 6-20 and 7-11 • *Natural sea states,* 2003, Jean Bougis, Institute of Engineering Sciences of Toulon and Var, University of Toulon and Var, pages 07-4/14

• *Sea spectra revisited,* 1999, Walter H. Michel, Marine Technology Vol. 36 No. 4, Winter 1999, pp. 211-227 • *Air-Sea Interaction: Instruments and Methods,* F. Dobson, L. Hasse, R. Davis, p. 524
• *Proposed Spectral Form for Fully Developed Wind Seas Based on the Similarity Theory of S. A. Kitaigorodskii,* Pierson, Willard J., Jr. and Moskowitz, Lionel A., Journal of Geophysical Research, Vol. 69, 1964, p.5181-5190

• http://web.mit.edu/13.42/www/handouts/reading-wavespectra.pdf • *A Fourier approximation method for steady waves,* 1981, Rienecker, M.M. and Fenton, J.D., Journal of Fluid Mechanics • *A new numerical method for surface hydrodynamics,* 1987, West, B.J. and Brueckner, R.S and Janda, M. and Milder, M. and Milton R.L, Journal of Geophysics Research

2.13 Wind patterns

The only wind models currently implemented in Xdyn are 'static' models : they vary the wind in space but not in time. More specifically, they define a wind profile which changes the wind speed with altitude (but not its direction).

2.13.1 Simulation without wind

To run a simulation with no wind, just specify the no wind model in the environmental models section, as follows:

```
- model: no wind
```

Note that this model is the default wind model. It will therefore be present in all cases if no other wind model is defined.

2.13.2 Uniform wind

The simplest wind model is the uniform wind model, which results in an identical speed at any point in space and constant over time. Such a model is parameterized as follows:

```
- model: uniform wind
  velocity: {unit: m/s, value: 8} direction: {unit:
  deg, value: 135}
```

The direction of the wind follows the same convention as for the swell.

2.13.3 Power law

2.13.3.1 Formulation The power law for the wind speed profile changes the wind speed according to its altitude z:

$$U(z) = U_r\cdot(\frac{z}{z_r})^\alpha$$

where *'Ur'* is the average reference speed given at the altitude *'zr',* and *'ÿ'* is an empirical coefficient which depends in particular on the roughness of the terrain. On land it is usually set to '0.143', but at sea a value of '0.11' is more appropriate.

2.13.3.2 Parameterization The power law wind speed profile is parameterized as follows:

```
- model: power law wind profile
  velocity: {unit: m/s, value: 8} direction: {unit:
  deg, value: 135} reference height: {unit: m,
  value: 10} alpha: 0.11
```

2.13.4 Logarithmic profile

2.13.4.1 Formulation The logarithmic law for the wind speed profile changes the wind speed according to its altitude z:

$$U(z) = \frac{u_\star}{\kappa} ln(\frac{z-d}{z_0})$$

where *'uÿ'* is the friction velocity, *'ÿ'* is the Von Karman constant (' 0.41'), *'d'* is the zero speed plane (altitude at which the wind speed vanishes) and 'z0 ' is the roughness length.

If the speed *'Ur'* is known at an altitude *'zr',* we can deduce the speed at any other altitude:

U(z) = U_r\cdot\frac{ln((z-d)/z_0)}{ln((z_r-d)/z_0)}

It is this description that was chosen for the model implemented in Xdyn, with a zero speed plane at zero altitude ('d = 0', realistic hypothesis at sea).

The roughness length 'z0' can be related to the friction velocity *'uÿ'* using models such as Charnock or Volkov (the Volkov model takes into account the swell for sea roughness). In Xdyn, the choice of the value of 'z0' is left to the user. Values between '0.001m' and '0.005m' are usual at sea, with a value of '0.002m' which can be used as a default.

2.13.4.2 Parameterization The logarithmic wind speed profile is parameterized as follows:

```
- model: power law wind profile
  velocity: {unit: m/s, value: 8} direction:
  {unit: deg, value: 135} reference height: {unit:
  m, value: 10} roughness length: {unit: m, value:
  0.002}
```

2.13.5 References

- *Wind profile power law,* Wikipedia article in English, consulted on 18/12/2020.
- *Log wind profile,* Wikipedia article in English, consulted on 12/18/2020. •
Roughness length, Wikipedia article in English, consulted on 12/18/2020. # metacenter

2.14 Definitions

(definitions valid in this chapter)

The metacentre *'M'* is a point defined as the intersection of the axes of application of the Archimedes force for small variations in inclination. The algebraic distance between the center of gravity *'G'* of a ship and its metacentre 'M', denoted *'GM',* makes it possible to characterize the stability of a ship:

- A low or negative value of *'GM'* corresponds to a “loose” ship, ie easier to heel and which will not tend to return quickly to its upright position. A slack vessel will be more likely to capsize in adverse environmental conditions.

- A high value of *'GM'* corresponds to a “stiff” ship, ie a ship that is quite difficult to list and which will be subject to a rapid roll, potentially uncomfortable for the passengers and the crew.

2.15 Usage in xdyn

The calculation of the GM is carried out by the simulator using a particular effort model called “GM”. This model is content to calculate GZ for two neighboring positions to deduce GM by numerical derivation. It uses for the calculation of GZ a hydrostatic model (non-linear hydrostatic (fast) or non-linear hydrostatic (exact)) specified in parameter. To avoid calculating the hydrostatic forces three times (once for the simulation itself and twice for the calculation of the GM), it was decided to make the GM model a particular hydrostatic model. Thus, it must be used to the exclusion of any other hydrostatic model.

The model evaluates GZ for the current state and then modifies the angle *'ÿ'* by a value *of ÿ'* specified in parameter then approaches GM by

$$GM = \frac{GZ(\theta+d\theta)-GZ(\theta)}{d\theta}$$

Here is an example of parameterization:

```
external forces:
    - model: GM
      name of hydrostatic force model: non-linear hydrostatic (fast) roll step: {value: 1, unit:
      degree}
```

2.16 Theoretical foundations

The metacentre is defined as the intersection of the axes of application of the Archimedes force for small variations in inclination, or, in other words, the point where the resultant of the pressure that the water exerts on the ship (heeled , i.e. inclined) meets the median plane thereof. For this definition to have meaning, it is necessary to specify under what conditions these axes intersect.

2.16.1 Proof of existence of the metacentre

If we consider two positions of the ship 'X0, X1' and any plane of inclination (non-horizontal), we can calculate the projection of the line of action of the resultant of the hydrostatic forces for 'X0' and ' X1' on this tilt plane.
The lines intersect at a point called the metacentric point. Without additional assumption, this point does not have any other particular property.

If we place ourselves under the conditions of applicability of Euler's theorem (infinitely close isocarene positions) and if we assume that one of the positions is an equilibrium position, the plane containing the corresponding carene centers at these positions (denoted 'B0' and 'B1' respectively) and the center of gravity *'G'* is vertical since 'B0' is vertical to 'G'. Also, there is a rotation

allowing to pass from one position to another (according to Euler's theorem). The plane of this rotation is not necessarily the plane *'(G,* B0, B1)'. If we add a hypothesis of transverse and longitudinal symmetry (double-ended shell), then these two planes are necessarily confused. In this case, the metacentric point is at the intersection of the lines of action of the hydrostatic forces (and no longer simply of their projection). We can then write:

$$GZ = GM\sin(\alpha)$$

where *'ÿ'* denotes the angle of the isocarene rotation.

When the angle of inclination *'ÿ'* tends towards 0, the metacentric point *'H'* tends towards a point called metacentre.

2.16.2 Definitions

2.16.2.1 Floating plane Plane of the free surface of the liquid at rest

2.16.2.2 Floatation Intersection of the float and the plane of flotation

2.16.2.3 Waterline Perimeter of the waterline

2.16.2.4 Center of buoyancy Center of gravity of the buoyancy

2.16.2.5 Hull Submerged part of the float, located below the waterline . Its center of gravity *'C'* is called “centre of hull”. It is the point of application of the hydrostatic forces.

2.16.2.6 Isocarene waterlines Waterlines limiting hulls of the same volume. It can be demonstrated that there is an isocarene buoyancy.

2.16.2.7 Inclination axis Intersection of two infinitely adjacent isocarene waterlines.

2.16.2.8 Envelope of isocarene waterlines Envelope of the family of planes made up of all isocarene waterlines.

2.16.2.9 Thrust surface Location of the hull centers corresponding to all isocarene waterlines. If we assume the surface of the thrusts to be convex (which is not necessarily the case, as Thearle shows in Theoretical Naval Architecture), we can approximate it locally by an osculating sphere whose radius is the radius of curvature of the surface. The center of this sphere is the metacentre.

2.16.3 Characterization

2.16.3.1 Characterization of the buoyancy surface: Euler's theorem Euler's theorem stipulates that the center of buoyancy corresponding to a position of the float is also the center of buoyancy of an infinitely close isocarene position. A corollary of this theorem is that, the flotations varying little and having a fixed point (the center of buoyancy), according to the theorem of Euler on the rotations, there exists a rotation making it possible to pass from the one to the other.

Euler's theorem also provides another characterization of the buoyancy surface: isocarene flotations admit an envelope surface which is touched by each of them at the corresponding center of buoyancy. This envelope surface is the flotation surface.

2.16.3.2 Characterization of the thrust surface: Dupin's theorem The plane tangent to the thrust surface at the center of thrust is parallel to the corresponding waterline. The hydrostatic thrust is therefore normal to the thrust surface.

This theorem provides a characterization of the thrust surface: the lines of action of the hydrostatic thrusts corresponding to isocarene positions are the normals to the same surface which is the thrust surface.

2.16.3.3 Existence of an isocarene position For any vessel position, there is an isocarene position.

Let *'X0* = (z0, ÿ0, ÿ0)' and *'X1* = (z1, ÿ1, ÿ1)' be two buoyancy parameters such that *'ÿ0* ÿ ÿ1' and *'ÿ0* ÿ ÿ1'. Let 'Vmax' be the volume of the hull when the ship is completely submerged. We note *'I'* the interval '[0, Vmax]'.

We note *'Vz'* the partial map *'Vz : z* ÿ *V (z,* ÿ1, ÿ1)'. *'V* ' being continuous by assumption, *'Vz'* is also. Moreover, we know that 'Vz(ÿÿ) = 0' and 'Vz(+ÿ) = Vmax' so according to the intermediate value theorem, *'ÿV0* ÿ *I, ÿz2 : Vz(z2)* = V0 ' which means that *'X0* = (z0, ÿ0, ÿ0)' and *'X1* = (z2, ÿ1, ÿ1)' are isocarenes.

2.16.3.4 Restoring couple and metacentre: Bouguer's theorem The restoring couple *'Mr'* is by definition:

$$M_r = F_{\textrm{HS}} \cdot GZ = F_{\textrm{HS}} \cdot GM\cdot \sin(\theta)$$

For small isocarene variations of 'ÿ', the volume not varying, 'FHS' is therefore constant

$$dMr = F_{\textrm{HS}}\cdot(\sin(\theta)dGM + GM\cdot\cos(\theta)d\theta)$$

or

$$dMr = F_{\textrm{HS}}\cdot dGZ$$

from where

$$dGZ = \sin(\theta)dGM + GM\cdot\cos(\theta)d\theta$$

As *'GM'* is constant,

$$dGZ \sim GM\, d\theta$$

SO

$$GM\sim \frac{dGZ}{d\theta}$$

We also have Bouguer's theorem:

$$GM\sim \frac{I_{Oxx}}{V}$$

2.16.4 Extension: use of the GM in dynamics

To use the concept of metacentre for a real ship on the swell, it is necessary to overcome the previous assumptions (flatness of the free surface, small isocarene rotations, symmetries). We then pose as a definition (and no longer a first-order approximation):

$$GM=\frac{dGZ}{d\theta}$$

The *'GM'* therefore gives an indication of the rate of variation of the hydrostatic restoring torque : the greater the *'GM'* for a given hull volume, the more the restoring moment will tend to vary rapidly for small changes in inclination 'ÿ'.

2.17 Bibliography

- *Theoretical Naval Architecture,* Samuel James Pope Thearle, ISBN 978-1236765673
- *A Treatise on the Stability of Ships,* Edward James Reed, page 89 • *Fluid mechanics: general equations and statics of fluids,* Rock Guével, National School of Mechanics - University of Nantes, 1969, page 113
- *Naval hydrodynamics: theory and models,* 2009, Alain Bovis, Presses de ENSTA, page 80 # Stability curves

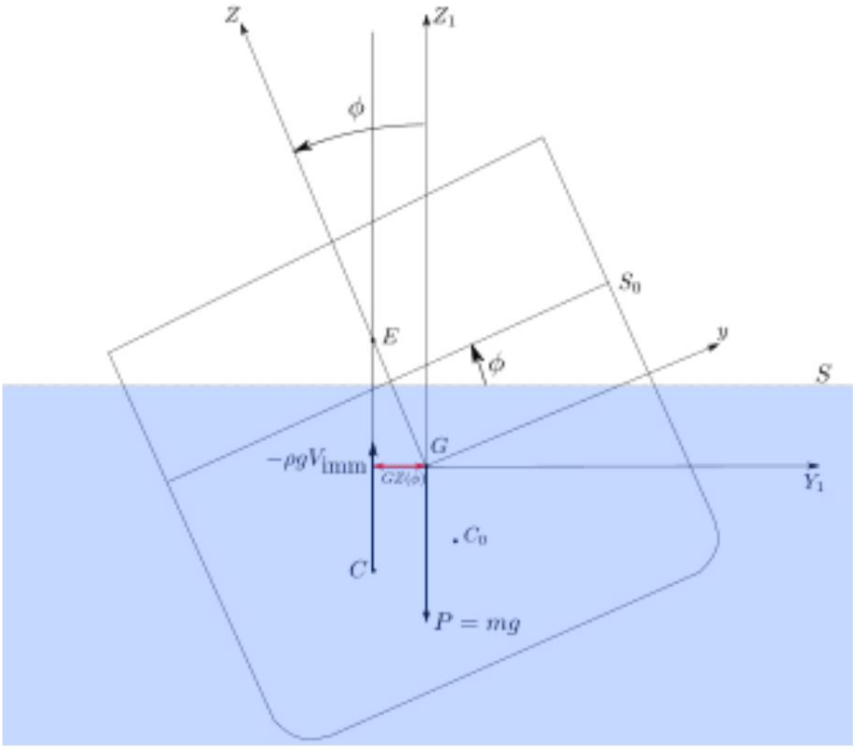
2.18 Roll stability

We are interested here in the stability of the ship in roll which constitutes a basic condition for the safety of the ship, and is the subject of strict regulations.

We consider a ship partially submerged in calm water (the free surface therefore being flat and horizontal) whose submerged volume 'ÿ' is delimited by a surface called “hull” (or wet surface) and denoted 'C' . We call “waterline” and we note *'S'* the intersection of the volume of the ship with the plane of

the free surface. We therefore have *'ÿÿ* = C ÿ S'. We note *'G'* the center of gravity of the ship and *'C'* the center of 'ÿ' (called “hull center”).

The *'Y* ' and *'Z'* axes of the “body” marker are denoted *'y'* and 'z' respectively . The situation can be represented by the following figure:



The algebraic distance *'GZ = yC* ÿ *yG'* is the lever arm of the hydrostatic restoring torque. This must be sufficient to right the ship. A necessary and sufficient condition for the hydrostatic restoring torque to be a righting torque is that the metacentric point *'E'* is located above the center of gravity 'G'. Point *'E'* is at the intersection of line *'Cz'* and line *'C0z'*.

2.18.1 Calculation algorithm of *“GZ”*

To calculate *'GZ(ÿ)',* it is necessary to know the position of the center of the hull for an equilibrium position at an angle of heel 'ÿ'. We therefore start by calculating the position of equilibrium for a given list, then we calculate the center of the hull and *'GZ'* is then given by 'GZ(ÿ) = *yC (ÿ)* ÿ yG'.

2.18.1.1 Calculation of the equilibrium position at a given heel Let *'X = (z, ÿ,* ÿ)' be the state of the system. We assume that the ship is subjected to the only forces

gravity and hydrostatics. We note *'f'* the function which associates with *'X'* the sum of the forces applied to the system:

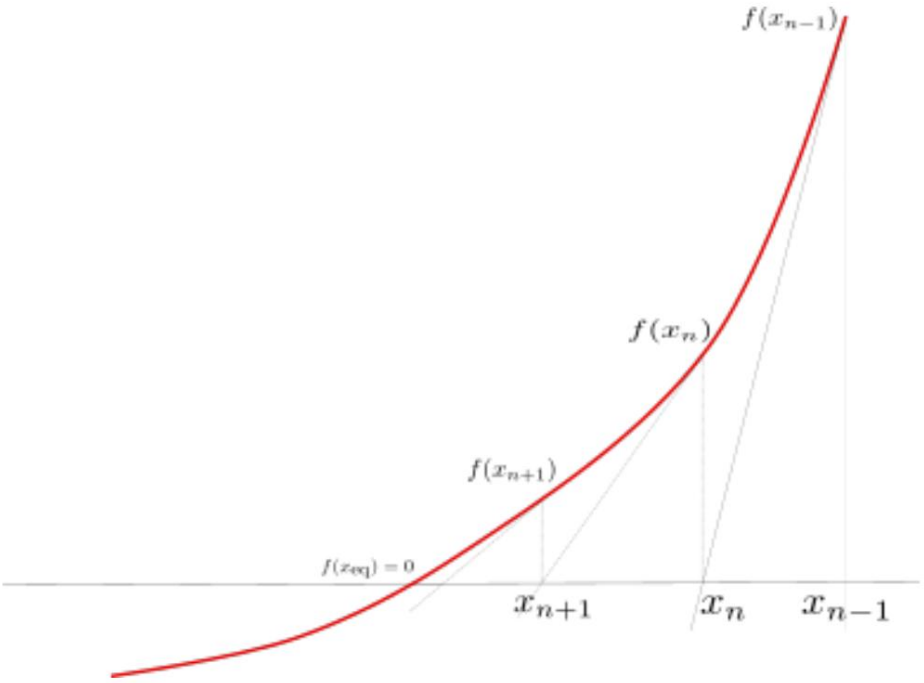
f(X) = F_{\textrm{hs}}(X) + m\cdot\mathbf{g}

where *'m'* denotes the mass of the system and **'g'** is the acceleration vector due to gravity.

When the system is at equilibrium, we have:

f(X_{\textrm{eq}})=0

To solve this equation, one can for example use Newton's method
Raphson :



X_{n+1} = X_n - f'(X_n)^{-1}f(X_n)

On note

K(X) = \frac{\partial f}{\partial X}(X)

X_{n+1} = X_n - K^{-1}(X_n)f(X_n)

The matrix *'K(Xn)'* is numerically estimated by linearizing *'f'* around *'Xn'.*
Let *'ÿX'* be a small displacement around *'Xn'* and 'ÿF = *(ÿFz, ÿMÿ, ÿMÿ)'* the corresponding force variation.

K(X_n) \Delta X = \Delta F

For '1 ÿ *i* ÿ *3,* $\sum_{j=1}^{3}$ kijÿxj = *ÿFi* '.

If the small displacement that one considers is carried out exclusively along the axis 'j', one finds:

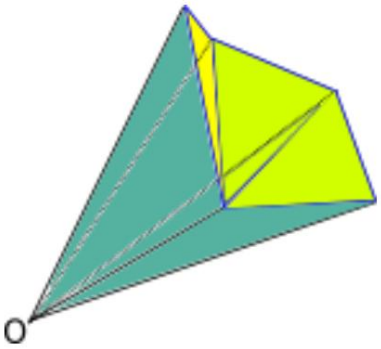
'kijÿxj = ÿFi ' so

$$k_{ij} = \frac{\Delta F_i}{\Delta x_j}$$

In practice, to evaluate the terms of the matrix *'K(Xn)',* we consider separately three small displacements around *'Xn' (one* per axis) and we use the previous formula to evaluate the terms *'kij* ' three by three.

A possible simplification is to consider that the matrix *'K'* varies little and therefore to evaluate it only once (rather than at each step of the Newton-Raphson algorithm).

2.18.1.2 Calculation of the center of the hull *'C'* The hull is discretized by polygons . To calculate its center of volume, we transform these polygons into triangles and, for each triangle, we calculate the (algebraic) volume of the tetrahedron with base this triangle and vertex at the origin.



By performing the sum of these elementary volumes, we find the volume delimited by the mesh. The hull center is calculated as follows.

Let 'P1, P2, P3' be the three vertices of one of the triangles. The elementary volume *'dV* ' associated with this triangle is the determinant of the vectors 'P1, P2, P3' :

$$dV=\textrm{it}(P_1,P_2,P_3)$$

The coordinates of the center (with respect to the origin chosen for the tetrahedrons) are given by:

$$x_C = \frac{1}{\sum_{\textrm{facet}}dV}\sum_{\textrm{facet}}\frac{x(P_1)+x(P_2)+x(P_3)}{4}d$$

$$y_C = \frac{1}{\sum_{\textrm{facet}}dV}\sum_{\textrm{facet}}\frac{y(P_1)+y(P_2)+y(P_3)}{4}d$$

$$z_C = \frac{1}{\sum_{\textrm{facet}}dV}\sum_{\textrm{facet}}\frac{z(P_1)+z(P_2)+z(P_3)}{4}d$$

2.18.1.3 Other calculation method A simpler method, since it does not require the explicit calculation of the center of the hull, is to project the vector *'GB'* onto the vector *'y'* of the vertical plane attached to the body:

GZ = y\cdot GB

or

$$'y = \frac{x_{\text{body}}^{(\text{down})} \times z_{\text{down}}^{(\text{down})}}{x_{\text{body}}^{(\text{down})} \times z_{\text{down}}^{(\text{down})}}$$

where *'x '* designates the coordinates of the vector *'x'* of the frame body, expressed body (ned) in the frame NED and *'z '* the coordinates of the vector *'z'* of the frame NED expressed in the frame NED.

It turns out that it is not necessary to know the *'z'* coordinate of the GB vector . In effect,

$$'\ddot{y}\ddot{y}0\ \ddot{y}\ \mathbf{R:}\ GB = \frac{M\times F}{\ddot{y}F\ \ddot{y}} + \ddot{y}0F'$$

where *'M'* is the moment in *'G'* of the hydrostatic force and *'F'* the resultant of the hydrostatic force.

On pose

GB_{\lambda} = \frac{M\times F}{\left\|F\right\|} + \lambda F

In the NED frame, we have:

F = \left[\begin{array}{c}0\\0\\f_z\end{array}\right]

M = \left[\begin{array}{c}m_x\\m_y\\m_z\end{array}\right]

SO

GB_{\lambda} = \left[\begin{array}{c}mf_x\\mf_y\\mf_z+\lambda F\end{array}\right]

where we noted

M\times F = \left[\begin{array}{c}mf_x\\mf_y\\mf_z\end{array}\right]

$$'GZ\ddot{y} = y \cdot GB\ddot{y} = \ddot{y}\ \begin{matrix} x \\ \text{and} \end{matrix}\ \ddot{y} \times \ddot{y}\ \begin{matrix} 0 \\ 0 \\ 1 \\ \text{with} \end{matrix}\ \ddot{y}\ \begin{matrix} mfx \\ mfy \\ mfz + \ddot{y}F \end{matrix}\ \ddot{y}$$

from where

$$'\ddot{y}\ddot{y}\ \ddot{y}\ R,\ GZ\ddot{y} = \frac{xmx}{fz} + \frac{ymy\ fz}{}$$

or, with the usual rotation conventions:

GZ = \frac{\cos(\psi)\cdot\cos(\theta)\cdot m_x + \sin(\psi)\cdot\cos(\theta)\cdot m_y}
{f_z\left\|x_{\textrm{body}}^{\textrm{(ned)}}\right\|}

2.19 Tool for calculating *'GZ'*

The installation of xdyn contains, in addition to the xdyn executable , another executable called gz, allowing to calculate static stability curves, also commonly called “GZ” curve, according to the heel 'ÿ'.

This tool takes as input command line parameters and a YAML file in the same format as that for the simulator. Unlike the simulator , the gz tool does not use the initial conditions, the outputs, or the external forces specified in the YAML file.

python echo=False, results='verbatim', name='gz-command-line-arguments' from subprocess import check_output import re r=re.compile(r"Righting.*USAGE:", re.DOTALL) print(re.sub(r,"", check_output(['gz']).decode('utf-8')))

Example :

python echo=False, results='verbatim', name='gz-example' from subprocess import check_output cmd = ['gz', 'tuto_execution/test_ship_in_waves.yml', '-s', 'tuto_execution/test_ship.stl', '--dphi', '10', '--phi_max', '40'] print(' '.join(cmd)) print(check_output(cmd).decode('utf-8'))

2.20 References

- *Hydrodynamics of Offshore Structures,* 2002, Bernard Molin, Editions TECHNIP, ISBN 2-7108-0815-3, page 398
- *Seakeeping: Ship Behavior in Rough Weather,* 1989, ARJM Lloyd, Ellis Horwood Series in Marine Technology, ISBN 0-7458-0230-3, page 191 • *Ship Dynamics,* 1986, Pierre Devauchelle, Library of the French Institute of Aid for Maritime Professional Training, ISBN 2-225-80669-1 , page 168 • *Naval hydrodynamics: theory and models,* 2009, Alain Bovis, Presses de l'ENSTA, page 79 • *Ship stability - examination of a dossier,* CETMEF, December 2012, page 11 # How the solver works

At the heart of the simulator, the solver performs the temporal integration of ordinary differential equations.

2.21 Formulation of the problem

Let *'n* ÿ Nÿ ', *'p* ÿ N', *'m* ÿ N'. A function is called a *model*

$$f:\mathbf{R}^n\times\mathbf{R}^m\times\mathbf{R}\times\mathbf{R}^p\rightarrow\mathbf{R}^n$$

- *'n'* is called “number of states” • *'p'* is the number of system parameters • *'m'* is the number of system inputs

A differentiable function *'X : t* ÿ **R** ÿ *X(t)* ÿ *Rn'* is called *the state vector* of this system: these are the variables that summarize all the information calculated by the model (for example, position, attitude, angular velocity and rotational speed).

'U ÿ *Rm'* are system *inputs* (eg commands).

'P ÿ *Rp* are the system parameters, i.e. the constants.

The differential equation that we want to integrate is:

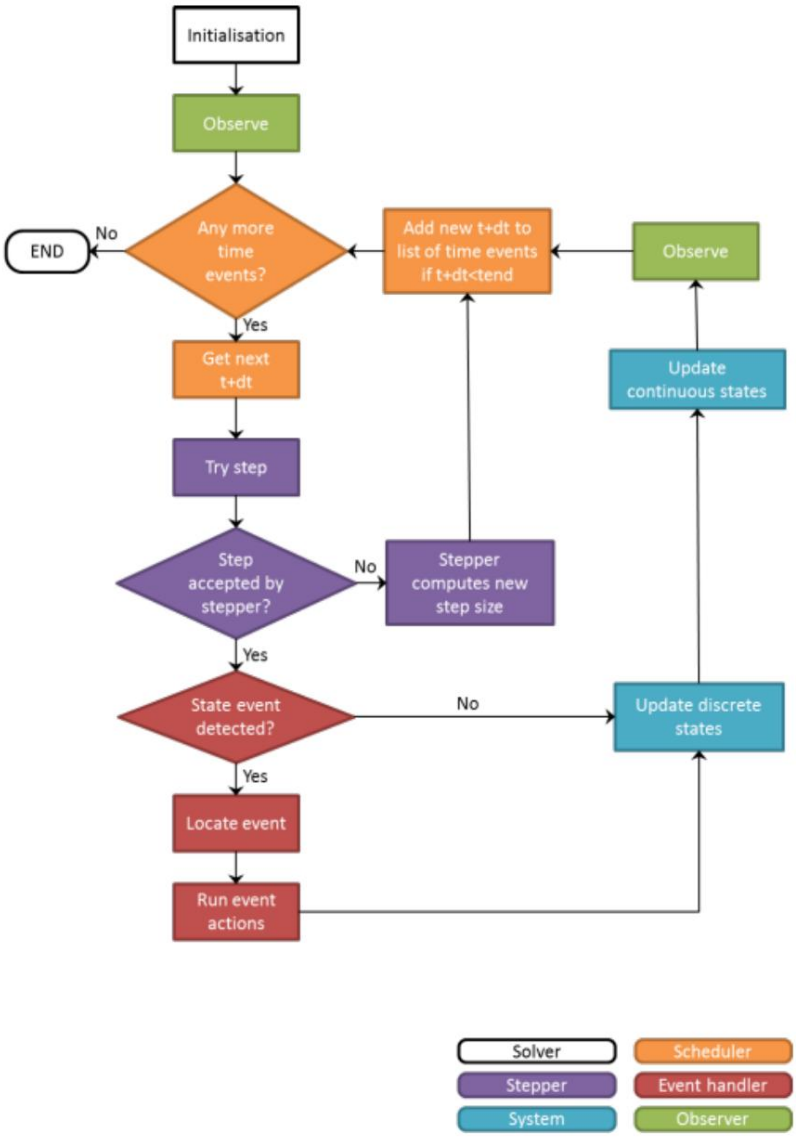
$\dot{X} = \frac{dX(t)}{dt} = f(X,U,t,P)$

2.22 General architecture

The solver consists of five elements:

- a *stepper* which calculates, for an integration step *'dt',* 'X(t+dt)' according to of 'f', 'X', 'U', *'t'* and 'P'.
- a *scheduler* which calculates the next time step (and therefore the integration step *'dt'* taking into account any events (discontinuities of the function 'f')).
- an *event handler* which takes care of locating and processing events. It is this component that determines what actions to perform when an event is detected (restart of the solver, stop, change of states, etc.).
- an *observer* whose function is to perform actions during the simulation (for example, export states to a file or to a visualization system). • a *system* which implements the calculation of the function 'f'.

The following figure illustrates the interactions between these components.



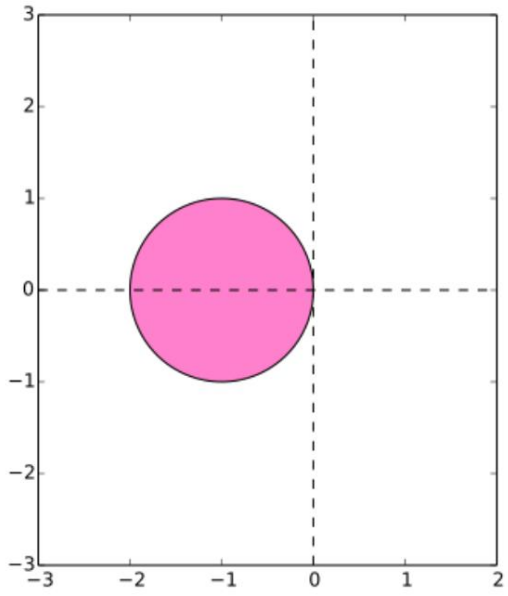
2.23 Steppers

The steppers realize the integration of *f* on a time step. Currently, three steppers are implemented:

2.23.1 Euler

$$\hat{X}(t+dt) = X(t) + f(X,t,U,P)\cdot dt$$

It is the fastest stepper, but also the least numerically stable: if this method is applied to the differential equation *'y = k* y', then the numerical solution is unstable when the product *'dt* k ' is in -outside the region ' {z ÿ **C :** *ÿz* + 1ÿ ÿ 1} '. In practice, it is only used for testing because other steppers show better performance.



2.23.2 Runge-Kutta 4

$$\hat{X}(t+dt) = X(t) + \frac{dt}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

with

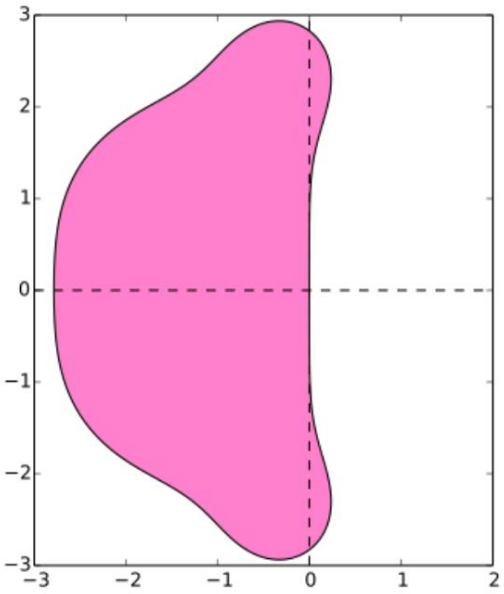
$$k_1 = f(X, t, U, P)$$

$$k_2 = f\left(X+\frac{dt}{2}\cdot k_1, t + \frac{dt}{2}, U, P\right)$$

$$k_2 = f\left(X+\frac{dt}{2}\cdot k_2, t + \frac{dt}{2}, U, P\right)$$

$$k_4 = f\left(X+dt\cdot k_3, t+dt, U, P\right)$$

It is a widely used stepper in engineering.



2.23.3 Runge-Kutta Cash-Karp

It is an adaptive step method which makes it possible to estimate the integration error. The error estimate is used to control the integration step of the scheme.

‘X^(t + *dt) = X(t)* + $\frac{37}{378}$ k1 + _ $\frac{250}{621}$ · k3 + $\frac{125}{594}$ · k4 + $\frac{512}{1771}$ · k6'

The error committed is approximated by the following relation

‘e(t+dt) = k5 $\frac{37}{378}$ $\frac{2825}{27648}$ k1 + _ $\frac{250}{621}$ $\frac{18575}{48384}$ · k3 + $\frac{125}{594}$ $\frac{13525}{55296}$ · k4 + ÿ $\frac{277}{14336}$ + $\frac{512}{1771}$ _ 1 4 · k6'

with

k_1 = dt\cdot f\left(X, t, U, P\right),

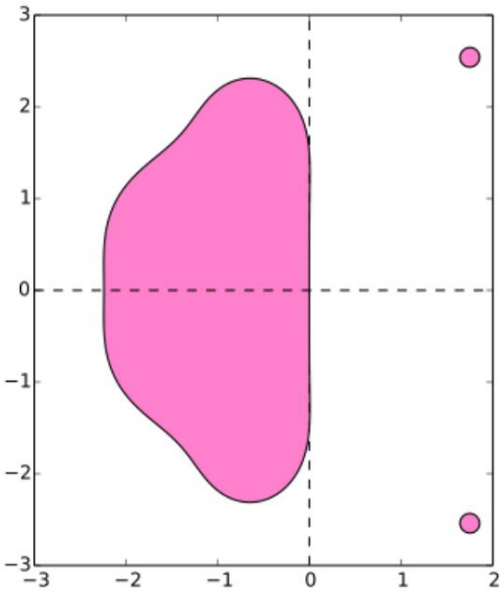
k_2 = dt\cdot f\left(X + \frac{1}{5}\cdot k_1, t+\frac{1}{5}\cdot dt, U, P\right),

k_3 = dt\cdot f\left(X + \frac{3}{40}\cdot k_1 + \frac{9}{40}\cdot k_2, t+\frac{3}{10}\cdot

k_4 = dt\cdot f\left(X + \frac{3}{10}\cdot k_1 + \frac{-9}{10}\cdot k_2 + \frac{6}{5}\cdot

k_5 = dt\cdot f\left(X + \frac{-11}{54}\cdot k_1 + \frac{5}{2}\cdot k_2 +\frac{-70}{27}\cdo

k_6 = dt\cdot f\left(X + \frac{1631}{55296}\cdot k_1 + \frac{175}{512}\cdot k_2 + \frac{575



Modeling of diffraction and radiation efforts

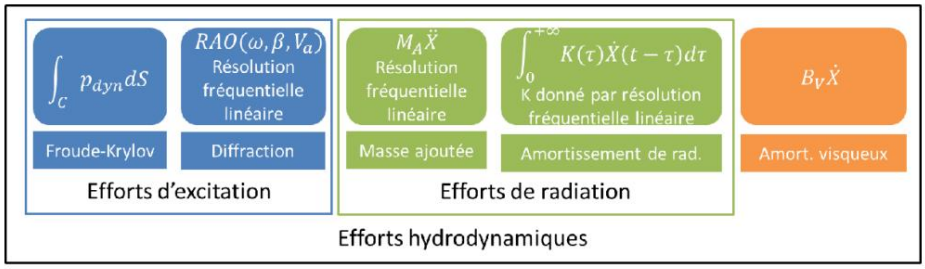
This section proposes a breakdown of the hydrodynamic forces according to a diagram classically used for the resolution of the problems of behavior at sea . The hydrodynamic forces are then supposed to consist of the sum of:

- resulting excitation forces:
  - pressures applied to the hull by the incident swell (Froude-Krylov forces). xdyn allows two modeling of these forces: ÿ a linear modeling based on the interpolations of the hydrodynamic databases HDB or PRECAL_R, ÿ a non-linear modeling, where the incident wave pressures are integrated at each moment on the exact geometry of the body in movement relative to the deformed free surface. – swell modified by the presence of the body (assumed fixed), or diffraction.
- forces related to the movements of the ship in initially calm water (without swell), and to the generation of associated waves (radiation). These efforts themselves consist of:
  - components in phase with the acceleration of the body, and comparable to inertial terms. These terms are of an order of magnitude close to the terms of mechanical inertia, and must be associated with these (in the left member of the equation of motions to be solved) in order to avoid the numerical instabilities which appear if we consider them

as external forces.

– components in phase with the velocities of the body, corresponding to damping terms. These terms can be expressed in the time domain from impulse formulations, using information from a frequency calculation.

• these terms of damping are only of potential origin, and are not sufficient to represent the physics of dampings for certain degrees of freedom, corresponding in particular to the mechanical resonances.
It is then necessary to add a damping of viscous origin, which can be calculated in different ways.



2.24 Potential for interaction between the wave and the obstacle

Water is assumed to be inviscid, incompressible, homogeneous and isotropic and an irrotational flow is considered. Assuming irrotational flow implies (from Poincaré's lemma) that the velocity derives from a potential that we call 'ÿT : *(x, y, z,* t) ÿ *ÿT (x, y, z,* t)'. By definition, the velocity at any point of the flow is therefore given by:

V(x,y,z,t) = \textrm{grad}{\Phi_T(x,y,z,t)}

The potential of the incident swell is known if we place ourselves within the framework of the linear theory of Stokes. This potential will be designated by 'ÿI '. We have in fact:

$$\text{'ÿI } (y, z, t) = \frac{gH \cosh(k(zÿh))}{2ÿ \cosh(kh)} \cos(kÿ ÿ ÿt)$$

with: $\text{'ÿ}^2 = g\ k \tanh(kh)$ '

where *'k'* denotes the wave number, *'ÿ'* the pulsation of the swell, *'h'* the water depth , *'H'* the double amplitude (or trough) and *'g'* the acceleration due to gravity.

On pose :

\Phi_P := \Phi_T-\Phi_I

which is called “potential of interaction between the swell and the obstacle”. It is this potential that interests us here since it is the only indeterminate.

We constrain 'ÿP ' to be a harmonic function of the pulsation time *'ÿ' :*

\Phi_P(x,y,z,t) = \Re{\Psi_P e^{-i\omega t}}

2.25 Conditions Satisfied by the Interaction Potential between the swell and the obstacle

The unknown potential 'ÿP ' must satisfy the following conditions:

1. The fluid being incompressible and the flow irrotational, 'ÿP ' satisfies Laplace's equation 'ÿÿP = 0'.
2. If the obstacle is assumed to be stationary, the free surface condition is written: + *ÿ* $g \cdot \frac{\ddot{y}\ddot{y}P}{\ddot{y}z}$ 2ÿP *(x, y, 0,* t) = 0' 3. The potential being indeterminate if we do not impose a condition on the direction of propagation of the energy, we will impose the Sommerfeld radiation condition (divergent phenomenon; the energy moves away from the obstacle) ÿ ÿ' (radiation : ‘ ÿ *r*|ÿÿ*P ÿr* ÿ i‖k‖ÿP | ÿ 0‘ quand *‘r = x* $\sqrt{\quad^2 + and^2}$ condition at infinity). This condition expresses that, in the distance, the diffraction-radiation waves have wave number 'k', propagate in the radial direction and their amplitude decreases like '1/ ÿ r'.
4. The bottom being impermeable, the following component *'z'* of the velocity y is zero ÿÿP : 'Vz(x, *y, h,* t) = $\frac{\ddot{y}\ddot{y}P}{\ddot{y}z}$ *(x, y, h,* t) = 0' and hence even : *ÿz* 5. The $\overline{\quad}$ *(x, y,* h) = 0‘. obstacle being impermeable, '(gradÿP) n = $\frac{\ddot{y}\ddot{y}P}{\ddot{y}n}$ = ÿ(gradÿI)·n+Vo ·*n'* where *'Vo'* is the normal speed of the obstacle at a point *'P'* and *'n'* is the normal outside the obstacle.

2.26 Resolution

If the obstacle is assumed to be fixed, condition (5) is written:

$$\frac{\partial \Psi_P}{\partial n} = - \frac{\partial \Psi_I}{\partial n}$$

This condition reflects the reflection (or diffraction) of the incident swell on the fixed obstacle. A potential 'ÿPD' satisfying the conditions (1) to (4) and the diffraction condition is called diffraction potential.

If we consider only one elementary oscillatory motion of the following obstacle ' ÿÿP its 'j'-th degree of freedom, the condition (5) is written: = *nj* '. A potential *ÿn* 'ÿP *Rj* ' satisfying conditions (1) to (4) and this condition is called 'j'-th elementary radiation potential and corresponds to the motion generated by this elementary oscillatory motion of the obstacle.

Ultimately, the complete solution *'ÿD'* of the diffraction-radiation problem obtained by superimposing the “fixed obstacle” solution and the elementary oscillatory solutions can be written:

$$\Phi_D = \Re\left[\Psi_{\textrm{PD}}e^{-i\omega t} + \sum_{j=1}^6 f_j\Psi_{\textrm{PR}_j} e^{-i\omega t}\right]$$

by adopting the following notations:

• *'V* = [f1, f2, f3] ' is the speed of translation of the body • 'ÿ = [f4, f5, f6] ' is its speed of rotation

2.27 Expression of the hydrodynamic efforts

On pose ‘ÿPR = ÿ $\sum_{j=1}^{6}$ fjÿP *Rj* *ÿiÿt ' e*

The hydrodynamic forces are written:

$$F_{\textrm{hd}} = \rho \frac{d}{dt} \int_{C}\Phi_T dS$$

$$= \rho \frac{d}{dt} \int_{C}\Phi_I+\Phi_{\textrm{PD}}+\Phi_{\textrm{PR}} dS$$

We call 'FFK = *ÿ* $\frac{d}{dt}\int_C$ *ÿIdS'* “Froude-Krylov efforts” and *'FD* = $\frac{d}{dt}\int_C$ *ÿPDdS'* “diffraction forces”. Together they constitute the wave excitation forces *ÿ 'FE'.* The forces *'ÿ* of radiation' and are denoted $\frac{d}{dt}\int_C$ *ÿPRdS'* are called “forces by *'FR'.* So we have :

$$F_{\textrm{hd}} = F_E + F_R$$

2.28 Calculation of the efforts of radiation

We have, for the *'k'* axis :

$$F_{\textrm{R}_k} = \rho \frac{\partial}{\partial t} \int_C \Phi_{\textrm{PR}}n_k dS$$

$$= \rho \frac{\partial}{\partial t} \Re \int_C \sum_j \Psi_{\textrm{PR}_j} f_je^{-i\omega t}$$

$$= \rho \Re \int_C \sum_j -i\omega \Psi_{\textrm{PR}_j} f_j e^{-i\omega t} n_k dS$$

We break down 'ÿPRj ' into its real part and its imaginary part: 'ÿPRj = *ÿR* + iÿI *jj*

We then have:

$$F_{\textrm{R}_k} = \rho \Re \int_C \sum_j \Re(-i\omega \Psi_j^R + \omega \Psi_j^I) f_j e^{-i\omega t} n_k dS = \rho \Re \int_C \sum_j -i\omega f_j e^{-i\omega t} \Psi_j^R + \sum_j \omega f_j e^{-i\omega t} \Psi_j^I n_k dS$$

We note that

$$-i\omega f_j e^{-i\omega t} \Psi_j^R = \frac{\partial}{\partial t}(f_je^{-i\omega t})$$

SO

$$F_{\textrm{R}_k} = \rho \Re \int_C \sum_j \omega f_j e^{-i\omega t} \Psi_j^I + \sum_j \fra = \rho\omega \sum_j\Re (f_j e^{-i\omega t})\int_C \Psi_j^I n_k dS + \rho \sum_j \frac{d}{dt}\Re(f_j e^{-i\omega t})\int_C\Psi_j^R n_k dS$$

On pose *‘Uj = ÿ(fj e ÿiÿt)‘*

'FRk = *ro* j *Uj* C *ÿI jnkdS + ÿ j* $\frac{dUj}{dt}$ C *ÿR nkdS‘* j

But according to the condition (5) written for the elementary potentials of radiation,

$$n_k = \frac{\partial \Psi_k^R}{\partial n}$$

So we have :

F_{\textrm{R}_k} = \rho\omega \sum_j U_j\int_C\Psi_j^I \frac{\partial \Psi_k^R}{\partial n} dS + \rho \sum_j \frac{dU_j}{dt} \int_C \Psi_j^R \frac{\partial \Psi_k^R}{\partial n} dS

On pose :

{M_A}_{jk}(\omega) = -\rho \int_C \Psi_j^R \frac{\partial \Psi_k^R}{\partial n} dS \textrm{

{B_r}_{jk}(\omega) = -\rho \omega \int_C \Psi_j^I \frac{\partial \Psi_k^R}{\partial n} dS \

One calls *'MA'* matrix of the added masses (which comes from what the solid displaces the fluid) and *'Br'* matrix of the dampings due to the radiation (terms of non-viscous damping due to the dispersion of energy by the waves generated by the solid).

We then have:

'FRk = ÿ $_j$ *UjBrjk(ÿ)* + $_j$ $\frac{dUj}{dt}$ *Mike(ÿ)* '

By taking *'U = dÿ/ dt',* we obtain a vector formulation of the forces in the frequency domain:

'FR(t) = ÿ MA(ÿ) $\frac{d\,2ÿ(t)}{dt2}$ + Br(ÿ) $\frac{dÿ(t)}{dt}$

2.29 Properties

One can show by using the second identity of Green, that the matrices *'MA'* and *'Br'* obtained previously are symmetric and that the matrix *'MA'* is positive definite: one can thus consider *'MA'* as an inertia matrix that this is called “added inertia”. One can also find the expression of *“MA”* of the evaluation of the kinetic energy of the fluid:

‘2Ec = ÿ $_{oh}$ (ÿÿ)2dV = *ÿ* ÿÿ ÿ $\frac{ÿÿ}{ÿn}$ *dS = ÿfkfl* $_{ÿÿ}$ *ÿk* $\frac{ÿÿl}{dS}$ *= ÿMAklfkfl* ' *ÿn*

2.30 Temporal writing

When one writes the balance of the forces applied to the solid, one has:

(M+M_A(\omega)) \ddot{X} + B_r(\omega) \dot{X} = F_{\textrm{others}}

Although this equation looks like a differential equation, it is not because it only describes motions in a sinusoidal steady state: this equation is only a representation of the ship's frequency response.

This finding was made in 1962 by WE Cummins, then employed by the US Army's David Taylor Model Basin *(The Impulse Response & Ship Motions,* Report 1661, October 1962).

In this paper, Cummins sets out to make hydrodynamic stresses explicit in the time domain. To do this, he makes the assumption that the movements

of the ship are a time-invariant linear system and that, therefore, the response of the ship to any excitation can be deduced from its impulse response. It considers the velocity potential 'ÿj (t)' of the flow during an impulse response along the *'j'* axis and breaks it down into two components:

- on the one hand, *'ÿj* (t)' the velocity potential (normalized by the velocity *'vj* ') of the flow during a pulsed excitation (Dirac) of amplitude *'vj* ' on the axis 'j',
- on the other hand, *'ÿj* (t)' the velocity potential (normalized by the displacement *'ÿxj* ') of the flow following the Dirac on the 'j' axis.

We therefore have, for an impulse excitation of the axis *'j' :*

$$\Theta = v_j\Psi_j + \phi_j(t)\Delta x_j$$

The velocity potential due to an arbitrary movement along the *'j'* axis is then written :

$$\Theta = \dot{x}\Psi_j + \int_{-\infty}^t \phi_j(t-\tau)\dot{x_j}(\tau)d\tau$$

The forces acting on the hull along the *'k'* axis due to an excitation of the *'j'* axis can be expressed as a function of the dynamic pressure of the flow:

$$F_{jk} = -\int_S p_j n_k dS$$

But by definition, *'p =* $\frac{\ddot{y}\ddot{y}}{\ddot{y}t}$

Differentiating under the integral sign we get:

$$\frac{\ddot{y}\ddot{y}}{\ddot{y}t} = \ddot{x}j\ddot{y}j + \int_{\ddot{y}\ddot{y}}^{t} \frac{\ddot{y}\ddot{y}j\,(t\ddot{y}\ddot{y})}{\ddot{y}t}\,x\ddot{y}j\,(\ddot{y}\,)d\ddot{y}$$

It follows:

$$\ddot{y}\,Fjk = \ddot{y}\,C \quad x\ddot{}j\ddot{y}jnkdS + p \int_{\ddot{y}\ddot{y}\,C}^{t} \frac{\ddot{y}\ddot{y}j\,(t\ddot{y}\ddot{y})}{}nkdSx\ddot{y}j\,(\ddot{y}\,)d\ddot{y}\,\ddot{y}t$$

On pose :

$$A_{jk} = \rho\int_C\psi_j n_k dS \textrm{ (added masses)}$$

$$K_{jk}(t) = \rho\int_C\frac{\partial\phi_j(t)}{\partial t}n_k dS \textrm{ (fonctions de ret}$$

We then have:

$$\ddot{y}\,Fjk = \ddot{}xjAjk + \int_{\ddot{y}\ddot{y}}^{t} K(t\,\ddot{y}\,\ddot{y}\,)\,\ddot{y}xj\,(\ddot{y}\,)d\ddot{y}$$

2.31 Relationship between time and frequency formulations

to the road

The frequential formulation is written:

$$-F_R = M_A(\omega)\frac{d^2X}{dt^2} + B_r(\omega)\frac{dX}{dt}$$

The temporal formulation is:

$$\ddot{y}\,FR = A\,\mathrm{dt}2^{\frac{d\,2X}{}} + \int_{\ddot{y}\ddot{y}}^{t} K(t\,\ddot{y}\,\ddot{y}\,)\,\frac{dX}{dt}\,(t\,)dt$$

The potential codes provide the 'MA(ÿ)' and 'Br(ÿ)' matrices at any frequency, but what about the *'A'* and *'K'* matrices used by the temporal formulation?

Two years after Cummins, in 1964, Ogilvie proposed a method to determine the *'A'* and *'K'* matrices as a function of the *'MA'* and *'Br' matrices.* To do this, he considers that the motion of the solid is oscillating with a pulsation *'ÿ' :*

'X(t) = *cos(ÿt)'*

By substituting in the frequential formulation, one obtains:

'FR = ÿ 2MA(ÿ) cos(ÿt) + ÿBr(ÿ) *sin(ÿt)'*

With regard to the temporal formulation, we have:

'FR = ÿ 2A cos(ÿt) + *ÿ* $\int_{ÿÿ}^{t}$ K(t ÿ *ÿ)* sin(ÿt)dÿ '

By performing the change of variable *'ÿ* ÿ *t* ÿ ÿ on a :

'FR = ÿ 2A cos(ÿt) + *ÿ* $\int_{0}^{+ÿ}$ K(ÿ) sin(ÿ(t ÿ *ÿ))dÿ* '

By expanding ' sin(ÿ(t ÿ *ÿ))'* we get:

'FR = oh ² *A* ÿ $\frac{1}{oh}$ $\int_{0}^{ÿ}$ K(ÿ) sin(ÿt)dÿ cos(ÿt) + *ÿ* $\int_{0}^{+ÿ}$ K(ÿ) cos(ÿt)dÿ *sin(ÿt)'*

The following relations must therefore be valid for all *'ÿ' :*

'MA(ÿ) = *A* ÿ $\frac{1}{oh}$ $\int_{0}^{+ÿ}$ K(ÿ) sin(ÿ)dÿ '

B_r(\omega)=\int_0^{+\infty}K(\tau)\cos(\omega\tau)d\tau

By letting *'ÿ'* tend to infinity, we have:

A = M_A(\infty) = \lim_{\omega\rightarrow\infty}M_A(\omega)

'K' is obtained by taking the inverse Fourier transform of *'Br' :*

K(t) = \frac{2}{\pi}\int_0^{+\infty} B_r(\omega)\cos(\omega\tau)d\omega

2.32 Taking feedrate into account

2.32.1 Oscillation speed and average speed

All of the theory described so far is valid for a body in oscillation around a position of equilibrium. In practice, it is also applicable to a body in **translation at constant speed** (no rotation or acceleration so that the moving frame remains Galilean) in the horizontal plane, subject to ' $\frac{dX}{dt}$ corrections detailed in the following section. The speed ' to be used in the previous formulation is then the speed of oscillation, i.e. the instantaneous speed from which the average speed is subtracted:

\dot{X_b} = \dot{X} - V_s

On note *'Vs = X* $= \frac{dX}{dt}$ for ease of writing.

In xdyn, the average speed is calculated over a period T representative of radiation effects and determined by the user, which is also the period used for the convolution with 'K'. This behavior is controlled by the remove constant speed YAML key in the settings of this effort template.

2.32.2 Correction with feedrate

In the case of an oscillating body with **constant forward speed,** the boundary conditions are modified:

- The free surface condition contains additional terms proportional to the velocity,
- The slip condition on the body must integrate its forward speed in the term 'V0'.

If these conditions are applied from the resolution, the frequency results are dependent on the feed rate. On the other hand, if we neglect the coupling between the generation of waves linked to the advance and that linked to the radiation (which amounts to linearizing the free surface condition), we can obtain the frequency results at an arbitrary advance speed based on results at zero speed.

2.32.2.1 Correction on the frequency results The correction is applied to the *'MA'* and *'Br'* coefficients of the frequency domain as follows for a constant speed U along the x axis:

{M_A}_{i,5}' = {M_A}_{i,5} - \frac{U}{\omega^2} {B_r}_{i,3}

{M_A}_{i,6}' = {M_A}_{i,6} + \frac{U}{\omega^2} {B_r}_{i,2}

{B_r}_{i,5}' = {B_r}_{i,5} + U {M_A}_{i,3}

{B_r}_{i,6}' = {B_r}_{i,6} - U {M_A}_{i,2}

For more details on the development of calculus, see Jean Bougis' doctoral thesis (1980).

The same correction can be applied for a speed V along the y axis, and the composition of the two corrections is written by linearity:

{M_A}_{i,4}' = {M_A}_{i,4} + \frac{V}{\omega^2} {B_r}_{i,3}

{M_A}_{i,5}' = {M_A}_{i,5} - \frac{U}{\omega^2} {B_r}_{i,3}

{M_A}_{i,6}' = {M_A}_{i,6} + \frac{U}{\omega^2} {B_r}_{i,2} - \frac{V}{\omega^2} {B_r}_{i,1

{B_r}_{i,4}' = {B_r}_{i,4} - V {M_A}_{i,3}

{B_r}_{i,5}' = {B_r}_{i,5} + U {M_A}_{i,3}

{B_r}_{i,6}' = {B_r}_{i,6} - U {M_A}_{i,2} + V {M_A}_{i,1}

We can write more conveniently:

$$M_A(\omega,V_s) = {M_A}_0(\omega) + \frac{1}{\omega^2} {B_r}_0(\omega) \cdot L_s(V_s)$$

$$B_r(\omega,V_s) = {B_r}_0(\omega) - {M_A}_0(\omega) \cdot L_s(V_s)$$

Where *'Ls'* is a selection matrix that depends on the average speeds:

$$L_s(V_s) = \begin{bmatrix} 0 & 0 & 0 & 0 & \bar{V} \\ 0 & 0 & 0 & 0 & -\bar{U} \\ 0 & 0 & 0 & -\bar{V} & \bar{U} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.32.2.2 Modification of the temporal formulation The formulation in the time domain is broadly the same, except for certain assumptions which can no longer be applied. In particular, the *'Br'* coefficients do not necessarily tend towards zero for large pulsations, so we have:

$$\vec{F_{rad}} = -A_\infty(V_s) \cdot \ddot{X} - B_\infty(V_s) \cdot \dot{X_b} - \int_0^{+\i$$ where ' *Xÿ* b ' is the speed of oscillation (around the average speed) and:

$$K(\tau,V_s) = \frac{2}{\pi}\int_0^{+\infty} [B_r(\omega,V_s) - B_\infty(V_s)]\cos(\omega\ta$$

By using the previous correction and the notation *'A = MA0,ÿ',* we have:

$$K(\tau,V_s) = \frac{2}{\pi}\int_0^{+\infty} [{B_r}_0(V_s) - {M_A}_0(\omega) \cdot L_s(V_s)$$

$$K(\tau,V_s) = \frac{2}{\pi}\int_0^{+\infty} {B_r}_0(V_s) \cos(\omega\tau)d\omega - \frac{2}$$

$$K(\tau,V_s) = K_B(\tau) - K_A(\tau) \cdot L_s(V_s)$$

And :

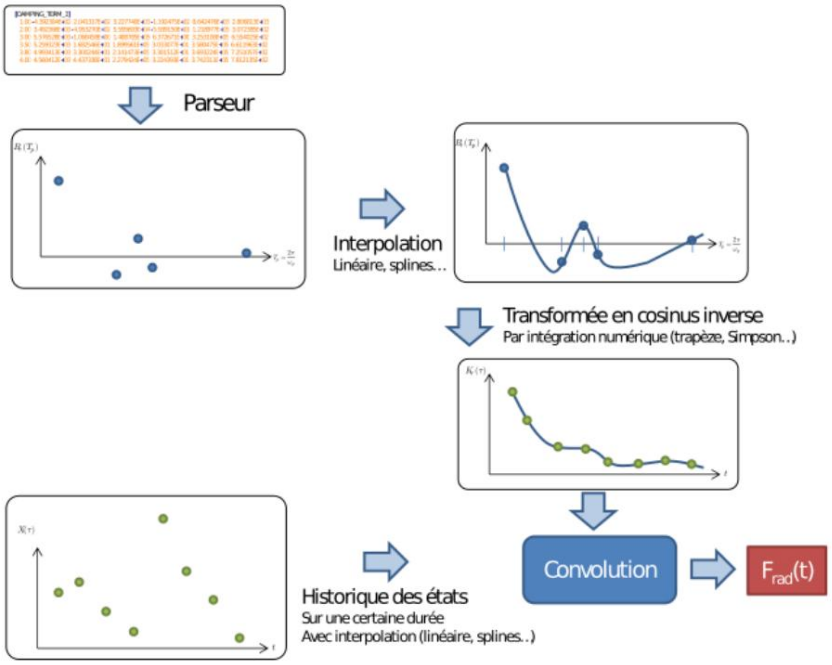
$$\vec{F_{rad}} = -A \cdot \ddot{X} + A \cdot L_s(V_s) \cdot \dot{X_b} - \int_0^{+\infty} K(\$$

2.33 Numerical calculation of the depreciations of radiation

In practice, one uses as entry of the simulator the HDB files (convention of basic hydrodynamic writing resulting originally from Diodore) or the files PRECAL_R, which contain the matrices of damping of radiation with various pulsations. These files are used in an interpolation table (either piecewise linear interpolation or splines) and then the following integral is evaluated for different values of *'ÿ '*

$$K_{i,j}(\tau) = \frac{2}{\pi}\int_{\omega_{\textrm{min}}}^{\omega_{\textrm{max}}}B_{i,j}(\o$$

This integral is calculated using a numerical integration scheme (method of rectangles, trapezoids, Simpson's rule or Gauss-Kronrod).



One then calculates the efforts of damping of radiation by taking into account the history over a period *'T' :*

$$F_{\textrm{rad}}(t)\sim\int_0^{T}\dot{X}(t-\tau)K_r(\tau)d\tau$$

It is important to note that these forces are expressed in the hydrodynamic calculation reference : a change of reference is necessary to express them in the “body” reference.

2.33.1 Settings

To use this model, write model: radiation damping. The radiation damping matrices are read from an HDB file (Diodore format) or a PRECAL_R file. This file contains the *'Br'* matrices for different periods. As the documentation indicates, the following steps are performed:

- Reading of the HDB file or the PRECAL_R file: its path is entered in the hdb (or raodb, respectively) key.
- Interpolation of damping function matrices: so-called “natural” splines are used, that is to say whose second derivative is zero at the ends or, which comes to the same thing, which are extended by straight lines at the ends of the domain.

- Calculation of delay functions by numerical integration: the integration algorithm is chosen by entering the key type of quadrature for cos tranform. Known integration types are: rectangle,

trapezoidal, simpson, gauss-kronrod, clenshaw-curtis, vein and burcher. The integration bounds are specified by omega min and omega max. If these bounds are not included in the interval ' *2ÿ* ', a warning message is displayed (because in this case the integration continues outside the domain of definition of the delay functions which are then extrapolated).

- Interpolation of the delay functions during the convolution: as for the damping functions, natural splines are used. The number of discretization points from which this interpolation is carried out (the number of values of *'ÿ* ' for which the integral is calculated *'Ki,j (ÿ) = Bi,j (ÿ)* · cos(ÿÿ)dÿ ') is given by nb of points for delay function discretization.

- State interpolation: when calculating the convolution integral, the states are interpolated linearly between two times. • Calculation of the convolution: the integration algorithm is specified by type of quadrature for convolution, which can take the same values as type of quadrature for cos transform.
- Verbosity: the calculation of the radiation damping efforts comprising many steps and being extremely sensitive to the integration limits and the types of algorithms used, we propose the display of results, namely the interpolated dampings on a finer grid than that provided as input (in order to control the errors due to the discretization) and the delay functions (in order to validate the limits of integration and the algorithm used). To enable verbosity, set the output Br and K key to true. Otherwise we set it to false.

- Using the speed of oscillation: when the boolean key remove constant speed is true, xdyn will subtract the average speed *'V* s' (calculated over the period tau max) from the total speed *'Xÿ* b' to use the speed of oscillation *'Xÿ* ' for the calculation of the effort of radiation. If this key is set to false, **xdyn uses the full speed.** This parameter is optional and its default value is false.

- Correction with forward speed: the boolean key forward speed correction can take the value true to apply the correction of the coefficients with forward speed. This key is optional and can therefore be omitted, the correction will then not be applied.

Here is an example of formatting using an HDB file:

```
- model: radiation damping hdb:
  test_ship.hdb type of
  quadrature for cos transform: simpson type of quadrature
  for convolution: clenshaw-curtis nb of points for retardation
  function discretization: 50 omega min: {value: 0, unit: rad/s} omega
  max: {value: 30, unit: rad/s} tau min:
  {value: 0.2094395, unit: s}
```

```
    tau max: {value: 10, unit: s} output Br
    and K: true remove constant
    speed: true forward speed
    correction: true
```

Here is the same data entry using a PRECAL_R file:

```
- model: radiation damping raodb:
    test_ship.ini type of
    quadrature for cos transform: simpson type of quadrature
    for convolution: clenshaw-curtis nb of points for retardation function
    discretization: 50 omega min: {value: 0, unit: rad/s} omega max: {value:
    30, unit: rad/s} tau min: {value: 0.2094395,
    unit: s} tau max: {value: 10, unit: s} output Br
    and K: true remove constant speed: true
    forward speed correction: true
```

The used sections of the potential calculation code output files are:

- For HDB files: Added_mass_Radiation_Damping, DAMPING_TERM_* subsections where * is an integer from 1 to 6 designating the axis (x, y, z, k, m, n, respectively).
- For PRECAL_R files: B_mimj where i and j are integers from 1 to 6 representing the positions in the matrices. These sections are present if the key sim > parRES > expAmasDampCoef of the PRECAL_R input XML file is true (or 1).

2.33.2 Rectangle method

It is the simplest method which consists in interpolating the function *'f'* to be integrated by a constant function (polynomial of degree 0).

If *'xi* ' is the interpolation point, the formula is:

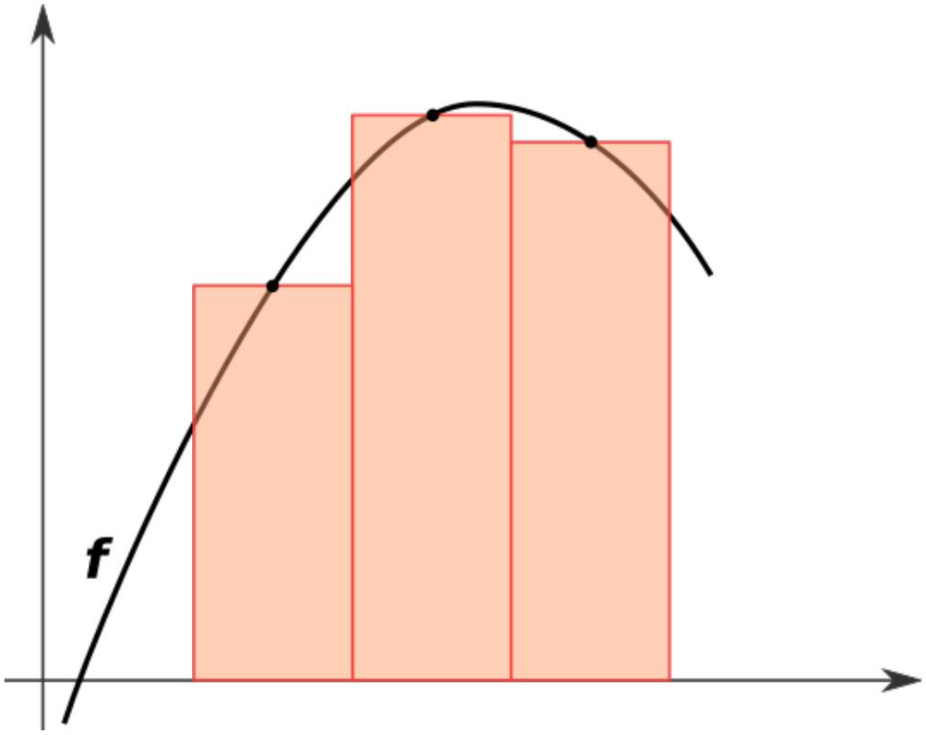
I(f) = (b-a) f(x_i)

The choice of *'xi* ' influences the error 'E(f) = *I* ÿ *I(f)' :* - If *'xi* = a' or *'xi* = b', the error is given by 'E(f) = (ÿ), *ÿ ÿ [a,* b]' This is the ''rectangle method" which is of order 0. - If *'ÿ = (a* + b)/2', the error is given by 'E(f) = is of order 1. $\frac{(b\ddot{y}a)^3}{24} f \ddot{y}\ddot{y}(\ddot{y}), \ddot{y}$ ÿ *[a,* b]'. This is the midpoint method which

Thus, the choice of the midpoint improves the order of the method: that of the rectangle is exact (i.e. 'E(f) = 0') for constant functions while that of the midpoint is exact for polynomials of degree 1. This is explained by the

fact that the integration gap of the midpoint method gives rise to two evaluation errors, of approximately equal absolute values and opposite signs.

Source: Wikipedia



Source: Wikipedia

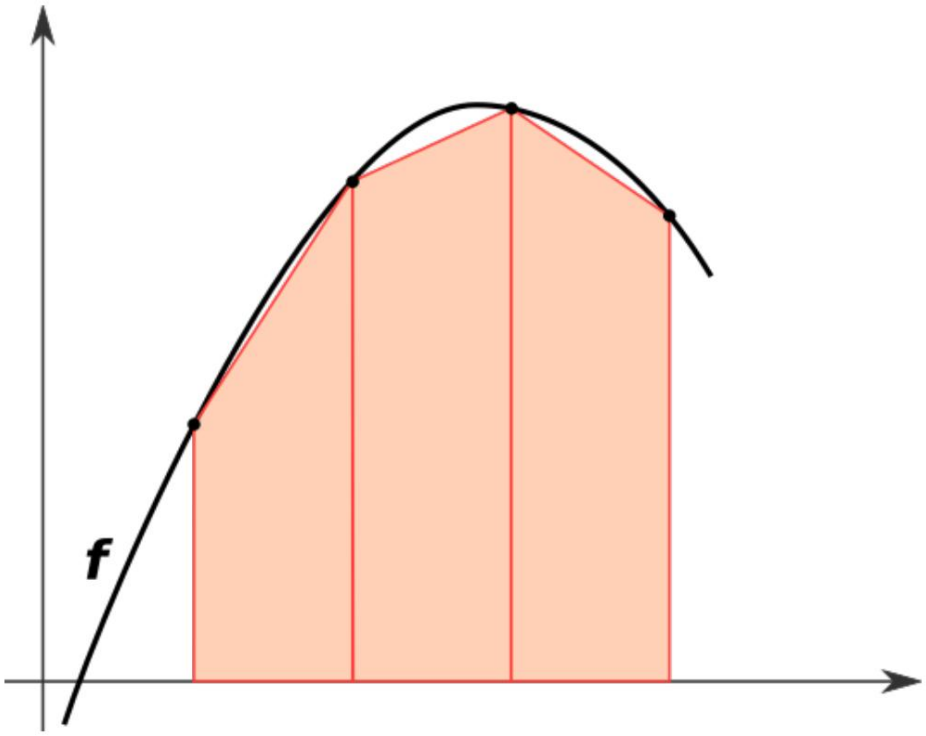
2.33.3 Trapezium method

By interpolating *'f'* by a polynomial of degree 1, the two interpolation points *'(a,* f(a))' and *'(b,* f(b))' suffice to draw a segment whose integral corresponds to the area of a trapezoid, justifying the name trapezium method which is of order 1 f(a) : :'I(f) = *(b* ÿ a) 2 $\frac{+f(b)}{}$

The error is given by: 'E(f) = ÿ $\frac{3\,(bÿa)}{12}$ *f ÿÿ(n), n* ÿ *[a,* b]'

According to the expressions of the error, the trapezium method is often less efficient than the midpoint method.

Source: Wikipedia



Source: Wikipedia

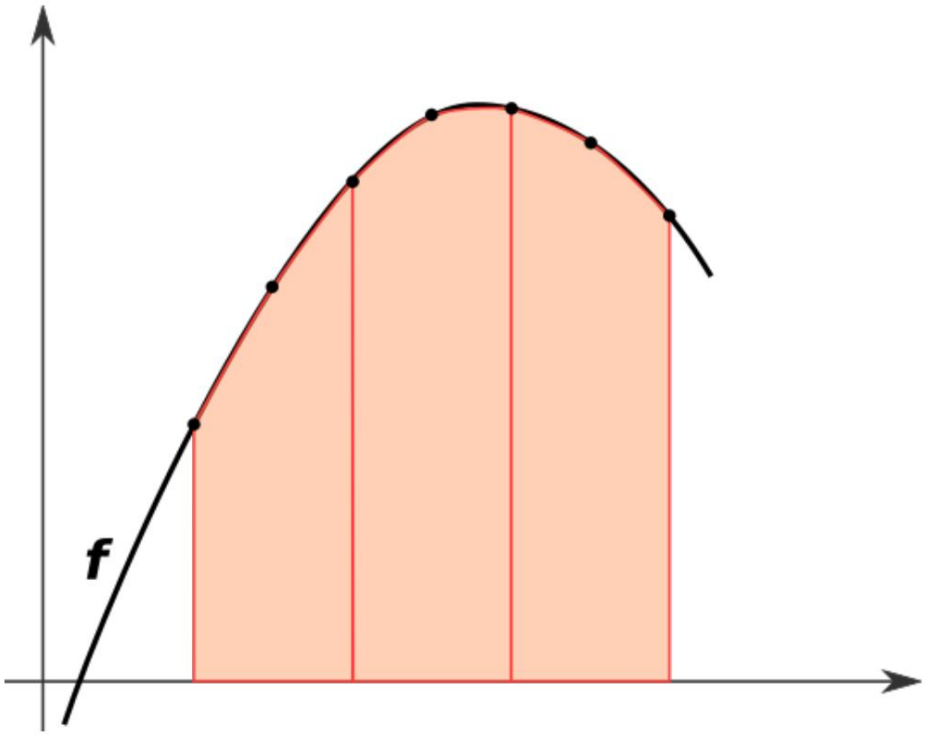
2.33.4 Simpson's rule

By interpolating *'f'* by a polynomial of degree 2 (3 degrees of freedom), 3 points (or conditions) are necessary to characterize it: the values at the extremities 'a', 'b', and the one chosen in their middle 'x1 /2 = *(a* + b)/2'. Simpson's method is based on a polynomial of degree 2 (integral of a parabola), while remaining exact for polynomials of degree 3; it is therefore of order 3: :'I(f) = (bÿa) / 6 [f(a) + 4f(x1/2) + f(b)]'

The global error is given by :'E(f) = ÿ 2880 (bÿa)^5 *f* (4)(ÿ), *ÿ* ÿ *[a,* b]'

Note: like the midpoint method which characterizes a polynomial of degree 0 and which remains exact for any polynomial of degree 1, Simpson's method characterizes a polynomial of degree 2 and remains exact for any polynomial of degree 3. of a sort of “anomaly” where beneficial compensations to the order of the method are produced.

Source: Wikipedia



Source: Wikipedia

2.33.5 Quadrature the Gauss-Kronrod

The Gauss-Kronrod quadrature formula is an extension of Gaussian quadrature. When a Gaussian quadrature is calculated on an interval and this interval is divided into two parts, none of the points cannot be reused (except the median point in the case of an odd number of points). The Gauss-Kronrod formula , created in the 1960s by Alexander Kronrod, makes it possible to transform an order *'n'* diagram into an order diagram '3n + 1' by adding to the *'n'* points of the Gaussian quadrature *' n*+ 1' zeros of the Stieltjes-Wigert polynomials.

The Stieltjes-Wigert polynomials are orthogonal polynomials for the weight function:

w(x)=\pi^{-1/2}\cdot k\cdot x^{k^2\log(x)},x\in\mathbf{R}_+^*, k>0

On pose

q_k=e^{-\frac{1}{2k^2}}

The Stieltjes-Wigert polynomials are then written:

p_0(x)=q^{1/4}

et'pn ,k(x) = $\frac{(ÿ1)nq^{\frac{n}{2}+\frac{1}{4}}_k}{ÿ\,(qk;qk)n}$ $\sum_{n=0}^{n}$ $q_k^{n^2}$ (ÿ *ÿq$\overline{kx}$) n* '

Or

k\in[1,n]

\left[\begin{array}{c}n\\\nu\end{array}\right]=\prod_{i=0}^{\nu-1}\frac{1-q^{n-i}}{1-q^{i+1

with *'q'* the Pochhammer symbol

$$\text{'}(q;\ a)n = \begin{cases} \prod_{d=0}^{n-1} (1 - cry\ ),\ n > 0 \\ 1,\ n = 0 \\ \prod_{j=0}^{|n|} (1 - qaj\ ),\ n < 0 \\ \prod_{j=0}^{\infty} (1 - qaj\ ),\ n = \infty \end{cases}$$

(coefficient 'q'-binomial)

In order to further accelerate the convergence, we use Gauss Kronrod integration repeatedly (since this method offers the advantage of being able to reuse the calculation points of the previous iteration) and we apply *the 'ÿ'* - Wynn's algorithm.

The Gauss-Kronrod formulas are implemented in standard numerical libraries such as those of Netlib (QUADPACK, in particular routine DQAGS).

- Weisstein, Eric W. "Stieltjes-Wigert Polynomial." From MathWorld-A Wolfram Web Resource.
- Szegö, G., **Orthogonal Polynomials,** 4th ed. Providence, RI: Amer. Math. Soc., p. 33, 1975. •

R. Piessens, E. deDoncker–Coffee, C. Uberhuber, D. Kahaner (1983). **Quadpack: a Subroutine Package for Automatic Integration;** Springer Verlag.

2.34 Computation of the matrices of added mass

In the simulator, the added mass matrix is either read directly from the YAML file, or extracted and linearly interpolated from an HDB or PRECAL_R file. For the calculation of the efforts of damping of radiation, one needs the mass added to infinite frequency. In order to guarantee the symmetry and the positive and defined character of the matrix (the coefficients tend to oscillate strongly in the vicinity of *'T* = 0'), we do not extrapolate the data from the HDB files to zero: we directly use the values read in the HDB file for the weakest period. It is assumed that the meshes used for the calculation of the added masses (resolution of the potential) are sufficiently fine for the result to have a meaning. The results given by PRECAL_R being already given with infinite pulsation, no interpolation is necessary.

In the output file of PRECAL_R, the mass matrices added according to the pulsation (which are not used by xdyn) are in signal sections such as:

[signal37]
name = A_m1m1
description = A_m1m1 - added mass coefficient mode 1,1

If the calcAmasDampCoefInfFreq key of the PRECAL_R parameter file is set to true, PRECAL_R writes the matrix of added masses with infinite pulsation (used by xdyn) in the [added_mass_damping_matrix_inf_freq] section.

If several advance speeds or several wave incidences are specified in the PRECAL_R file, only the matrix concerning the first of these speeds and the first of these incidences is taken into account (this is the matrix total_added_mass_matrix_inf_freq_U1_mu1).

2.35 References

- *Practical Source Code for Ship Motions Time Domain Numerical Analysis and Its Mobile Device Application,* 2011, Zayar Thein, Department of Shipping and Marine Technology, CHALMERS UNIVERSITY OF TECH NOLOGY, Göteborg, Sweden, page 18.

- *Study of diffraction-radiation in the case of an indeformable float driven by a constant average speed and stressed by a sinusoidal swell of low amplitude,* 1980, Jean Bougis, Thesis with a view to obtaining the grade of Doctor-Engineer, National School of Mechanics, Nantes, France.

3 Effort Models

In previous versions of xdyn, the forces applied to the ship were subdivided into uncommanded forces (which only require the ship states to be calculated) and commanded forces (which, in addition to the ship states, use variables called “ commands” supplied separately). Since then, these effort models have been unified (uncommanded efforts just have an empty list of commands).

This is why the external forces are now entered indifferently in the external forces or controlled forces section (for compatibility with the old xdyn models) in the form of a list of models with or without parameters. The only key common to all effort models is model : each model has its own parametrization otherwise (possibly no parametrization). Here is an example of an external forces section:

```
external forces:
  - model: gravity -
  model: non-linear hydrostatic (fast)
```

3.1 Filtered states

Some force models (example: forward resistance) are only consistent if they use averaged or filtered values as input (for example for forward resistance, average forward speed, excluding disturbance of the swell).

The definition of these filters is made using the optional filtered states section, sub-section of each bodies (therefore at the same level as the external forces key) of which here is a complete example:

filtered states:

x:

type of filter: moving average duration in seconds : 23

and:

type of filter: moving average duration in seconds : 2

With:

type of filter: moving average duration in seconds : 3

in:

type of filter: moving average duration in seconds : 4.879

in:

type of filter: moving average duration in seconds : 5.98

In:

type of filter: moving average duration in seconds : 78.63

p:

type of filter: moving average duration in seconds : 12.93

q:

type of filter: moving average duration in seconds : 6.3

r:

type of filter: moving average duration in seconds : 63

phi:

type of filter: moving average duration in seconds : 2.3

theta:

type of filter: moving average duration in seconds : 3

psi:

type of filter: moving average

```
        duration in seconds : 2.3
```

Each section x, y, z, u, v, w, p, q, r, phi, theta, psi is optional: thus, the following YAML is valid:

```
filtered states:
    x:
        type of filter: moving average duration in
        seconds : 23
```

If a state is not in the filtered states section, xdyn will use its unfiltered values (duration of the moving average equal to zero). If no filtered states section is defined, xdyn will always use unfiltered states . Moreover, the filtering of a variable is defined only once: all the models using a filtered state will use exactly the same definition.

Currently, only moving average filtering is implemented. The calculation of this moving average is done as follows:

- if the history of values contains only one value, we return this one
- we set *'T'* to the minimum of the integration time and the length of history available • if *'T* = 0', we return the last value of

the state (the current value at 't') • otherwise , we linearly interpolate the value of the state at *'t* ÿ T', where *'t'* is the current time • we carry out an integration of the state by the trapezium method (coherent

with the linear interpolation performed in the previous step) • The result is the value of this integral divided by *'T'*

For the Euler angles, the filtering is carried out on the four quaternions qr, qi, qj and qk then the filtered quaternions are used to find the values of the angles. A specific filtering is applied to the quaternions for each Euler angle.

Models that use these filtered states are:

- resistance curve
- remote effort models (gRPC)

Filtered states are available in xdyn output and in effort models (internal and gRPC): they are not provided to controllers or environment models and are not present in the “model exchange” interface and “co-simulation”. Indeed, these filtered states are in the hand of the creator of the xdyn model (hydrodynamic expert) and depend on the type of modeling. One thus decouples the filtering made for the needs of the hydrodynamic models and that made for the systems external to xdyn. Thus, if we modify the filtering linked to the force models, we will not modify the behavior of the controllers and vice versa.

An example of using filtered states is presented in tutorial 14.

3.2 Gravity forces

3.2.1 Description

The weight is given in the NED reference by:

'FP = m · **guz'**

where *'m'* designates the mass of the ship (in kg), *'g'* the acceleration of gravity (in m/s^2) and **'uz'** the vertical unit vector, expressed in the frame in which we want to project the force.

3.2.2 Settings

To subject a solid to gravity, we write:

- model: gravity

The value of *'g'* used is the one defined in the environmental constants section and for the mass we take the coefficient *'M3,3' (the* term in Z, therefore) of the inertial matrix.

An example of solid simulation subjected only to gravity (free fall) is available in the tutorials.

3.2.3 References

• *Physics v.* 1, 2001, D. Halliday, R. Resnick and KS Krane, John Wiley and Sons, ISBN 0-471-32057-9 • *Physique théorique,* Lev Landau et Evgueni Lifchits, éd. MIR, Moscow

3.3 Nonlinear hydrostatic forces

3.3.1 Description

The non-linear hydrostatic forces are due to the static pressure (that is to say independent of the speed of the fluid) exerted on the hull. The main hypothesis here is staticity, that is to say that we consider the hull at rest and a flat free surface. If this last assumption is not checked, it is necessary to add a corrective term, which corresponds to the efforts of excitation of Froude-Krylov. In other words, for the hydrostatic model, the calculation must be carried out assuming the free surface at rest. The taking into account of the pressure due to the swell is made by the Froude-Krylov model. The force 'FHS' exerted by the water on the hull must be calculated as the integral of the hydrostatic pressure 'pHS' over the total submerged surface:

$$\textbf{F}_{\textrm{HS}} = \int_{S}p_{\textrm{HS}}(z)\cdot \textbf{n} dS$$

The parameterization of non-linear hydrostatic forces in the simulator is described here.

An example of use is presented in the tutorials.

3.3.2 Calculation of the resultant of the hydrostatic forces integrated on shell

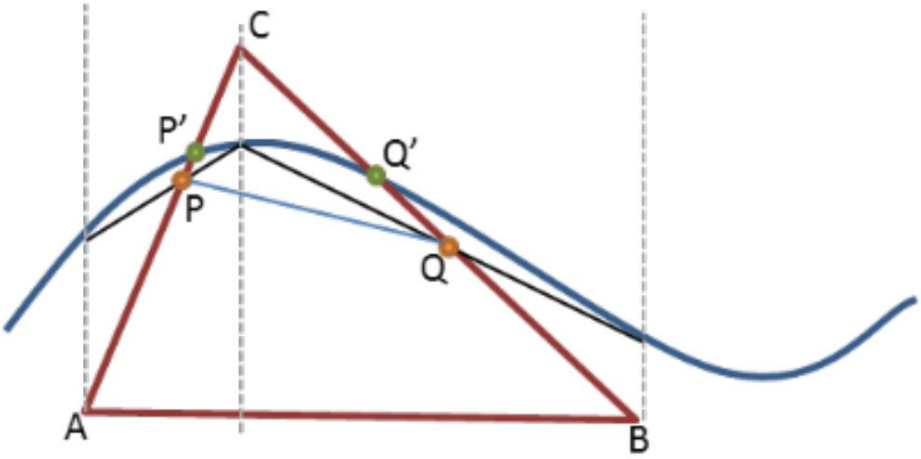
To numerically evaluate this integral, it is necessary to discretize the hull by means of a surface mesh. The definition of this mesh is made here.

The facets of the mesh can then be divided into three categories:

- totally emerged facets: their contribution is zero
- totally immersed facets: their contribution is *ÿ · g · dSzGn* where *'zG'* designates the projection of the centroid of the facet on the *'z'* axis (in m), **'n '** is the unit normal vector to the surface (oriented towards the outside, that is to say towards the fluid) and *'dS'* is the surface of the facet (in mˆ2)
- the partially immersed facets: they must be separate into a submerged part and an emerged part.

We do not choose, for *'zG',* the water height above the centroid: to do so would amount to including part of the Froude-Krylov excitation forces, which act as a correction if the free surface is not not plane.

When the partially immersed facets are separated into an emerged sub-facet and an immersed sub-facet, the situation is as follows:



The free surface is shown in blue. The real points of intersection are P' and Q', but as the calculation of the function *'ÿ'* representing the elevation of the free surface is expensive, we calculate the points P and Q, respective barycenters of the segments [AC] and [BC], assigned coefficients corresponding to the height of water above them. This amounts to approaching the free surface by a plane orthogonal to the facet and passing through P and Q. This approximation is all the more accurate as the meshes are small compared to the wavelength since for a free surface modeled at l order 1 by a monochromatic and unidirectional swell, we have:

$$\eta = \sin(\omega\cdot t - k\cdot x + \phi) = -k\cdot x + o(x)$$

So the error we make can be written in the form:

$$\epsilon(k,L) \sim 1-\cos\left(k\cdot\frac{L}{2}\right) \sim \frac{k^2\cdot L^2}{8}$$

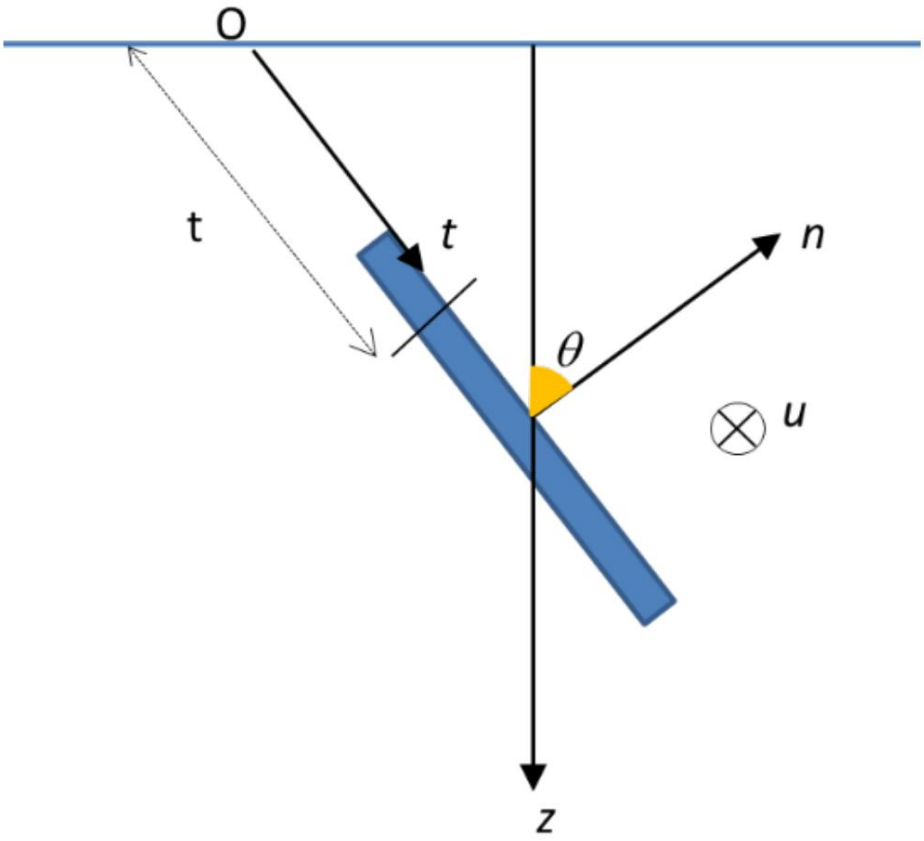
where *'k'* designates the wave number and *'L'* the characteristic dimension of the mesh.

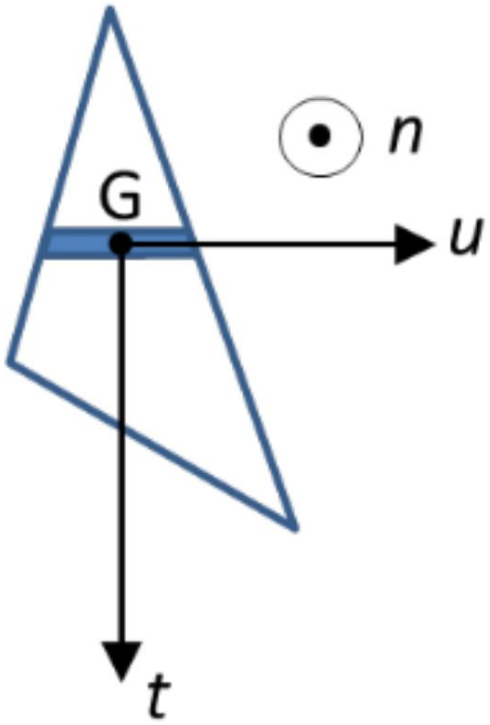
3.3.3 Calculation of the hydrostatic moment

For the calculation of the moment, it is necessary to know the point of application of each elementary force which is generally located below the centroid of the facet (except if the pressure is uniform, in which case these two points are confused).

One can either calculate exactly this point of application (we then obtain the **non-linear hydrostatic model (exact)),** or make the approximation that the point of application coincides with the centroid (thus giving the **non-linear hydrostatic** model **(fast)).**

To calculate the application point, we define the following notations:





One designates by *'tR'* and *'uR'* the coordinates of the point of application of the forces in the plane *('t,* u') and *'tG'* and *'uG'* the coordinates of the centroid of the facet in this same reference. The mark *('t,* u') is centered at the centroid of the facet.

The hydrostatic forces are written:

\textbf{F}_{\textrm{HS}} = \int_{P\in S} \rho\cdot g\cdot z(P) \textbf{dS}(P)

with 'z(P) = *t* sin ÿ'

from where

\textbf{F}_{\textrm{HS}} = \rho\cdot g\cdot \sin\theta \textbf{n} \int_S t dS

Or ‘ $_{S\,2}$*tdS = tdtdu = du tdt = U* · $\frac{T^2}{}$ *= OUT* · $\frac{T}{2}$ *= S* · *tG'*

from where

\textbf{F}_{\textrm{HS}} = \rho\cdot g\cdot t_G \cdot S \textbf{n}

'tR' must verify:

\int_S t dF = F\cdot t_R

either

t_R = \frac{\int_S t^2 dS}{t_G S}

Or ‘ $_S$ *t 2dS* = It(G) + *S* · *t* ²G so

where 'It(G)' is the second moment of inertia of the surface with respect to the axis parallel to *'t'* and passing through 'G'.

So we have :

t_R = t_G + \frac{I_t(G)}{t_G\cdot S}

Similarly, we find:

u_R = u_G + \frac{I_u(G)}{t_G\cdot S}

but *'uG* = 0' by definition of the reference *'(t,* u)', so

u_R = \frac{I_u(G)}{t_G\cdot S}

In practice, we find during simulations that the two models are quite close on the *'z'* axis since the amplitude of the force is identical in both cases. The differences are rather on the level of the moments and are all the more notable that the meshes are large compared to the solid (at the limit, when the surface of the facets tends towards zero, the two models coincide). The most obvious difference is obtained when we simulate the roll oscillations (i.e. around 'x') of a cube meshed by six right-angled triangles : we obtain for the fast model the following parasitic displacements *'y'* which do not appear with the exact model.

Nevertheless, the exact model implying the calculation of the matrices of inertia of each mesh (in particular meshes generated dynamically by calculating the intersection of the hull and the free surface), it is very expensive in time of calculation (one can note an order of magnitude compared to the fast model).

3.3.4 New hydrostatic model

This model uses a different approach: instead of integrating the efforts on all the facets, one calculates the immersed volume of the complete mesh and its centroid and one writes:

F_{\textrm{hs}} = \rho\cdot V\cdot \mathbf{g}

This model has the advantage of forcing the resultant to follow 'z'.

This model will eventually replace the non-linear hydrostatic (fast) and non-linear hydrostatic (exact) models because it is both more accurate than non-linear hydrostatic (exact) and as fast as non-linear hydrostatic (fast).

3.3.5 Settings

To use the fast model, we write:

- model: non-linear hydrostatic (fast)

for the specific model:

- model: non-linear hydrostatic (exact)

and for the new model:

- model: hydrostatic

The new model can also output the position (Bx, By, Bz) of the hull center (submerged). All you have to do is add Bx and/or By and/or Bz in the data list of the output section .

An example of solid simulation subjected to hydrostatic forces (oscillations in immersion) is available in the tutorials.

3.3.6 References

Introduction to fluid mechanics - CVG 2516, Statics of Fluids, Ioan
some

3.4 Efforts de Froude-Krylov

3.4.1 Description

The Froude-Krylov forces are part of the wave excitation forces. They correspond to the forces generated by the pressure field of the swell, assuming that the ship does not disturb the flow. They are calculated by integrating the dynamic pressure (pressure field of the incident swell) on the hull. In practice, they can be neglected as soon as the body is more than half a wavelength deep:

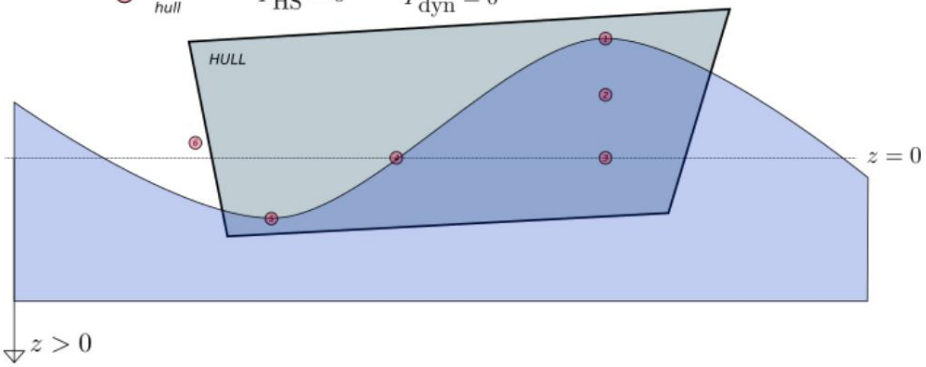
'FFK(t) = $\int_{P=(x,y,z)\in S}$ pdyn(x, *y, z,* t)dS(P)‘

The expression of the dynamic pressure depends on the wave model used and is described here (for the Airy wave).

The total pressure in the fluid, at a given point, is the sum of the hydrostatic pressure and the dynamic pressure. When we jointly use the hydrostatic model and the Froude-Krylov model, we are in the following situation:

$P_{\mathrm{HS}} = \rho g z \qquad P_{\mathrm{dyn}} = -\rho g \eta_0 \frac{\cosh(k(h-z))}{\cosh(kh)} \eta(x,y,t)$ *(without stretching)*

①	$z=-1$	$P_{\mathrm{HS}} = -\rho g$	$P_{\mathrm{dyn}} = +\rho g \frac{\cosh(h+1)}{\cosh(h)}$	$\eta(x,y,t)=1$
②	$z=-0.5$	$P_{\mathrm{HS}} = \frac{-\rho g}{2}$	$P_{\mathrm{dyn}} = +\rho g \frac{\cosh(h+0.5)}{\cosh(h)}$	$\eta(x,y,t)=1$
③	$z=0$	$P_{\mathrm{HS}} = 0$	$P_{\mathrm{dyn}} = +\rho g$	$\eta(x,y,t)=1$
④	$z=0$	$P_{\mathrm{HS}} = 0$	$P_{\mathrm{dyn}} = 0$	$\eta(x,y,t)=0$
⑤	$z=1$	$P_{\mathrm{HS}} = +\rho g$	$P_{\mathrm{dyn}} = -\rho g \frac{\cosh(h-1)}{\cosh(h)}$	$\eta(x,y,t)=-1$
⑥	*outside hull*	$P_{\mathrm{HS}} = 0$	$P_{\mathrm{dyn}} = 0$	



3.4.2 Settings

3.4.2.1 Non-linear version (integration of the pressures) To use this model, one inserts the following line in the section external forces:

- model: non-linear Froude-Krylov

3.4.2.2 Linear version This version uses the results of a potential calculation (AQUA+ or PRECAL_R) and performs the multiplication with the wave spectrum. Its parametrization is identical to that of the diffraction forces, also based on RAOs. The same remarks apply as for the diffraction forces (changes of reference and method of calculation), described below (section diffraction forces).

The section used for the HDB files is FROUDE-KRYLOV_FORCES_AND_MOMENTS, subsection INCIDENCE_FKFM_MOD_* for the modules (where * is a number designating the incidence) and INCIDENCE_FKFM_PH_* for the phases. For example :

```
[FROUDE-KRYLOV_FORCES_AND_MOMENTS]
[INCIDENCE_FKFM_MOD_001]  0.0000 3.500
        4.832189E+04 0.000000E+00 2.475141E+05 0.000000E+00 2.139539E+07 0.000000
        3.600 4.103861E+04 0.000000E+00 3.251938E+05 0.000000E+00 1.861110E+07 0.000000
[INCIDENCE_FKFM_PH_001]          0.0000
        3.500 -1.806307E+00 -3.141593E+00 2.527970E+00 -3.141593E+00 2.627867E+00 -3.141593
        3.600 1.320355E+00 -3.141593E+00 1.264921E+00 -3.141593E+00 1.403906E+00 -3.141593
        3.700 1.316362E+00 -3.141593E+00 2.356064E-01 -3.141593E+00 4.982495E-02 -3.141593
```

For the PRECAL_R files it is F_inc_m* (where * equals 1 to 6, for x, y, z, k, m, n), present when the option sim > parRES > expIncWaveFrc of the input XML file of PRECAL_R is set to true or 1.

In the external forces section we add (to use an HDB file):

```
- model: linear Froude-Krylov hdb:
    test_ship.hdb calculation
    point in body frame: x: {value: 0.696, unit:
          m} y: {value: 0, unit: m} z: {value:
          1.418, unit: m}

    mirror for 180 to 360: true
    use encounter period: true
```

or (to use a PRECAL_R file):

```
- model: linear Froude-Krylov raodb:
    test_ship.ini mirror for 180
    to 360: true use encounter period:
    true
```

- hdb or raodb : path to the frequency calculation results file.
  The path can be absolute or relative: if it is relative, it is relative to the folder from which xdyn is launched.
- mirror for 180 to 360 : used to be able to specify only the part of the CAD between '0ÿ ' and '180ÿ ', even if it means symmetrizing it with respect to the axis (Ox) to obtain the points between '180ÿ ' and '360ÿ '. In practice, this means that we take 'RAO(Tp, ÿ) = *RAO(T p, 2ÿ* ÿ ÿ)' if *'ÿ* > ÿ' and that mirror for 180 to 360 vaut true.
- calculation point in body frame : coordinates of the calculation point.
  Only for HDB files. For PRECAL_R files, the calculation point is necessarily the center of gravity. • use encounter period :
  Optional. Tells xdyn to calculate the encounter period *'Te'* to interpolate the diffraction strength from the HDB or PRECAL_R file. This option only works when using frequency results at zero speed.

3.4.3 References

- *Environmental Conditions and Environmental Loads,* April 2014, DNV RP-C205, Det Norske Veritas AS, page 47
- *Hydrodynamics of Offshore Structures,* 2002, Bernard Molin, Editions TECHNIP, ISBN 2-7108-0815-3, page 185
- *Sea Loads on Ships And Offshore Structures,* 1990, O. M. Faltinsen, Cam bridge Ocean Technology Series, ISBN 0-521-37285-2, pages 16, 39, 59 • *Seakeeping: Ship Behaviour in Rough Weather,* 1989, A. R. J. M. Lloyd, Ellis Horwood Series in Marine Technology, ISBN 0-7458-0230-3, page

67-68

- *Offshore Hydromechanics,* 2001, J.M.J. Journée and W.W. Massie, Delft University of Technology, sections 6-20 and 7-11

3.5 Diffraction forces

3.5.1 Description

The diffraction forces are due to the modification of the pressure field due to the presence of the ship. They are interpolated from hydrodynamic tables.
Like the radiation stresses and the added mass stresses, these tables are calculated by solving a boundary condition problem for the velocity potential: we therefore use codes based on potential methods, such as Aqua+.

The tables contain the force transfer functions (Response Amplitude Operators or RAO) and are parameterized in pulsation, incidence and forward speed. These are first-order effort RAOs). The diffraction forces are due to the diffraction of the swell by the fixed body, while the radiation damping comes from the dissipation of the energy of the body during its movement, by the creation of waves. This difference results only in a difference of boundary conditions in the potential modeling.

3.5.2 Numerical calculation

The force RAOs are read from an HDB or PRECAL_R file. This table gives, once interpolated, two RAO functions per axis *'k'*

$(u,\omega,\beta)\mapsto {RAO^{k}}_{\textrm{module}}(u,\omega,\beta)$

$(u,\omega,\beta)\mapsto {RAO^{k}}_{\textrm{phase}}(u,\omega,\beta)$ • 'u'

designates the longitudinal component of feed rate *(low* frequency) projected in the body reference (a current limitation of xdyn is that we use for these calculations not the low frequency speed but the instantaneous speed),

- *'ÿ'* designates the pulsation of the swell (and not the pulsation of encounter, since in the formula below we involve **'k** · x'),
- *'ÿ'* is the angle of incidence between the *'X'* axis of the vessel and the direction of wave propagation

To calculate the forces and the moments, for a wave with propagation direction ' ÿ', the RAOs are summed as for the calculation of the elevation of the free surface, following the AQUA+ convention:

'Fk(xH, *yH, t,* u) = ÿ , *ÿ* ÿ $\sum_{i=1}^{nf\,reg} \sum_{j=1}^{where} a_{i,j} \cdot \sin(k_i \cdot (xH \cdot \cos(ÿj) + yH \cdot \sin(ÿj)) ÿ ÿi \cdot t ÿ RAOk$ phase(u, *ÿi , ÿ* ÿ *ÿj) + ÿij)'

- *"xH"* and *"yH"* indicate the coordinates of the point of calculation of the forces hydrodynamics,

• *'t'* is the current time. • *'ai,j* '

is the amplitude of the swell, i.e.

$$a_{i,j}^2=A(\omega_i,\gamma_j)^2 = 2 S(\omega_i)d\omega D(\gamma_j) d\gamma$$

for an irregular wave - *'ÿi* ' and *'ÿj* ' correspond to the discretization of the wave spectrum. It should be noted that *'ÿi* ' does not correspondto the vessel's encounter pulse with the swell since this is calculated by AQUA+. The term **'k** · x' allows to take into account the speed of advance. - *'ÿij* ' random phase (see the Airy wave model). - *'u'* is the is a projection of the ship's speed relative to the NED terrestrial reference on the *'X'* axis of the body reference . - The term *'ÿ ÿ ÿj* ' corresponds to the calculation of the angle of incidence of the swell, respecting the Aqua+ convention, i.e. between 0 and ' *ÿ'* for a swell propagating to port and *'ÿ'* and *'2ÿ'* for a swell propagating to starboard (0 for a swell propagating from the stern to the front, ie swell from the astern, and *'ÿ'* for a swell propagating from the front to back, ie head swell).

The RAO read from the file not being modified, the previous expression gives a force torque expressed in a Z reference point upwards and at the point of calculation of the RAO: to express it in the Z reference point downwards of xdyn, one carries out the change of reference following:

$$\tau_{\textrm{HDB}} = \left[\begin{array}{c} F_X\\ F_Y\\ F_Z\\ M_X\\ M_Y\\ M_Z\\ \end{array} \right]$$

$$\tau_{\textrm{xdyn}} = \left[\begin{array}{c} F_X\\ -F_Y\\ -F_Z\\ M_X\\ -M_Y\\ -M_Z\\ \end{array} \right]$$

The calculated torsor is then moved by xdyn from the point of calculation of the files HDB or PRECAL_R to the point of resolution of the assessment of the forces.

3.5.3 Taking feedrate into account

The HDB and PRECAL_R files give the transfer function of the diffraction forces as a function of the incident wave period, at a given forward speed . These results therefore take into account the feed rate if applicable.

However, as for the added mass and radiation damping coefficients , it is simpler - when the feedrate is unknown *a priori* - to use zero feedrate results.

We then consider the period given in the HDB or PRECAL_R file as the meeting period *'Te* = ', with: $\frac{2\,p}{\ddot{y}e}$

$$\omega_e = \omega - \vec{V_s} \cdot \vec{k}$$

Where ' *ÿk'* is the wave vector, in the direction of wave propagation and of length *'k* = '. We then have the following expression for the diffraction forces $\frac{2p}{\ddot{y}mih}$

$$\text{'Fk(xH, } yH, t, \text{ u)} = \ddot{y} \sum_{i=1}^{nf\ req} \sum_{j=1}^{where} RAOk\ \text{module(u, } \ddot{y}e, \ddot{y}\ \ddot{y}\ \ddot{y}j\ );\ ai,j \cdot \sin(k \cdot (xH \cdot \cos(\ddot{y}j\ ) + yH \cdot \sin(\ddot{y}j\ ))\ \ddot{y}\ \ddot{y}i \cdot t\ \ddot{y}\ RAOk\ \text{phase(u, } \ddot{y}e, \ddot{y}\ \ddot{y}\ \ddot{y}j\ ) + \ddot{y}ij\ )'$$

This choice corresponds to an approximation which neglects the coupling between the wave field linked to the speed of advance and that of the diffracted swell.

3.5.4 Settings

3.5.4.1 With an HDB file To use this model, we write model: diffraction. The only parameter of this model is the path to the HDB file containing the first order stress RAOs. The corresponding section in the HDB file is DIFFRACTION_FORCES_AND_MOMENTS.

```
- model: diffraction
  hdb: test_ship.hdb
  calculation point in body frame: x: {value:
      0.696, unit: m} y: {value: 0, unit: m}
      z: {value: 1.418, unit: m}

  mirror for 180 to 360: true use
  encounter period: true
```

It should be noted that the calculation point does not appear in the HDB files, it must be entered in the YAML file (calculation point in body frame) but that no verification can be made.

The point of calculation of the diffraction forces is not necessarily the center of gravity, nor even the point of resolution of Newton's equation. On the other hand, it is necessarily a fixed point in the reference of the solid.

The parameter mirror for 180 to 360 is used to be able to specify only the part of the CAD between '0ÿ ' and '180ÿ ', even if it means symmetrizing it with respect to the axis (Ox)

to get the points between '180ÿ ' and '360ÿ '. In practice, this means that we take 'RAO(Tp, ÿ) = *RAO(T p, 2ÿ* ÿ ÿ)' if *'ÿ* > ÿ' and mirror for 180 to 360 is true.

The use encounter period boolean parameter is optional and tells xdyn to calculate the encounter period *'Te'* to interpolate the diffraction strength from the HDB file. This option is useful when using frequency results at zero speed.

3.5.4.2 With a PRECAL_R file One can also read these RAO from a PRECAL_R file. The PRECAL_R file must have been generated with the expIncWaveFrc key activated. The RAOs are then present in the F_dif_m[...] signals of the [RAOs] section.

```
- model: diffraction
    raodb: test_ship.raodb.ini mirror for
    180 to 360: true use encounter period:
    true
```

Contrary to the case of an HDB file, the calculation point is known: it is the center of gravity. You should therefore not fill in the calculation point in body frame field in the YAML file.

The other parameters are the same as in the case of the HDB file (shown above).

If several feedrates are specified in the PRECAL_R file, only the RAO concerning the first of these speeds are taken into account.

3.5.5 References

- *Notice d'utilisation AQUA+ 1.1/MF/N1,* septembre 1993, G. Delhommeau, ECN/LMF/DHN, pages 4 à 6. • *NEMOH Theory - General Notations and Conventions,* page 2 • *19th WEGMT School - Numerical Simulation of Hydrodynamics: Ships and Offshore Structures - Offshore structures - Seakeeping codes AQUADYN and AQUAPLUS, Gérard DELHOMMEAU* , page 10

3.6 Resistance to progress

3.6.1 Assumptions

We assume the rectilinear propulsion, of constant intensity and direction and located in the plane *('x',* 'y') of the NED frame.

It is also assumed that there is no swell (initially calm water), that the trim and the sinkage of the ship are constant and that its heel is nil. Otherwise , it would be necessary to interpolate the resistance along X according to the

speed, but also the vertical force and the pitching moment, but this model does not allow it.

Finally, it is assumed that the resistance to advancement is collinear with the projection on the horizontal plane of the propulsive force. Given these

assumptions, we speak of **towing resistance** at a given speed and we note *'RT '* the force necessary to tow the vessel at this speed in calm water.

3.6.2 Modeling

D'Alembert's paradox is that when we tow an object immersed in a supposedly perfect fluid and in an infinite medium, its resistance is zero. Experienced mentally, of course, we do not observe this phenomenon. That implies that

- water is not a perfect fluid: it has a viscosity which slows down the object by friction and by the production of turbulent structures,
- and/or the free surface is not in equilibrium and opposes the movement of the solid.

We therefore break down the towing resistance into two components:

- the viscous resistance, linked to the friction of the water on the hull and to the pressures related to detachments or turbulent structures;
- wave resistance, due to the creation of a wave field by the ship.

3.6.3 Resistance curve

3.6.3.1 Description A simple way to obtain the forward resistance force is to interpolate a curve of resistance as a function of the speed of the solid with respect to the NED frame projected on the X axis of the body frame (that we note 'u'). The forward resistance curve is obtained beforehand (and not calculated by xdyn) and entered in the YAML file.

If *'f : u ÿ R = f(u)'* designates the interpolation function of the forward resistance curve, the force torque, expressed at the hydrodynamic calculation point, is:

\tau_{\textrm{res}} =\left[\begin{array}{c}X\\Y\\Z\\K\\M\\N\end{array}\right] =\left[\begin

3.6.3.2 Settings This model is accessible via the resistance key curve.

The forward resistance forces are entered according to the *filtered* forward speed *'u' (longitudinal* axis only), i.e. the projection along the *'x'* axis of the body reference of the speed of the vessel relative to the NED fix.
The filtering is defined in the filtered states section, if it is filled in: in

otherwise, the model uses the unfiltered ' *u'* feedrate. The resistance curve interpolation is done using cubic splines.

- model: resistance curve

  speed: {unit: knot, values: **[0,1,2,3,4,5,15,20]}** resistance: {unit: MN, values: [0,1,4,9,16,25,225,400]}

This force is oriented along the axis ' ÿ x' of the reference body.

3.6.4 Model of Holtrop & Mennen

3.6.4.1 Description When a running resistance curve is not available , empirical models can be used (at the cost of reduced accuracy). This is particularly useful when the shape of the hull is not known, but you have access to the main dimensions of a ship.

The model proposed by Holtrop & Mennen in 1982 (then revised by Holtrop in 1984) is based on a regression on several hundred ship hulls. From the main geometric characteristics of the ship, it provides an estimate of the various components of the resistance force (wave resistance, friction, etc.) with an accuracy of around 20-30%.

This model is only valid for displacement vessels, within the limits of the vessels used during the regressions, ie large commercial vessels. Moreover, this model is based on hull shapes used in the years preceding the regressions, and is therefore not always adapted to recent changes in ship shapes.

3.6.4.2 Breakdown of forces The Holtrop & Mennen model breaks down the force of resistance to progress as follows:

F_{tot} = F_W + F_F + F_{App} + F_A + F_B + F_{Tr}

where: - *'FW* ' is the wave resistance - *'FF* ' is the friction force on the hull - *'FApp'* is the friction force on the appendages - *'FA'* is a model/full scale correlation component - *' FB'* is the force related to the bulbous bow - *'FT* r' is the resistance due to a partially submerged transom

For more details on the formulation, see the two articles in reference (Holtrop & Mennen 1982, Holtrop 1984).

3.6.4.3 Input parameters Even if the Holtrop & Mennen model does not require knowing the shape of the hull, a certain number of parameters must be entered. Some values are optional because they can be evaluated by empirical formulas derived from regressions, but it is preferable to use a known value when it is available.

- *'Lwl' :* length of the waterline • *'Lpp' :* length between the parallels • *'B' :* master-beautiful

• *'T* a': draft aft • *'T* f': draft draft • *'V ol' :* displacement (volume) • [optional] *'S' :* wet surface • *'lcb' :* center position waterline on the longitudinal axis, in % of *'Lwl',* with the midships origin (between the 2 parallels). For example, a ship of length *('Lwl = 50m')* whose center of buoyancy is placed 1.5m forward of the middle (between parallels), will have *'lcb* = 3 (negative values are possible). • *'Cm' :* midsection coefficient • *'Cwp' :* waterline coefficient • *'Abt' :* transverse area of the bulbous bow (zero if no bulbous bow) • *'hb' :* vertical center position of the area *'Abt'* in relation to the keel • *'At' :* submerged transverse area of the transom (zero if transom not submerged)

• *'Sapp' :* total wetted surface of the appendages • [optional] *'iE' :* horizontal stern half-angle (angle formed between the waterline at rest and the plane of symmetry of the ship)

• [optional] '1 + k1': shape coefficient of the hull • '1 + k2': total shape coefficient of the appendages. It is given approximately for each type of appendix, and must be added together to take several appendages into account.

Appendix	Shape factor 1+k2
Rudder on skeg	1.5 - 2.0
Rudder on stern Offset	1.3 - 1.5
suspended rudders with twin propellers 2.8 Shaft bracket 3.0 Skeg 1.5 - 2.0 Bulb shaft 2.0 Propeller shafts 2.0 - 4.0 Stabilizers 2.8	
Sonar dome 2.7 Anti-roll keels 1.4	

• *'Cstern' :* shape coefficient of the stern, the value of which is fixed for each type of stern.

Stern shape Coefficient Cstern
Barge form -25
Sections in 'V' -10
Sections standardes 0
'U' sections +10

3.6.4.4 Parameterization The Holtrop & Mennen model can be parameterized in xdyn as follows:

```
- model: Holtrop & Mennen
  Lwl: {value: 325.5, unit: m}
  Lpp: {value: 320, unit: m}
  B: {value: 58, unit: m}
  Ta: {value: 20.8, unit: m}
  Tf: {value: 20.8, unit: m}
  Vol: {value: 312622, unit: m^3}
  lcb: 3.48
  S: {value: 27194, unit: m^2}
  Abt: {value: 25, unit: m^2} hb: {value:
  2.5, unit: m}
  Cm: 0.998
  Cwp: 0.83
  At: {value: 0, unit: m^2}
  Sapp: {value: 273.3, unit: m^2}
  Cstern: 0
  iE: {value: 25, unit: deg} 1+k1: 1.5

  1+k2: 2
  apply on ship speed direction: true
```

The boolean key apply on ship speed direction projects the force on the direction of the ship's speed vector rather than on the longitudinal axis.

This is not normally recommended because the model assumes velocity only along the longitudinal axis. On the other hand, this makes it possible to simulate to a lesser extent lateral forces in the absence of an additional model and when the ship is drifting (note however, the lateral forces obtained have not been subject to any validation!).

3.6.5 References

- *Ship Dynamics,* 1986, P. Devauchelle, Library of the French Institute for Aid to Professional Maritime Training, ISBN 2-225-80669-1 , pages 58, 81 and 97
- *Naval hydrodynamics: theory and models,* 2009, A. Bovis, Presses de ENSTA, page 205 and 337
- *An approximate power prediction method,* 1982, J. Holtrop et G.G.J. Men nen, International Shipbuilding Progress, vol. 29, no. 335, pp. 166-170. • *A statistical re-analysis of resistance and propuslion data,* 1984, J. Holtrop, International Shipbuilding Progress, vol. 31, no. 363, pp. 272-276.

3.7 Forces of viscous damping

3.7.1 Description

The movements of a solid evolving in a fluid are damped due to the energy that this solid communicates to the fluid. These dissipative forces come on the one hand from the waves generated by the movements of the fluid (and which correspond to the damping of radiation), and on the other hand from the viscous damping due to the shearing of the fluid on the hull (appearance of a vortex wake or turbulent which dissipates energy, essentially on the roll axis). It is the latter that interest us in this section.

Non-viscous dampings (radiation) are, because of the modeling used to obtain them, linear compared to the speed. Viscous dampings are frequently modeled by a quadratic velocity law.

The linear damping model should only be used to take into account the dissipative forces of radiation, if these are not already obtained by the frequency calculation (by the radiation damping model which uses the hydro database of the HDB file). Along the axes, some terms predominate over others. For movements of small amplitudes, linear forces are preponderant. For large movements, it is the reverse.

In addition to their physical significance, the terms dampings also have an incidence on the simulation insofar as they tend to stabilize the diagrams of explicit numerical integration (type Runge-Kutta for example).

When the surge damping and drag resistance models are used together, additional precautions should be taken so as not to model the same physical phenomenon twice. It is therefore necessary to decompose the longitudinal speed into a low frequency component (used by the forward resistance model) and a high frequency component (for the damping model). This decomposition is not yet implemented in xdyn.

3.7.2 Modeling

For a description of the notations adopted here, reference may be made to the description of the hydrodynamic calculation benchmark.

The speed of the current (speed of the water relative to the NED reference, projected into the NED reference) is noted:

$$'\text{Calf/ground} = \begin{pmatrix} \text{Ucourant} \\ \text{Vcourant} \\ 0 \end{pmatrix}$$

We define :

$$\nu_{\textrm{local}} = {}^{\textrm{local}}T_{\textrm{body}} \nu_b - {}^{\textrm{local}}T_{\textrm{NED}}V_{\textrm{eau/sol}}$$

\omega_{\textrm{local}} = {}^{\textrm{local}}T_{\textrm{body}}\omega_{nb}^b

If the efforts of radiation are not if these are already obtained by the frequential calculation (by the model of damping of radiation which uses the database hydro of the file HDB), the linear dampings are written (in the reference hydrodynamic calculation):

F_{\textrm{al}}=-D_l\left[\begin{array}{c}\nu_{\textrm{local}}\\\omega_{\textrm{local}}\end

where *'Dl* ' is the linear damping matrix read from the parameter file.

For the quadratic dampings:

F_{\textrm{aq}}=-D_q(\nu_{\textrm{local}})\left[\begin{array}{c}\nu_{\textrm{local}}\\\omeg

Or

$$
\text{'Dq(ÿlocal)} = \begin{bmatrix} d11 \cdot |ulocal| & d12 \cdot |vlocal| & d13 \cdot |wlocal| & d14 \cdot |plocal| & d15 \cdot |qlocal| & d16 \cdot |rlocal| \\ d21 \cdot |ulocal| & d22 \cdot |vlocal| & d23 \cdot |wlocal| & d24 \cdot |plocal| & d25 \cdot |qlocal| & d26 \cdot |rlocal| \\ d31 \cdot |ulocal| & d32 \cdot |vlocal| & d33 \cdot |wlocal| & d34 \cdot |plocal| & d35 \cdot |qlocal| & d36 \cdot |rlocal| \\ d41 \cdot |ulocal| & d42 \cdot |vlocal| & d43 \cdot |wlocal| & d44 \cdot |plocal| & d45 \cdot |qlocal| & d46 \cdot |rlocal| \\ d51 \cdot |ulocal| & d52 \cdot |vlocal| & d53 \cdot |wlocal| & d54 \cdot |plocal| & d55 \cdot |qlocal| & d56 \cdot |rlocal| \\ d61 \cdot |ulocal| & d62 \cdot |vlocal| & d63 \cdot |wlocal| & d64 \cdot |plocal| & d65 \cdot |qlocal| & d66 \cdot |rlocal| \end{bmatrix}
$$

the *"(dij)"* being the coefficients of the quadratic damping matrix read from the parameter file.

3.7.3 Settings

The parameterization of the linear damping forces is made by a matrix informed in the following way:

```
- model: linear damping
   damping matrix at the center of gravity projected in the body frame: row 1: [ 0, 0,  0,  row
        2: [ 0, 0,  0,  row 3: [ 0, 0,          0,          0, 0]
        1.9e5,  row 4: [ 0, 0,  row              0,          0, 0]
        5: [ 0, 0,  row 6: [ 0, 0,               0,          0, 0]
                                0, 1.74e4,       0,          0, 0]
                                0,               0, 4.67e6, 0] 0,
                                                            0, 0]
```

This matrix is the *'Dl* ' matrix described in the documentation.

The parametrization of the quadratic damping forces is made by a matrix informed in the following way:

```
- model: quadratic damping
   damping matrix at the center of gravity projected in the body frame: row 1: [ 0, 0,  0,  0,
        row 2: [ 0, 0,  0,  0,  row 3: [ 0, 0,          0, 0]
        1.9e5,  0,  row 4: [ 0, 0,  0, 1.74e4,          0, 0]
                                                         0, 0]
                                                         0, 0]
```

```
row 5: [ 0, 0,  row      0,      0, 4.67e6, 0] 0,
6: [ 0, 0,               0,                 0] 0,
```

This matrix is the *'(dij)'* matrix described in the documentation.

3.7.4 References

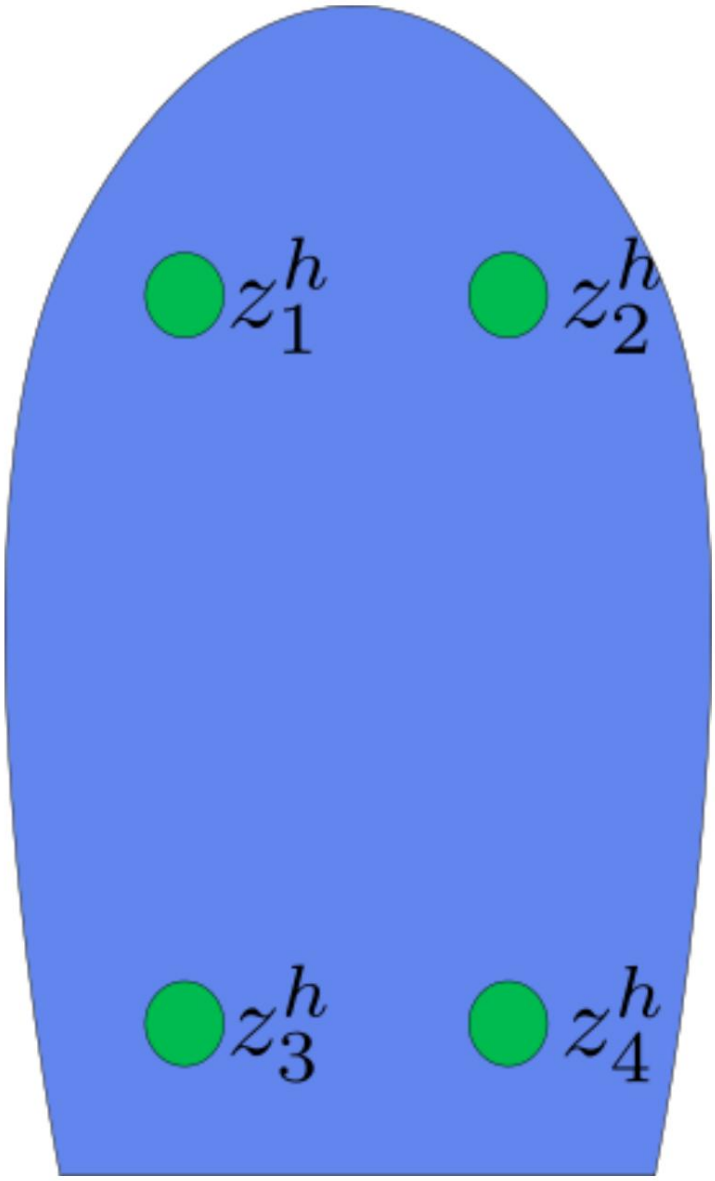
- *Hydrodynamics of Offshore Structures,* 2002, Bernard Molin, Editions TECHNIP, ISBN 2-7108-0815-3, page 276
- *Sea Loads on Ships And Offshore Structures,* 1990, O. M. Faltinsen, Cam bridge Ocean Technology Series, ISBN 0-521-37285-2, page 223 •

Seakeeping: Ship Behaviour in Rough Weather, 1989, A. R. J. M. Lloyd, Ellis Horwood Series in Marine Technology, ISBN 0-7458-0230-3, page 223 • *Marine Control Systems: Guidance, Navigation and Control of Ships, Rigs and Underwater Vehicles,* 2002, THor I. Fossen, Marine Cybernetics, ISBN 82-92356-00-2, page 71

3.8 Linear hydrostatic forces

3.8.1 Description

The linear hydrostatic forces model is much faster to calculate than its non-linear counterpart. Usually, the linear hydrostatic forces are just calculated from a hydrostatic matrix given as input (or possibly calculated from the position of the center of gravity and geometric data). The four-point implementation presented here is very specific.



The following variables are used:

\overline{z} = \frac{1}{4}\sum_{i=1}^4 z_i^h

$\overline{'\ddot{y}} = \frac{1}{2}\left(\text{atan}(z\ d12_{z}^{h\,h\,\ddot{y}z\,1}) \text{atan}(z + d43_{4}^{h\,h\,\ddot{y}z\,3})\right)$ $\overline{' \ddot{y}} = \frac{1}{2}\left(\text{atan}(z\ d24_{z}^{h\,h\,\ddot{y}z\,4}) \text{atan}(z + \frac{h\,h\,\ddot{y}z\,3}{1})\right)$

d13

where *'dij = (x* $\sqrt{}$ $_{i}^{h\,h\,\ddot{y}\,x\,j})^2 + (and_{i}^{h\,h\,\ddot{y}\,y\,j})$)' is the distance between two measurement points.

The force torque is given in the NED reference by:

‘Fhs = K3×3 ÿ *z* ÿ *z*¯ÿ zeq ÿ
ÿ ÿ *ÿ*¯ÿ ÿeq
ÿ *ÿ* ÿ *ÿ* ÿ ÿeq ÿ

'zeq, ÿeq, ÿeq' are the equilibrium values entered in the parameter file .

3.8.2 Settings

```
- model: linear hydrostatics z eq:
  {value: 0, unit: m} theta eq:
  {value: 0, unit: deg} psi eq: {value: 0,
  unit: deg}
  K row 1: [1, 0  0]
  K row 2: [0, 1  0]
  K row 3: [0, 0  1] x1:,
  {value: 10, unit: m} y1: {value:
  -10, unit: m} x2: {value: 10, unit:
  m} y2: {value: 10, unit: m} x3:
  {value: -10, unit: m} y3: {value:
  -10, unit: m} x4: {value: -10, unit:
  m} y4: {value: 10, unit: m}
```

The coordinates *'(xi , yj)'* are given in the body reference. The coefficients of the matrix *'K'* are given in SI unit.

3.9 Effort constant

3.9.1 Description

This model makes it possible to add to the balance of forces a constant force in the chosen reference mark. For example, one can define a constant force in the NED frame to simulate a wind force, or even define a constant force in the body frame to approximate a propulsion force.

3.9.2 Settings

```
- model: constant force frame:
    TestShip x: {value:
    0, unit: m} y: {value: 0, unit:
    m} z: {value: 0, unit: m}

    X: {value: 10, unit: kN}
    Y: {value: 20, unit: kN}
    Z: {value: 30, unit: kN}
    K: {value: 100, unit: kN*m}
```

M: {value: 200, unit: kN*m}
N: {value: 300, unit: kN*m}

The coordinates of the point of application of the force are noted x, y and z, expressed in the reference designated by frame. The coordinates X, Y, Z, K, M and N of the force torque are also expressed in this reference.

3.10 Model of quadratic hydrodynamic effort by polar

3.10.1 Description

The forces of the fluid on an object in relative motion with respect to this fluid, in a plane parallel to this flow, can be represented in a quadratic way using lift and drag coefficients:

$$F_l = \frac{1}{2} \rho C_l(\alpha) S_{ref} U^2$$

$$F_d = \frac{1}{2} \rho C_d(\alpha) S_{ref} U^2$$

Or :

• *'Fl* ' is the lift force, perpendicular to the direction of the flow, • *'Fd'* is the drag force, parallel to the direction of the flow, • *'ÿ'* is the density of the fluid , • *'Cl* ' and *'Cd'* are the lift and drag coefficients (respectively), • *'ÿ'* is the angle of attack, • *'Sref* ' is a reference surface (usually a projected surface),
• *'U'* is the velocity of the flow, depending on the velocity of the body and the that of the fluid.

We can also represent the moment exerted by the fluid on the object, always in the same plane:

$$M = \frac{1}{2} \rho C_m(\alpha) S_{ref} d_{ref} U^2$$

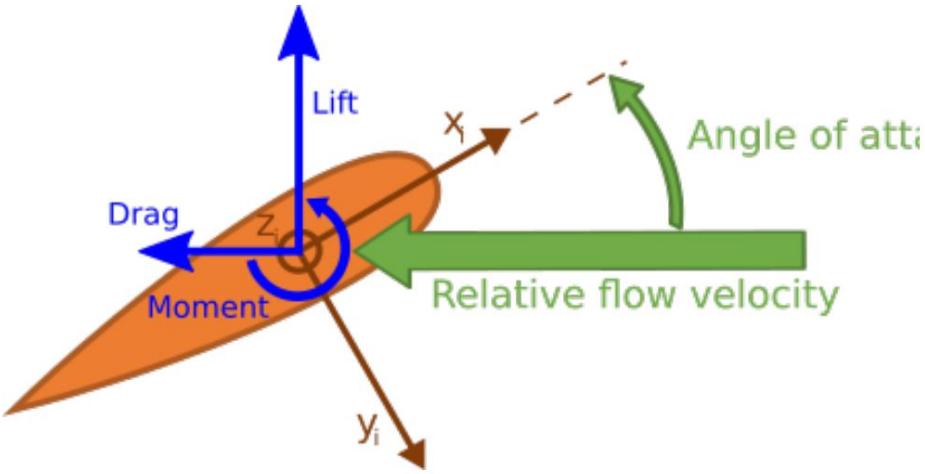
Or :

• *'M'* is the moment exerted by the fluid flow, • *'Cm'* is the moment coefficient, • *'dref* ' is a reference length used to dimension the moment, usually the chord of a supporting profile, sometimes ' *Sref* '.

3.10.2 Assumptions

In xdyn, such a model is implemented to simulate an interaction with water (anti-drift plan, hydrofoil, hull...). One defines a fixed local reference compared to a known reference (generally that of the body), and one calculates the speed of the flow and the resulting forces at the origin of this local reference and in the plane '(ÿxi , ÿyi) '.

The angle of attack *'ÿ'* and the relative flow velocity *'U = |Vÿ |'* are then defined in the plane *'(ÿxi , ÿyi)'* of the local coordinate system:



The angle of attack 'ÿ', defined between -180° and 180°, is formed between the relative velocity projected in the plane *'Vÿ '* and the longitudinal axis *' ÿxi '* of the local coordinate system.

It should be noted that the coefficients *'Cl '*, *'Cd'* and *'Cm'* are generally very dependent on the flow regime. xdyn does not check the Reynolds number, it is up to the user to ensure that the coefficient polars are relevant with regard to the flow conditions. The flow velocity *'U'* and the angle of attack *'ÿ'* are available at the output of the force model, under the respective names U(model_name,body_name) and alpha(model_name,body_name).

3.10.3 Settings

The quadratic hydrodynamic effort model by polar is parameterized in xdyn with the following YAML section:

```
- model: hydrodynamic polar name:
    centreboard position
    of calculation frame: frame: body x:
          {value: 1, unit:
          m} y: {value: 2, unit: m} z:
          {value: 3, unit: m} phi:
          {value: 10, unit: deg} theta:
          {value: 20, unit: deg} psi: {value:
          30, unit: deg} reference area: {value:
          1000, unit: m^2} angle of attack:
    {unit: deg, values: [0,7,9,12,28,60,90,120,150,180]}
    lift coefficient: [0.00000,0.94828,1.13793,1.25000,1.42681,1.38319,1.26724,0.93103,0.3879
```

drag coefficient: [0.03448,0.01724,0.01466,0.01466,0.02586,0.11302,0.38250,0.96888,1.3157 take waves orbital velocity into account: false

Polar data of lift and drag coefficients can be given from -180° to 180° or from 0° to 180°. In this second case, an assumption of symmetry according to the longitudinal axis of the local coordinate system ' *ÿxi* ' is applied, which is useful for the symmetrical profiles.

The boolean key take waves orbital velocity into account adds the wave orbital velocity to the relative flow. With this setting enabled, a swell model must also be defined in the environment models section.

The optional keys moment coefficient (vector of coefficients) and chord length (length in {unit: value: ...}) make it possible to add the calculation of a moment around the origin of local coordinate system and according to *'ÿzi* '. This moment is always destabilizing for positive *'Cm' (therefore* positive around *'ÿzi* ' for *'ÿ* > 0' and negative for *'ÿ* < 0'). If chord length is specified, its value is used for *'dref* ', otherwise *'dref* = *Sref* '.

3.11 Quadratic aerodynamic force per inch model lar

3.11.1 Description

Wind forces on a structure can be represented quadratically using lift and drag coefficients:

$$F_l = \frac{1}{2} \rho_{air} C_l S_{ref} U^2$$

$$F_d = \frac{1}{2} \rho_{air} C_d S_{ref} U^2$$

Or :

- *'Fl* ' is the lift force, perpendicular to the direction of the airflow ,
- *'Fd'* is the drag force, parallel to the direction of the airflow, • *'ÿair'* is the density of the air, and *'Cd'* are the lift and drag coefficients (respectively), • *'Cl* ' is a reference surface (usually a projected surface), • *'Sref* ' • *'U'* is the speed of the airflow, depending on the speed of the body and from that of the wind

3.11.2 Assumptions

This force model is essentially valid for movements in the horizontal plane. The coefficients given by the user are valid for an attitude of the simulated body. It is therefore relevant to use this model for objects whose movements outside the horizontal plane are small, such as a

ship on calm sea. This model can be used to simulate the effect of wind on topsides or on a wind propulsion system.

In addition, some implementation details should be noted:

- The speed of the flow *'U'* corresponds to the relative speed of the air flow (wind and speed of the body), calculated at a point of the body specified in entry, and projected in the horizontal plane of the reference specific to the body ("body" tag); • The lift force is oriented in a favorable way to propulsion for a positive *'Cl'* coefficient ; • The drag force is oriented in the direction of the flow for a coefficient *'cd'* positive.

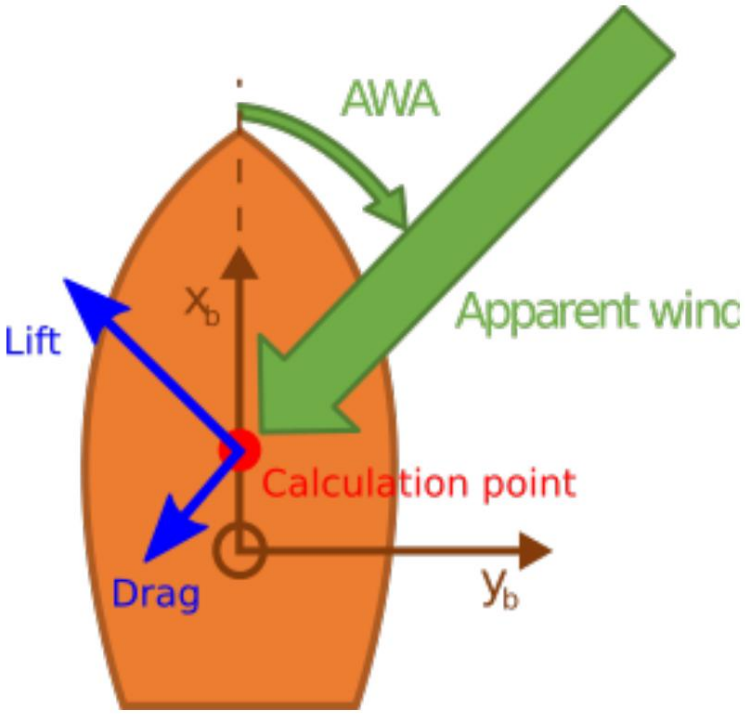
Finally, the calculation point given as input has two functions. It is on the one hand the point of calculation of the relative wind: the wind at its position and its speed are used. It is on the other hand the point of application of the force in the reference linked to the body.

3.11.3 Settings

The quadratic aerodynamic force model by polar is parameterized in xdyn with the following YAML section:

```
- model: aerodynamic polar name:
  main sail
  calculation point in body frame: x: {value:
      1, unit: m} y: {value: 2, unit:
      m} z: {value: 3, unit: m}
      reference area: {value: 1000,
  unit: m^2}
  AWA: {unit: deg, values: [0,7,9,12,28,60,90,120,150,180]} lift coefficient:
  [0.00000,0.94828,1.13793,1.25000,1.42681,1.38319,1.26724,0.93103,0.3879 drag coefficient:
  [0.03448,0.01724,0.01466,0.01466,0.02586,0.11302,0.38250,0.96888,1.3157
```

The AWA key (for Apparent Wind Angle) corresponds to the apparent wind angle, that is to say the angle of incidence of the flow in the frame specific to the body. A zero angle corresponds to a flow along the axis ' ÿ ÿx0' (of the proper benchmark, i.e. a headwind), an angle of 90° a flow along the angle ' ÿ ÿy0' (therefore a crosswind to starboard). . .



Polar data of lift and drag coefficients can be given from 0° to 360° or from 0° to 180°. In this second case, an assumption of symmetry according to the longitudinal axis of the body (' ÿx0' of the clean reference mark) is applied.

3.11.4 References

- Wikipedia page “Force on a sail”

3.12 MMG Maneuverability Model

3.12.1 Description

The maneuverability model proposed by the Maneuvering Modeling Group is one of the few widely used maneuverability models. It is based on the prior calculation of the hydrodynamic coefficients of a ship's hull, using experimental or advanced numerical methods (CFD). Once these coefficients have been obtained, the forces on the hull in the horizontal plane are calculated by polynomials.

3.12.2 Settings

This model is parameterized by the following YAML section:

```
- model: MMG maneuvering
  calculation point in body frame:
```

```
      x: {value: -11.1, unit: m} y: {value:
      0, unit: m} z: {value: 0, unit:
      m}
  Lpp: {value: 320, unit: m}
  T: {value: 20.8, unit: m}
  Xvv: -0.04
  Xrr: 0.011
  Xvr: 0.002
  Xvvvv: 0.771
  Yv: -0.315
  The: 0.083
  Yvvv: -1.607
  Yrvv: 0.379
  Yvrr: -0.391 Yrrr:
  0.008 Nv:
  -0.137 Nr:
  -0.049 Nvvv:
  -0.03 Nrvv:
  -0.294 Nvrr:
  0.055
  Nrrr: -0.013
```

3.12.3 References

- *Introduction of MMG standard method for ship maneuvering predictions,* 2014, H. Yasukawa, Y. Yoshimura, Journal of Marine Science and Technol ogy 20(1):37-52, doi:10.1007/s00773-014-0293-y. ## Efforts commandés

The forces commanded correspond, for example, to the propulsion, rudder and foil forces. They are also described in the section external forces (or controlled forces, for compatibility with older versions of xdyn).
The only YAML keys common to all ordered efforts are name (which is an identifier chosen by the user) and model (which is a string used to identify the type of model used).

Commands are specified in YAML, either in the same file as the effort templates, or in a separate (more modular) file.

Here is an example of commanded forces, which corresponds to the propeller model described here:

```
external forces:
  - name: port side propeller
    model: wageningen B-series position
    of propeller frame: frame:
        mesh(TestShip) x: {value:
        -4, unit: m} y: {value: -2, unit:
        m}
```

```
      z: {value: 2, unit: m} phi: {value:
      0, unit: rad} theta: {value: -10, unit:
      deg} psi: {value: -1, unit: deg} wake
      coefficient w: 0.9

  relative rotative efficiency eta: 1 thrust deduction
  factor t: 0.7
  rotation: clockwise
  number of blades: 3
  blade area ratio AE/A0: 0.5 diameter:
  {value: 2, unit: m}
- name: starboard propeller
  model: wageningen B-series position
  of propeller frame: frame: mesh(TestShip)
      x: {value: -4, unit: m} y:
      {value: 2, unit: m} z: {value: 2,
      unit: m} phi: {value: 0, unit: rad}
      theta: {value: -10, unit: deg} psi:
      {value: 1, unit: deg}

  wake coefficient w: 0.9
  relative rotative efficiency eta: 1 thrust deduction
  factor t: 0.7
  rotation: anti-clockwise
  number of blades: 3
  blade area ratio AE/A0: 0.5 diameter:
  {value: 2, unit: m}
```

Commands are defined in the commands and controllers sections described below.

3.13 Models of maneuverability

3.13.1 Description

The purpose of this effort model is to be able to write a maneuverability model in a fairly generic way, without having to recompile the source code. Simple expressions of states and time can be calculated, for example:

```
- model: maneuvering
  name: test
  reference frame:
      frame: NED
      x: {value: 0.696, unit: m} y: {value: 0,
      unit: m} z: {value: 1.418, unit:
      m}
```

```
        phi: {value: 0, unit: deg} theta:
        {value: 0, unit: deg} psi: {value: 0, unit:
        deg}
commands: [command1, b, a]
X: 0.5*rho*Vs^2*L^2*X_ Y:
0.5*rho*Vs^2*L^2*Y_ Z: 0

K: 0
M: 0
N: 0.5*rho*Vs^2*L^3*N_ Vs:
sqrt(u(t)^2+v(t)^2)
L: 21.569
X_: Xu*u_ + Xuu*u_^2 + Xuuu*u_^3 + Xvv*v_^2 + Xrr*r_^2 + Xvr*abs(v_)*abs(r_)
Y_: Yv*v_ + Yvv*v_*abs(v_) + Yvvv*v_^3 + Yvrr*v_*r_^2 + Yr*r_ + Yrr*r_*abs(r_) + Yrrr*r_^
N_: Nv*v_ + Nvv*v_*abs(v_) + Nvvv*v_^3 + Nvrr*v_*r_^2 + Nr*r_ + Nrr*r_*abs(r_) + Nrrr*r_^  u_: u(t)/Vs v_: v(t)/Vs r_: r(t)/
Vs*L


Xu: 0
Huu: 0
Huuuu: 0
Xvv: -0.041
Xrr: -0.01
Xvr: -0.015
Yv: -0.13
Yvv: -0.18
Yvvv: 0
Yvrr: 0
The: 0.015
Drive: 0.021
Yrrr: 0
Yrvv: 0
Nv: -0.37
NV: -0.12
Nvvv: 0
Nvrr: 0
Nr: -0.1
Nrr: 0.005
Nrrr: 0
Nrvv: 0
```

- reference frame: Defines the transformation making it possible to pass from a known reference (whose name is given by frame) to the reference in which the forces are expressed. The torsor is automatically moved to the center of gravity (point (0,0,0) of the “body” reference).
- commands: optional. The maneuverability model can accept

external commands. He can also use the commands of any other model of effort (but for that it is necessary to well inform the complete name of the command, that is to say for example PropRudd (rpm)) •

X, Y, Z, K, M, N : coordinates of the force torque (in the reference body), expressed at the point of application defined above.

Here is a (very simplified) example of a maneuverability model that uses the commands from a propeller + rudder model which illustrates the syntax for using the commands from another model in a maneuverability model:

```
- name: SBPropRudd
  model: propeller+rudder position
  of propeller frame: frame: test

      x: {value: -53.155, unit: m} y: {value:
      3.750, unit: m} z: {value: 6.573, unit:
      m} phi: {value: 0, unit: rad} theta:
      {value: 4., unit: deg} psi: {value: 0,
      unit: deg}

  wake coefficient w: 0.066 relative
  rotative efficiency etaR: 0.99 thrust deduction factor t:
  0.18
  rotation: anti-clockwise
  number of blades: 4
  blade area ratio AE/A0: 0.809 diameter:
  {value: 4.65, unit: m} rudder area: {value: 10.8,
  unit: m^2} rudder height: {value: 4.171, unit: m}
  effective aspect ratio factor: 1.7 lift tuning coefficient: 1.
  drag tuning coefficient: 1. position of rudder in
  body frame: x: {value: -57.36, unit: m}
  y: {value: 4.40, unit: m} z: {value: 4.26,
  unit: m}



- model: maneuvering
  name: man
  reference frame: frame:
      fremm x: {value:
      0, unit: m} y: {value: 0, unit: m}
      z: {value: 2.22, unit: m} phi:
      {value: 0, unit: deg} theta: {value: 0,
      unit: deg} psi: {value: 0, unit: deg}


  X: 0.5*SBPropRudd(rpm)^2
```

Y: 0
Z: 0
K: 0
M: 0
N: 0

All values are assumed in international system units. The model requires specifying X, Y, Z, K, M and N, the other keys can be arbitrary.

Accessory variables (such as tau in the example above) can be used. The model automatically checks at runtime that it has all the necessary keys and infers the order of evaluation, in other words, an expression is only evaluated when all the expressions on which it depends have been evaluated (regardless of the order in which they were declared).

The expressions g, nu and rho can be used and their values are those given in the environmental constants section of the YAML file.

We can evaluate these delayed values of the states x,y,z,u,v,w,p,q,r by writing x(t-tau) (for example) where tau denotes an expression whose value is positive. t implicitly designates the current time. The maximum history length is calculated just after reading the input YAML file and before the simulation, depending on the effort models used, as follows:

- Each effort model knows the maximum history duration it needs, based on its input parameters • The maximum history duration is taken as the maximum of the history durations needed by each effort model effort • Most of the effort models (gravity, hydrostatic, resistance to progress. . .) do not need history and therefore have a necessary duration of 0 seconds

- the only force models requiring a history are the radiation dampings (the necessary length is given by the key tau max) and, possibly, the maneuverability forces, when using delayed variables such as x(t- 23). In this case, the maximum necessary history length is obtained by analyzing the formal expression: for example, M: 1E6*(x(t-6) + command1*x(t-5) + 2*b*y(t-4) + z(t-3)/a) will require at least 6 seconds of history for x, 4 for y, and 3 for z.

3.13.2 Grammar

More formally, the models must obey the following grammar (format “Extended Backus-Naur Form” ou EBNF) :

'+' | = term operator_and_term* expr =
mul_operators ' ' add_operators = '*' | '/'
add_operators operator_and_term =
term

operator_and_factor = mul_operators factor = factor
term operator_and_factor* = base ('^'
factor exponent)* = ('(' expr ')') | atom =
base base
exponent
atom = function_call | identifier | double = identifier '(' expr
function_call ')' = alpha (alphanum | '_')*
identifier

3.14 Wageningen B series propellers

3.14.1 Description

In 1937, the Dutch engineer L. Troost, then an employee of the Maritime Research Institute Netherlands (MARIN) based in Wageningen (Netherlands), created the Wageningen series B propellers. In order to establish a basis for propeller design, he published in 1938 and again in 1940 a series of systematic open water tests of 120 “series B” propellers, which are, to this day, the best known open water test series, although MARIN and other institutes of research carried out others subsequently .

In 1975, Oosterveld and Ossannen used statistical regression to establish the polynomial model of Wageningen helices presented here.

A tutorial presents the use of this model in the simulator.

3.14.2 Open Water Model Assumptions

We adopt the following notations:

- *'T'* is the thrust of the propeller in open water (in N), that is to say the norm of the forces generated by the propeller along its axis (without taking into account the suction of the hull),
- *'Q'* is the torque generated by the propeller in open water around its axis. He is expressed in Nm, •

'n' is the number of revolutions the propeller makes per second (in rpm), • *'D'* is the diameter of the propeller (in m), • *'rho'* is the volumetric density of the water (in kg/mˆ3), • *'Va'* is the speed of the undisturbed flow upstream of the propeller, for the configuration in open water (in m/s).

The open water model is subject to the following assumptions:

- the interactions between the propeller and the hull are not taken into account (disturb bation of the fluid upstream of the propeller), •

nor interactions between the propeller and the free surface, • we neglect the effects of the swell (in particular its orbital speed and the pressure fields it generates) and currents oceanic,

The interest of this model is that it is parametric and makes it possible to represent the performances of the propeller in a dimensionless form. The same model can thus be applied (within a scale coefficient) to homothetic propellers.
An additional limitation of the polynomial model in open water is that, unlike the four-quadrant model, it is only valid in forward motion (i.e. for positive or zero *'n').*

3.14.3 Model derivation in open water

The open water model is a model based on physical considerations, the coefficients of which can be obtained both experimentally and numerically by solving the Navier-Stokes equations. The postulate is that, given the above assumptions, propeller thrust can be expected to depend on:

• its diameter *'D' (in* m), • the speed *'Va'* of the fluid advance (in m/s), • the speed of rotation *'n'* of the propeller (in tr /s), • the density *'ÿ'* of the fluid (in kg/mˆ3), • the dynamic viscosity *'µ'* of the fluid (in kg/(ms)), • the static pressure of the fluid *'p0* ÿ e ' at the level of the propeller.

We would therefore have:

$$T_0 \propto \rho^a\cdot D^b\cdot V_a^c \cdot n^d \cdot \mu^f\cdot(p_0-e)^g$$

Performing the dimensional analysis to express 'a', *'b'* and *'d'* in terms of the other coefficients, we find:

$$T_0 \propto \rho^{1-fg}\cdot D^{4-c-2f-g}\cdot V_a^c \cdot n^{2-cf-2g} \cdot \mu^f\cdot(p$$

Or, by grouping the terms of the same power:

$$T_0 \propto \rho\cdot n^2\cdot D^4\cdot \left(\frac{V_a}{n\cdot D}\right)^c \cdot\left(\fra$$

We define the thrust coefficient:

$$K_T = \frac{T_0}{\rho\cdot n^2\cdot D^4}$$

The advance coefficient *'J'* is defined by:

$$J=\frac{V_a}{n\cdot D}$$

The Reynolds number *'Rn'* is expressed here:

$$R_n = \frac{\rho \cdot n\cdot D^2}{\mu}$$

and the cavitation number 'ÿ0' is:

$$\sigma_0=\frac{p_0-e}{\frac{1}{2}\rho\cdot n^2\cdot D^2}$$

so there is a function *'f'* such that

$$K_T = f(J,R_n,\sigma_0)$$

Similarly, for the torque 'Q', the torque coefficient *'KQ'* is defined by:

$$K_Q = \frac{Q_0}{\rho\cdot n^2\cdot D^5}$$

The open water model consists in explaining the functions *'KT* ' and *'KQ',* from which the thrust and the torque can then be derived.

3.14.4 Accounting for hull and wake effects

When the flow at the propeller has been disturbed by the hull, the fluid velocity at the propeller *'Va'* is not equal (in absolute value) to the speed of the ship relative to the water *'Vs',* in other words *'Va* ÿ= Vs'. The forward speed *'Va'* is, in general, very difficult to know and it is assumed to be proportional to the speed of the ship. We therefore define a coefficient *'w' (for* “wake”) such that:

$$w=1-\frac{V_a}{V_s}$$

'w' is constant in steady state, when the propeller operates under nominal conditions . Orders of magnitude of this coefficient are given for example in Carlton, pages 70, 72, 73 and 74.

In addition, the propeller reduces the pressure at the rear of the ship, which increases its resistance to progress.

To take these phenomena into account, we introduce the suction coefficient *'t'* such that:

$$t = 1 - \frac{R_v}{T_p}$$

where *'Rv'* is the towing resistance (in N) at *'VS' speed,* with no propeller, and *'Tp'* is the sum of propeller thrust (also in N) when the vessel is going at *'VS'* speed in using the propeller.

The actual thrust *'Tb'* is then defined by:

$$T_b = (1-t)\cdot T_0 = (1-t)\cdot \rho\cdot n^2\cdot D^4 \cdot K_T(J, R_n,\sigma_0)$$

and the actual torque is

$$Q_b = \this_R\cdot Q_0 = \this_R\cdot \rho\cdot n^2\cdot D^5 \cdot K_Q(J,R_n,\sigma_0);$$

where *'ÿR'* is called **adaptation efficiency**

$$J = \frac{V_a}{n\cdot D} = \frac{(1-w)\cdot V_s}{n\cdot D}$$

3.14.5 Expression of the coefficients *‘KT ’* and *‘KQ’*

In order to make the coefficients independent of the size of the propeller, we define the fraction of surface of the propeller *'AE/A0',* where *'AE'* designates the area of the blades (in mˆ2) and $\frac{p\cdot D^2}{4}$ *'A0'* = is l area of the disk circumscribed to the helix. The series are valid for 0.30 ÿ *AE/A0* ÿ 1.05.

We also define the pitch *'P'* of the propeller, a geometric parameter which translates the theoretical distance traveled by the propeller in one revolution. This distance varies depending on the reference line chosen. Wageningen's B-series uses the **front step,** but there are other conventions. The series are parameterized in *'P/D'* and it is assumed that *'0.5* ÿ *P/D* ÿ 1.4'.

We note '2 ÿ *Z* ÿ 7' the number of blades of the propeller.

The coefficients of the polynomials for *'KT* ' and *'KQ'* are denoted *'C* $_{i}^{T}$ *Q* and $_{re}$ respectively, where *'i'* is an integer such that '1 ÿ *i* ÿ 47'. *'s(i)', 's 'u(i)', 'u* $_{y}$ *'C i* (i)', *'t(i)', 't* ^{y} (i)', ^{y}(i)', *'v(i)'* and *'v* ^{y}(i)' are exponents between 0 and 6.

KT (J, P/D, AE/A0, Z, Rn = 2×106) = $\sum_{i=1}^{4} k\,{}^{7C}{}^{T}$ *·J s(i) ·(P/D) t(i) ·(AE/A0) u(i) · Z v(i)* '

KQ(J, *P/D, AE/A0, Z, Rn* = 2 × 106) = *(i) · Z v* $_{y}$ $_{y}$ *(i)* ' $\sum_{i=1}^{4} k\,{}^{7C}{}^{Q}$ *· J s* ^{y} *(i) · (P/D)* $^{y\,t}$ *(i) · (AE/A0) u*

The coefficients *'C k* ^{T} and *'C k* ^{Q} are defined for a Reynolds number *'Rn* = 2×106 ', but the model has been extended for Reynolds numbers between '2×106 ' and '2×109 ' by introducing terms *'ÿKT* ' and *'ÿKQ '* additional:

KT (J, P/D, AE/A0, Z, Rn) = KT (J, P/D, AE/A0, Z, 2×106)+ÿKT *(J, P/D, AE/A0, Z,* Rn)'

KQ(J, *P/D, AE/A0, Z, Rn)* = KQ(J, *P/D, AE/A0, Z,* 2×106)+*ÿKQ(J, P/D, AE/A0, Z,* Rn)'

3.14.6 Area of validity

The Wageningen Series B model should only be used when the following assumptions hold:

- The number of *'Z'* blades must be between 2 (inclusive) and 7 (inclusive).
- Ratio ' $\frac{\text{'}}{AE}$ ' must be in the range ' [0.3, 1.05] '. The simulation will not launch otherwise.
- The pitch of the propeller *'P'* must satisfy *'P/D* ÿ *[0.5,* 1.4]'
- The advance coefficient *'J'* is such that '0 ÿ *J* ÿ 1.5'

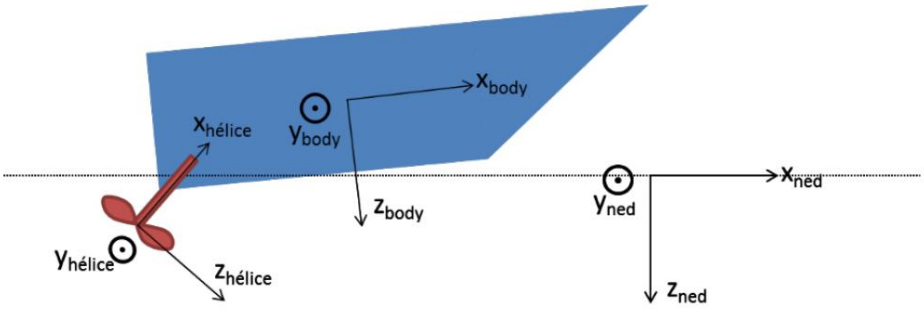
The conditions on *'Z'* and ' $\frac{BUT}{A0}$ ' concerning constant quantities during the simulation, they are verified before the launch and the simulation will not be carried out if these conditions are not verified.

If the advance coefficient *'J'* is outside the interval ' [0, 1.5] ', a warning message is displayed and *'J'* is saturated to be brought back into the interval: *'J* ÿ min(max(J, 0), 1.5)'

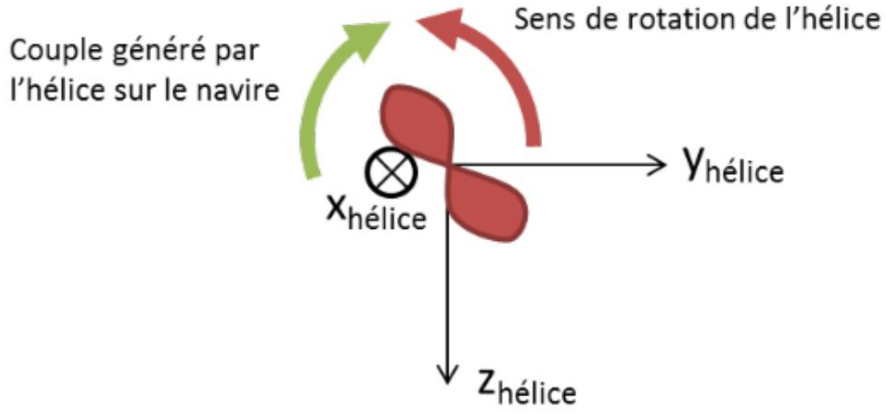
The condition on step *'P'* is checked during simulation and a warning message is displayed on the console. On the other hand, the pitch is not modified.

3.14.7 Expression of forces

The forces generated by the propeller are calculated in a specific frame entered in the position of propeller frame section of the YAML file. The thrust (ie the force generated by the propeller on the ship) is made in the direction of the positive *'x'* .



The direction of rotation of the propeller must also be specified because it determines the sign of the torque generated by the propeller on the vessel. We define this direction of rotation by standing behind the propeller, looking in the direction of the positive 'xpropeller' (therefore towards the front of the ship). In other words, the axis of rotation of the helix is not ' xhelix' but ÿ xhelix'. When the propeller rotates clockwise , it generates a counterclockwise torque, i.e. a **negative** sign torque when expressed in the propeller reference frame:



The torsor of the forces generated by the propeller and undergone by the ship (thus appearing in the right-hand side of the fundamental equation of the dynamics), expressed in the reference frame of the propeller, is therefore:

\tau_{\textrm{hélice}} = \left[\begin{array}{c}
X\\

$$
\begin{array}{l}
\text{AND}\\
\text{WITH}\\
K\\
M\\
N
\end{array}\right]_{\textrm{hélice}}
$$

$$
= \left[\begin{array}{c}
T_b\\
0\\
0\\
\kappa Q_b\\ 0\\
0
\end{array}\right]_{\textrm{hélice}}
$$

$$
= \left[\begin{array}{c}(1-t)\cdot
\rho\cdot n^2\cdot D^4 \cdot K_T(J, A_E/A_0, Z, P/D)\\ 0 \\ 0\\ \kappa \cdot \eta_R\cdot
\rho\cdot n^2\cdot D^5 \cdot K_Q(J, A_E/A_0, Z, P/D)\\ 0\\
0\end{array}\right]_{\textrm{hélice}}
$$

'ÿ' is -1 if the propeller rotates clockwise (standing behind the propeller and looking ahead of the ship) and +1 if it rotates counter-clockwise.

This torsor is then moved (change of point of application and change of coordinates) in the reference body in order to be summed with the others during the balance of the forces.

3.14.8 Settings

Here is an example of a possible configuration:

```
external forces:
  - name: port side propeller
    model: wageningen B-series position
    of propeller frame: frame:
        mesh(TestShip) x: {value:
        -4, unit: m} y: {value: -2, unit:
        m} z: {value: 2, unit: m} phi:
        {value: 0, unit: rad} theta:
        {value: -10, unit: deg} psi: {value: -1,
        unit: deg} wake coefficient w: 0.9
```

relative rotative efficiency eta: 1 thrust deduction
factor **t:** 0.7
rotation: clockwise
number of blades: 3
blade area ratio AE/A0: 0.5 diameter:
{value: **2,** unit: **m}**

• name : component name. User defined. Must correspond to that indicated in the expected command file. • model : model name. Must be wageningen B-series to use this model.

• position of propeller frame: definition of the propeller frame. • frame: frame in which x,y,z,phi,theta and psi are expressed. • x,y,z : projection of the position of the center of thrust of the propeller with respect to the center of the reference attached to the mesh and projected on the latter. • phi, theta, psi : definition of the rotation allowing to pass from the reference attached to the mesh to the reference attached to the helix, by following the chosen angle convention.

• wake coefficient : wake coefficient reflecting the disturbance of flow through the ship's hull. Between 0 and 1. • relative rotary efficiency: adaptation efficiency. • thrust deduction factor t: suction coefficient. • rotation definition of the direction of rotation to generate a positive thrust.
Used to calculate the sign of the moment generated by the propeller on the ship. Possible values are clockwise and anti-clockwise. If you choose clockwise, the propeller will turn clockwise (when standing at the back of the ship and looking towards the bow) and will generate a negative moment on the ship (in the propeller frame). See documentation. • number of blades: number of propeller blades. • blade area ratio AE/A0: surface fraction of the helix. • diameter: diameter of the propeller.

3.14.9 References

• *Marine Propellers and Propulsion,* 2007, John Carlton, Butterworth Heinermann, ISBN 978-07506-8150-6, page 89, 103
• *Seakeeping: Ship Behaviour in Rough Weather,* 1989, A. R. J. M. Lloyd, Ellis Horwood Series in Marine Technology, ISBN 0-7458-0230-3, page 404 • *KT, KQ and Efficiency Curves for the Wageningen B-series Propellers,* 1981 , Bernitsas , Ray , Kinley , University of Michigan
• *Offshore Hydromechanics,* 2001, JMJ Journée and WW Massie, Delft University of Technology, sections 4-40
• *Thrust Estimation and Control of Marine Propellers in Four-Quadrant Operations,* 2008, Luca Pivano, NTNU, ISBN 978-82-471-6258-3, pages 30 ,
• *The Wageningen Propeller Series,* 1992, Gert Kuiper, Marin publication

3.15 Simplified Heading Controller

3.15.1 Description

The purpose of this controller is to be able to perform wave simulations (for example to calculate RAO of movements by post-processing the temporal results of xdyn) by limiting heading variations. This controller directly generates a moment at the center of gravity of the body.

3.15.2 Expression of the forces

The moment generated is *Mz* = *Kÿ* · (ÿco ÿ ÿ) ÿ *Kr* · *r* = *ÿzzÿ*¨ where *'ÿzz'* designates the total moment of inertia (own inertia and added inertia) around the axis ' z'.

In the Laplace domain, the controller equation is written:

$$\sigma_{zz}p^2 + K_r p + K_{\psi} = 0$$

or, in canonical form:

$$p^2 + 2\zeta\omega_0 p + \omega_0^2 = 0$$

from where

'Kÿ = ÿzzÿ $^{2}_{0}$ a *'Kr = 2ÿÿ0ÿzz'*

These gains can be expressed as a function of the damping *'ÿ'* and the response time *'Tp'* given by *'Tp* = $\frac{2p}{oh0}$.

$$K_{\psi} = \sigma_{zz}\left(\frac{2\pi}{T_p}\right)^2$$

$$K_{r} = 2\zeta\sigma_{zz}\frac{2\pi}{T_p}$$

The heading 'ÿco' is given in the NED reference. If we assume that *'r* = 0', for *'ÿ* < ÿco', the generated moment must be positive, so *'Kÿ* · (ÿco ÿ ÿ) > 0'. Therefore, *'Kÿ* > 0'. Similarly, taking *'r* < 0' and *'ÿ* = ÿco', the generated moment must be positive to counter the velocity 'r', so ' ÿ *Kr* · *r* > 0', hence *'Kr* > 0'.

3.15.3 Settings

```
external forces:
  - name: controller
    model: simple heading controller ksi: 0.9

    Tp: {value: 4, unit: s}
```

• name: name of the controller (if several are used) • model: simple heading controller for this model • ksi: damping coefficient of the control law

• Tp: response time (second order system).

This model has only one command, the cap psi_co:

```
- name: controller
  t: [0,1,3,10]
  psi_co: {unit: deg, values: [25, 30, 40, 0]}
```

3.16 Simplified position controller

3.16.1 Description

The purpose of this controller is to be able to perform wave simulations (for example to calculate movement RAOs by post-processing xdyn's temporal results) by limiting heading and position variations. This controller directly generates moment and effort at the body's center of gravity.

3.16.2 Expression of forces

The force generated along the X axis is *'Fx = Kx* · (xco ÿ x) ÿ *Ku* · *u = ÿxxx¨'* where *'ÿxx'* designates the total moment of inertia (own inertia and added inertia) around the 'x' axis . The force generated along the Y axis is *'Fy = Ky* · (yco ÿ y) ÿ *Kv* · *v = ÿyyy¨'* where *'ÿyy'* designates the total moment of inertia (own inertia and added inertia) around the 'y' axis . The generated moment is *'Mz = Kÿ* · *(ÿco* ÿ ÿ) ÿ *Kr* · *r = ÿzzÿ¨'* where *'ÿzz'* denotes the total moment of inertia (own inertia and added inertia) around the 'z' axis.

In the Laplace domain, the controller equations are written:

$$\sigma_{xx}p^2 + K_u p + K_{x} = 0$$

$$\sigma_{yy}p^2 + K_u p + K_{y} = 0 \sigma_{zz}$$

$$p^2 + K_r p + K_{\psi} = 0$$

or, in canonical form:

$$p^2 + 2\zeta_x\omega_x p + \omega_x^2 = 0$$

$$p^2 + 2\zeta_y\omega_y p + \omega_y^2 = 0$$

$$p^2 + 2\zeta_{psi}\omega_{\psi} p + \omega_{\psi}^2 = 0$$

from where

'Kx = ÿxxÿ $_{x}^{2\cdot}$ and *'Ku = 2ÿxÿxÿxx'*

'This = ÿyyÿ $_{and}^{2\cdot}$ et *'Kv = 2ÿyÿyÿyy'*

'Kÿ = ÿzzÿ $_{p}^{2\cdot}$ a *'Kr = 2ÿÿÿÿzz'*

These gains can be expressed as a function of the damping *'ÿ'* and the response time *'T'* given by *'T =* $\frac{2p}{oh}$'.

$$K_{x} = \sigma_{xx}\left(\frac{2\pi}{T_x}\right)^2$$

$$K_{u} = 2\zeta\sigma_{xx}\frac{2\pi}{T_x}$$

$$K_{y} = \sigma_{yy}\left(\frac{2\pi}{T_y}\right)^2$$

$$K_{v} = 2\zeta\sigma_{yy}\frac{2\pi}{T_y}$$

$$K_{\psi} = \sigma_{zz}\left(\frac{2\pi}{T_{\psi}}\right)^2$$

$$K_{r} = 2\zeta\sigma_{zz}\frac{2\pi}{T_{\psi}}$$

The heading 'ÿco' is given in the NED reference. If we assume that *'r* = 0', for *'ÿ* < ÿco', the generated moment must be positive, so *'Kÿ* · (ÿco ÿ ÿ) > 0'. Therefore, *'Kÿ* > 0'. Similarly, taking *'r* < 0' and *'ÿ* = ÿco', the generated moment must be positive to counter the velocity 'r', so ' ÿ *Kr* · *r* > 0', hence *'Kr* > 0'.

3.16.3 Settings

```
external forces:
  - name: controller
    model: simple station-keeping controller ksi_x: 0.9 T_x:
    {value: 2, unit:
    s} ksi_y: 0.85 T_y: {value: 3, unit:
    s} ksi_psi: 0.8
    T_psi: {value: 4, unit: s}
```

• name: name of the controller (if several are used) • model: simple heading controller for this model • ksi_*: damping coefficient of the control law • T_*: response time (second order system) .

This model has three commands, heading psi_co, and position x_co, y_co (in the NED frame):

```
- name: controller
  t: [0,1,3,10] x_co:
  {unit: m, values: [25, 30, 40, 0]} y_co: {unit: m, values:
  [25, 30, 40, 0]} psi_co: {unit: deg, values: [25, 30, 40,
  0]}
```

3.17 Propeller and rudder

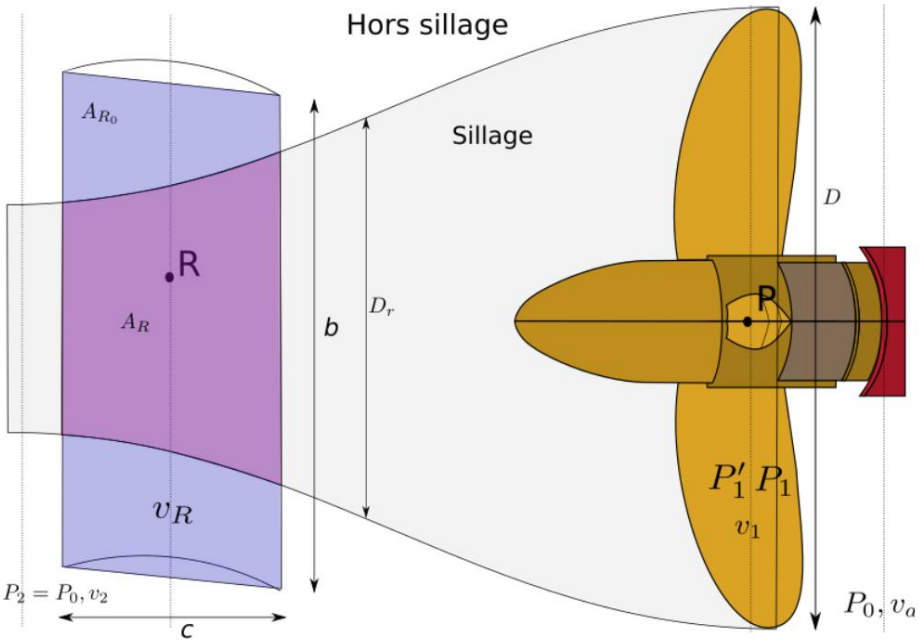
3.17.1 Description

This model describes the assembly consisting of a Wageningen propeller and a rudder. Both are used together as the rudder model uses the propeller slipstream

to calculate the forces due to the rudder (this model therefore assumes that the rudder is located downstream of the propeller)

3.17.2 Expression of forces

The following figure illustrates the modeled assembly:



The forces are calculated at point P (of the propeller) and then transported to the center of gravity. They are written:

F\textrm{tot}_P = F\textrm{safran}_P + F\textrm{helix}_P

The expression of the torsor 'FhelixP ' is given in the model “Wagenin propellers gen series B”.

The forces due to the rudder will be calculated at point R then the torsor will be moved to the center of gravity. In the following, we will simply denote 'Fsafran' the torsor at point R.

The chosen model separates the forces due to the rudder into two parts:

- The part coming from immersion in the wake of the propeller • The part simply due to the speed of the rudder in open water

F\textrm{safran} = F\textrm{safran}^{\textrm{sillage}} + F\textrm{safran}^{\textrm{hors sil

In the reference linked to the rudder, the latter only creates a resultant along the X and Y axes (in other words, Fz=0 and Mx=My=Mz=0).

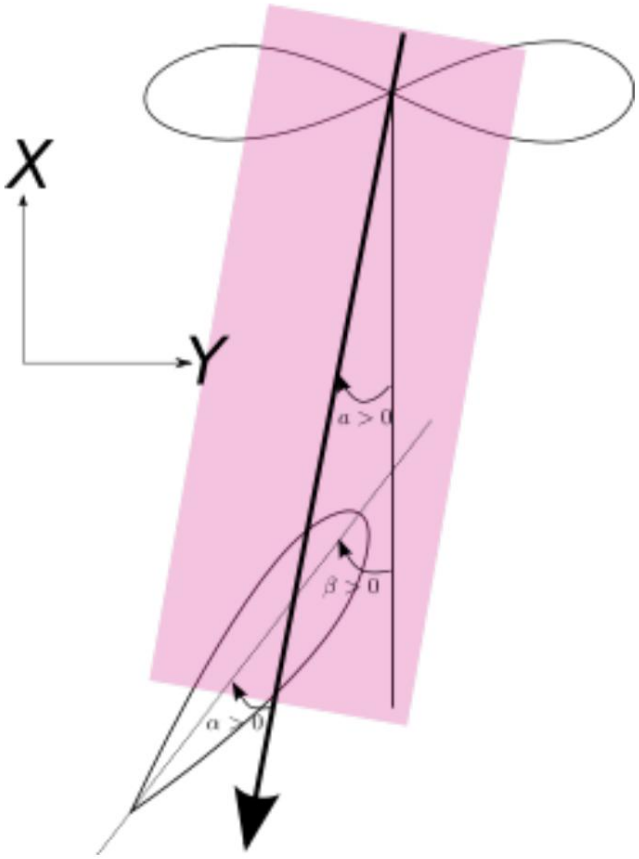
The components *'Fx'* and *'Fy'* of this resultant are expressed in the form:

F_x = -\textrm{Lift}(V_s, C_l, \alpha, S)\cdot\sin(\alpha) - \textrm{Drag}(V_s, C_d, \alpha

F_y = +\textrm{Lift}(V_s, C_l, \alpha, S)\cdot\cos(\alpha) - \textrm{Drag}(V_s, C_d, \alpha

The speed *'Vs'* and the area *'S'* are calculated differently depending on whether the part of the rudder in the wake of the propeller or that outside of this wake is considered.

The angle of incidence of the fluid in relation to the rudder is denoted *'ÿ'* and is defined in relation to the angle of incidence *'a'* of the fluid (in the NED reference) and the angle of the rudder *'ÿ' :*



\alpha = \beta - a(V_S) = \beta - atan2({V_S}_x, {V_S}_y)

\textrm{Lift}(V_S, C_l, \alpha, S) = \frac{1}{2}\rho S V_S^2 Cl(\alpha)\cos(\alpha) K_{\tex \textrm{Drag} (V_S, C_d, \alpha,

S) = \frac{1}{2}\rho S V_S^2 Cd(\alpha)\cos(\alpha) K_{\tex The coefficient ' cos(ÿ) ' allows to reduce rudder efficiency when *'ÿ'* becomes large.

In the following, we will detail the calculation of *'VS', 'Cl* ', *'Cd'* and *'S'* for the part

outside the wake and the inner wake part.

3.17.2.1 Calculation of *"VS"* The notations used appear on the diagram above . The thrust *'T'* generated by the propeller is equal to the change in momentum :

$$T = \rho\cdot A\cdot V_1\cdot(V_2 - V_a)$$

(Equation 3.31 *Marine Rudders & Control Surfaces* p. 49).

This thrust is also equal to the change in pressure multiplied by the area of the disc:

$$T = A\cdot(P_1'-P_1)$$

We write Bernoulli's equation upstream of the rudder, between 'P0' and 'P1', then downstream of the rudder, between *'P* ÿ' 1 and 'P2' :

$$P_0 + \frac{1}{2}\rho V_a^2 = P_1 + \frac{1}{2}\rho V_1^2$$

$$P_1' + \frac{1}{2}\rho V_1^2 = P_2 + \frac{1}{2}\rho V_2^2$$

from where

$$P_1' = P_2 + \frac{1}{2}\rho V_2^2 - \frac{1}{2}\rho V_1^2$$

$$P_1 = P_0 + \frac{1}{2}\rho V_a^2 - \frac{1}{2}\rho V_1^2$$

Then

$$P_1'-P_1 = P_2-P_0 + \frac{1}{2}\rho(V_2^2-V_a^2)$$

Pressures 'P2' and 'P0' correspond to undisturbed flows (very upstream and very downstream of the couple (rudder, propeller)). Therefore, these two pressures are equal:

$$T = A\cdot(P_1'-P_1) = \frac{1}{2}\rho A(V_2^2-V_a^2)$$

(equation 3.32 *Marine Rudders & Control Surfaces* p. 49)

We deduce the expression of 'V2' :

$$V_2 = \sqrt{V_a^2 + \frac{2T}{\rho A}}$$

The speed 'V1' at the level of the rudder can be deduced from the equality of two expressions of *'T' :*

- On the one hand, *'T* = ÿAV1(V2 ÿ Va)' 2
- on the other hand, *'T* = ½ ÿA(V 2 2 - V a) = ½ rA(V2 ÿ Va)(V2 + Va)' • But

We can deduce :

$$V_1 = \frac{1}{2}(V_a + V_2) = V_a + \frac{1}{2}(V_2-V_a)$$

The calculations of 'V0' and 'V2' being made in steady state, this expression of the speed 'V1' does not take into account the acceleration of the fluid between the propeller and

saffron. The effect of this acceleration is modeled by a factor *'KR'* called "contraction factor" (cf. *Marine Rudders & Control Surfaces* eq. 3.37 p.51). We thus obtain a speed *'uRS'* such that:

$$u_{RS} = V_a + K_R(V_2-V_a)$$

with

$$Kr = 0.5+\frac{0.5}{1+\frac{0.15}{\left|\frac{\Delta_x}{D}\right|}}$$

'ÿx' designates the distance between the helide and the rudder (along the 'x' axis) and *'D'* is the diameter of the propeller.

In order to factor this expression, we can express the speed 'V2' as a function of the speed *'Va' :*

$$V_2 = \sqrt{V_a^2 + \frac{2T}{\rho A}}$$

or another expression of *'T'* can be given from the Wageningen model

$$T = \rho n^2 D^4 K_T$$

$$\frac{2T}{\rho A} = \frac{8}{\pi} n^2 D^2 K_T$$

but the advance parameter *'J'* is written:

$$J = \frac{V_a}{n D}$$

SO

$$\frac{2T}{\rho A} = \frac{8}{\pi} \frac{K_T}{J^2} V_a^2$$

from where

$$u_{RS} = V_a\left(1 + K_R \left(\sqrt{1 + \frac{8 K_T}{\pi J^2}} - 1\right)\right)$$

On pose

$$C_{Th} = \frac{8}{\pi} \frac{K_T}{J^2}$$

$$u_{RS} = V_a\left(1 + K_R \left(\sqrt{1 + C_{Th}} - 1\right)\right)$$

This *'uRS'* speed was calculated by making the following assumptions:

- No friction • The propeller has an infinite number of blades • No modification of the radial speed by the propeller

In practice, differences can reach 30% between *'uRS'* and the measurements taken during tests. This is why the speed *'uRS'* is multiplied by a factor *'RF'* called the "reduction factor" (cf. eq 11.1 p. 371 *Marine Rudders & Control Surfaces) :*

$$RF = 1-0.135\sqrt{C_{Th}}$$

The speed in the wake of the propeller is therefore expressed (in the reference “body”):

$$V_S = \left[\begin{array}{c}RF\cdot V_a\left(1 + K_R \left(\sqrt{1 + C_{Th}} - 1\right)\right)\\ v\\0\end{array}\right]$$

The speed out of the wake is simply:

$$V_S = \left[\begin{array}{c}V_a\\ v\\0\end{array}\right]$$

where *'Va* = (1 ÿ w) u', *'w'* being the wake coefficient.

3.17.2.2 Calculation of *'Cl* ' We introduce the aspect ratio 'ÿ' (cf. *Manoeu vring Technical Manual* p. 76)

$$\Lambda = K_{\Lambda}\frac{b^2}{A_R}$$

where 'Kÿ' is a parameter entered by the user.

We use the Soeding formula (see *Manoeuvring Technical Manual,* eq. 1.2.8 p.77 and eq. 1.2.48 p.97):

$$Cl(\alpha) = 2\pi\frac{\Lambda(\Lambda+1)}{(\Lambda+2)^2}\sin(\alpha)$$

3.17.2.3 Calculation of *'Cd'* We use the following formula (cf. *Maneuvering Technical Manual,* p. 78 eq. 1.2.9)

$$C_d = 1.1 \frac{Cl^2}{\pi \Lambda} + Cd_0$$

The coefficient of resistance *'Cd0'* is worth:

$$Cd_0 = 2.5 C_f$$

(cf. *Maneuvering Technical Manual,* p. 78 (§ *for Cd0))*

'Cf ' is an ITTC coefficient found for example in *Marine rudders and Control Surfaces,* p.31 eq. 3.18:

$$Cf = \frac{0.075}{\left(\frac{\log(R_n)}{log(10.0)}-2\right)^2}$$

The Reynolds number *'Rn'* of the rudder is given by (cf. *Maneuvering Technical Manual,* p. 78 eq. 1.2.12):

$$Rn = Vs \frac{c}{\nu}$$

where the chord *'c'* of the rudder is:

$$c = \frac{A_R}{b}$$

3.17.2.4 Calculation of *'S'* The area *'AR'* of the rudder is separated into two parts: a part inside the wake and a part outside. The part inside the wake is obtained by considering the diameter of the wake *'Dw'* and the part outside by making the difference with *'AR'.*

$$S_{\textrm{sillage}} = \min(A_R, c\cdot D_w)$$

$$S_{\textrm{hors sillage}} = A_R - S_{\textrm{sillage}}$$

where *'c'* is the chord calculated above.

The wake diameter *'Dw'* is defined by:

$$\frac{D_w}{D_{\textrm{helix}}} = \sqrt{\frac{V_1}{u_{RS}}}$$

$$V_1 = V_a\left(1 + 0.5 \left(\sqrt{1 + C_{Th}} - 1\right)\right)$$

$$u_{RS} = V_a\left(1 + K_R \left(\sqrt{1 + C_{Th}} - 1\right)\right)$$

from where

$$\frac{D_w}{D_{\textrm{hélice}}} = \sqrt{\frac{1+0.5(\sqrt{1+C_{Th}}-1)}{1+K_R(\sqrt{1+C_{Th}}-1)}}$$

3.17.3 Settings

```
external forces:
  - name: Prop. & rudder
    model: propeller+rudder position
    of propeller frame: frame:
        mesh(TestShip) x: {value:
        -4, unit: m} y: {value: -2, unit:
        m} z: {value: 2, unit: m} phi:
        {value: 0, unit: rad} theta:
        {value: -10, unit: deg} psi: {value: -1,
        unit: deg} wake coefficient w: 0.9 relative
        rotative efficiency etaR: 1 thrust
    deduction factor t: 0.7


    rotation: clockwise
    number of blades: 3
    blade area ratio AE/A0: 0.5 diameter:
    {value: 2, unit: m} rudder area: {value:
    2.2, unit: m^2} rudder height: {value: 2, unit: m^2}
    effective aspect ratio: 1.7 lift tuning coefficient: 2.1
    drag tuning coefficient: 1 position of
    rudder in body frame: x: {value: -5.1,
    unit: m} y: {value: -2, unit: m} z:
    {value: 2, unit: m}
```

We find the parameters of the model 'Wageningen' which are not described again here (except model). We have the following additional parameters:

• model: propeller+rudder for this model, • rudder
area: *'AR'*, • rudder
height: 'b', • effective
aspect ratio: Parameter 'Kÿ' in the calculation of the aspect ratio (for the formula of Soeding) ci above, • lift tuning coefficient: 'Klift' in
the formulas above, • drag tuning coefficient: 'Kdrag' in the formulas
above, • position of rudder in body frame: (see diagram above), projected
into the “body” marker. coordinates of point *'P'*

This model has three commands:

• the rotational speed of the propeller, always positive for this model, defined per rpm,
• the “pitch to diameter” ratio, defined by P/D, •
the rudder angle, defined by beta.

Here is an example of a corresponding command section:

```
- name: controller t:
  [1,3,10] rpm:
  {unit: rpm, values: [3000, 3000, 4000]}
  P/D: [0.7,0.7,0.8] beta:
  {unit: deg, values: [10,-15,20]}
```

3.17.4 Outputs

To obtain the effort outputs of this model, one writes for example:

```
output: -
    format: csv
        filename: propRudd.csv data:
        [t, 'Fx(Prop. & rudder,TestShip,TestShip)', 'Fy(Prop. & rudder,TestShip,NED)']
```

When you ask for an effort, you specify:

• The component we want (Fx and My here) • The
name of the force model (“Prop. & rudder” in our example) • The name of
the body (“TestShip”) • The
expression mark (“TestShip” and “NED”). This marker can only be the “body” marker (“TestShip” here), the “NED” marker or the marker linked to the force model.

In the previous example, we obtain the projection along the *'X'* axis of the TestShip reference mark of the Prop force. & rudder acting on the ship and corresponding to the name of the actuator entered in the name key in order to be able to define several actuators of the same type, as well as the projection on the Y axis of the NED reference frame of the moment around the Y axis of the ship.

3.17.5 References

- *Marine Rudders & Control Surfaces, Principles, Data, Design & Applica tions,* Anthony F. Molland & Stephen R. Turnock, published by Elsevier Ltd., 2007, ISBN: 978-0-75-066944-3
- *Maneuvoeuvring Technical Manual,* Seehafen Verlag, 1993, ISBN 3-87743-902-0

3.18 Model Kt(J) & Kq(J)

3.18.1 Description

The purpose of this model is to specify propeller force curves *'Kt'* and *'Kq'* according to the advance coefficient *'J'* only. Apart from the calculation of *'Kt'* and *'Kq',* this model is identical to the Wageningen series B propeller model described above.
The torsor of the forces generated by the propeller and undergone by the ship (thus appearing in the right-hand side of the fundamental equation of the dynamics), expressed in the reference frame of the propeller, is therefore:

$$\tau_{\textrm{hélice}} = \left[\begin{array}{c} X\\ AND\\ WITH\\ K\\ M\\ N \end{array}\right]_{\textrm{hélice}} = \left[\begin{array}{c} T_b\\ 0\\ 0\\ \kappa Q_b\\ 0\\ 0 \end{array}\right]_{\textrm{helix}} = \left[\begin{array}{c}(1-t)\cdot \rho\cdot n^2\cdot D^4 \cdot K_T (J)\\ 0\\ 0\\ \kappa \cdot \eta_R\cdot \rho\cdot n^2\cdot D^5 \cdot K_Q(J)\\ 0\\ 0\end{array}\right]_{\textrm{helix}}$$

3.18.2 Settings

Here is an example of a possible configuration:

```
external forces:
  - name: port side propeller model:
    Kt(J) & Kq(J) position of
    propeller frame: frame: mesh(TestShip)
          x: {value: -4, unit: m} y:
          {value: -2, unit: m} z: {value: 2,
          unit: m} phi: {value: 0, unit: rad}
          theta: {value: -10, unit: deg}
          psi: {value: -1, unit: deg} wake
          coefficient w: 0.9


    relative rotative efficiency etaR: 1 thrust deduction
    factor t: 0.7
    rotation: clockwise
    diameter: {value: 2, unit: m}
    J: [-1.00000E+00,-8.00000E-01,-5.00000E-01,-2.50000E-01,-1.00000E-03,1.00000E-03, 2.000  Kt:
    [-4.50000E-01,-2.50000E-01,-1.90000E-01,-2.00000E-01,-2.00000E-01,3.25000E-01, 2.80  Kq:
    [-4.80000E-02,-3.30000E-02,-2.20000E-02,-2.50000E-02,-2.80000E-02,3.40000E-02, 3.26
```

• name : Component name. User defined. Must correspond to that indicated in the expected command file, • model : Name of the model. Must be wageningen B-series to use this model,
• position of propeller frame: Definition of the propeller frame, • frame: frame in which x,y,z,phi,theta and psi are expressed. • x,y,z : projection of the position of the center of thrust of the propeller with respect to the center of the reference attached to the mesh and projected on the latter, • phi,theta,psi : Definition of the rotation allowing to pass from the reference attached to the mesh to the reference attached to the helix, following the chosen angle convention,

• wake coefficient : wake coefficient reflecting the disturbance of flow through the ship's hull. Between 0 and 1, • relative rotary efficiency etaR: adaptation efficiency, • thrust deduction factor t: suction coefficient, • rotation definition of the direction of rotation to generate a positive thrust.
Used to calculate the sign of the moment generated by the propeller on the ship.
Possible values are clockwise and anti-clockwise. If you choose clockwise, the propeller will turn clockwise (when standing at the back of the ship and looking towards the bow) and will generate a negative moment on the ship (in the propeller frame).
See documentation,
• diameter: propeller diameter (in m), • J: advance coefficient. Corresponds to the Kt and Kq lists, • Kt: thrust coefficient as a function of J, • Kq: moment coefficient as a function of J.

3.18.3 Outputs

To obtain the effort outputs of this model, one writes for example:

```
output: -
    format: csv
        filename: prop.csv data:
        [t, 'Fx(port side propeller,TestShip,TestShip)', 'Fx(port side propeller,TestShi
```

In the previous example, we obtain the projection along the *'X'* axis of the TestShip marker of the port side propeller force (corresponding to the name of the actuator entered in the name key in order to be able to define several actuators of the same type) as well than the projection of this same force along the *'X'* axis of the NED reference.

3.19 Remote Effort Model

If you want to use an effort model that is not implemented in xdyn, you can implement it separately and call it from xdyn, without intervening on the xdyn source code.

3.19.1 Description

As for the remote wave models, we use “gRPC” technology : the force model will then be encapsulated in a service and called by xdyn using parameters specified in the xdyn YAML configuration file.

3.19.2 Settings

In the external forces section, we add a section of the following form:

```
- model: grpc
  name: some name
  url: http://localhost:50001
```

The model: grpc parameter tells xdyn that this is a remote effort model and url gives the address where the gRPC server can be reached.

These models have access to all commands defined in the simulation. The point of application for gRPC effort models is defined by the effort model itself, and in the same way as for the maneuverability model.

Thus, the responses returned to xdyn by the effort model are defined by:

```
message ForceResponse {

    double Fx = 1;                    // Projection of the force acting on "BODY"
    double Fy = 2;                    // Projection of the force acting on "BODY"
    double Fz = 3;                    // Projection of the force acting on "BODY"
    double Mx = 4;                    // Projection of the torque acting on "BODY
```

```
    5;  // Projection of                                    // Projection of the torque acting on "BODY  double My =
    wish to serialize.   the torque acting on "BODY  double Mz = 6;  map<string, double> extra_observations = 7; // Anything we
    Specific
}
```

3.19.3 Access to potential code results

It is possible to give access to the results of the potential codes (HDB or PRECAL_R) to a remote effort model. To do this, you must put an hdb (or raodb) key in the section defining this model. For example :

```
external forces:
  - model: grpc
    name: hdb force model
    url: force-model:9002
    hdb: test_ship.hdb
```

To use a PRECAL_R file, we would write:

```
external forces:
  - model: grpc
    name: precal force model url:
    force-model:9002
    raodb: test_ship. ini
```

The HDB or PRECAL_R file is then read by xdyn and its content is sent to the effort model (return value of the set_parameters method). If we use the Python framework from the interfaces repository, these results are stored in the third argument of the constructor:

```
class HDBForceModel(force.Model): """Outputs
      data from HDB in extra_observations."""

      def __init__(self, _, body_name, results_from_potential_theory): """Initialize parameters
            from gRPC's set_parameters."""" self.body_name = body_name
            self.results_from_potential_theory =
            results_from_potential_theory
```

The values read from the HDB and PRECAL_R files are converted into units of the International System, without multiples (for example Newtons and not kilo-Newtons).

Remote effort models can access the following fields:

Champs	Description	Clef HDB	Clef PRECAL_R	Unit
And	Infinite Frequency Added Mass Matrix	[Added_mass_Radiation_Damping]/added_mass_damping_matrix_inf_freq_total_i [ADDED_MASS/TOTAL_i] (unique for ment the first line, minimum period)		kg for '1 ÿ i, j ÿ 3', kg.m2 For '4 ÿ i, j ÿ 6', kg.m sinon
forces_module_tables	Modulus of diffraction, diffraction by pulsation omega and by incidence psi (M[omega][psi])	[DIFFRACTION_FORCES_AND_MOMENTS]/[INCIDENCE_EFM_MOD_i]	Column of left of signals F_dif_mi where i designates the degree of freedom, from 1 to 6	N/m for i < 4 and Nm/m for i > 3
efforts_phase_tables	Phase of diffraction, diffraction by omega pulsation and by psi incidence (M[omega][psi])	[DIFFRACTION_FORCES_AND_MOMENTS]/[INCIDENCE_EFM_PH_i]	radcolumn signal line F_dif_mi where i designates the degree of freedom, from 1 to 6	
diffraction_module_periods	Periods at which are defined the modules of the diffraction efforts	First column of [DIFFRACTION_FORCES_AND_MOMENTS]/[INCIDENCE_EFM_MOD_i]	Deducted from Dimensions/WAVE FREQUENCIES/waveFreq	s
diffraction_phase_periods	Periods at which define the phases of the diffraction efforts	First from column de [DIFFRACTION_FORCES_AND_MOMENTS]/[INCIDENCE_EFM_PH_i]	Deducted Dimensions/WAVE FREQUENCIES/waveFreq	s
diffraction_module_psis	Incidences at which the modules of the diffraction forces are defined	[DIFFRACTION_FORCES_AND_MOMENTS]/[INCIDENCE_EFM_MOD_i]	Dimensions/WAVE DIRECTIONS/waveDir	rad
diffraction_phase_psis	Incidences at which the phases of the diffraction efforts are defined	[DIFFRACTION_FORCES_AND_MOMENTS]/[INCIDENCE_EFM_PH_i]	Dimensions/WAVE DIRECTIONS/waveDir	rad

Champs	Description	Clef HDB	Clef PRECAL_R	Unit
effort module	Froude_krylov_module_tables Froude-Krylov, by omega pulsation and by psi incidence (M[omega][psi])	[FROUDE-KRYLOV_FORCES_AND_MOMENTS]/[INCIDENCE_FKFM_MOD_	COM_mFOR for signals i F_inc_mi where i < 4 and Nm/m designates the degree of from 1 to 6 Column of rad [FROUDE-	N/m left of freedom, for i > 3
froude_krylov_phase_tables	Phase of efforts Froude-Krylov, , by omega pulsation and by psi incidence (M[omega][psi])	KRYLOV_FORCES_AND_MOMENTS]/[INCIDENCE_FKFM_PH_i right of	signals F_inc_mi where i designates the degree of freedom, from 1 to 6	
froude_krylov_module_periods	Periods at which the modules of Froude-Krylov efforts are defined	First column of each row of [FROUDE-KRYLOV_FORCES_AND_MOMENTS]/[INCIDENCE_FKFM_MOD_	Deducted from Dimensions/WAVE FREQUENCIES/waveFreq	s
froude_krylov_phase_periods	Periods at which the phases of Froude-Krylov efforts are defined	First from column each line of [FROUDE-KRYLOV_FORCES_AND_MOMENTS]/[INCIDENCE_FKFM_PH_i rad	Deducted Dimensions/WAVE de FREQUENCIES/waveFreq	s
froude_krylov_module_psis	Incidences at which the modules of Froude-Krylov efforts are defined	[FROUDE-KRYLOV_FORCES_AND_MOMENTS]/[INCIDENCE_FKFM_MOD_Dimensions/WAVE DIRECTIONS/waveDir		
of the Froude-Krylov efforts are defined angular_frequencies Froude_krylov_phase_psis	Incidences at which the phases	[FROUDE-KRYLOV_FORCES_AND_MOMENTS]/[INCIDENCE_FKFM_PH_iDimensions/WAVE DIRECTIONS/waveDir		rad
Pulsations at which the coefficients of the added mass matrix and the radiation damping coefficients are defined		First column of [Added_mass_Radiation_Damping]/[ADDED_MASS_LINE_i]	Dimensions/WAVE FREQUENCIES/waveFreq	rad/s
forward_speed	Speed at which the calculations were performed	[FORWARD_SPEED]shipSpeed		m/s

Champs	Description	Clef HDB	Clef PRECAL_R	Unit
added_mass_coeff	coefficients of added masses, per pulsation matrix	[Added_mass_Radiation_Damping]/[ADDED_MASS_LINE_i]	Radiation_ A_mimj where i and for j are the '1 ÿ numbers of *i, j* ÿ line and of 3', column (from 1 to 6)	kg.m2 For '4 ÿ *i, j* ÿ 6', kg.m sinon
radiation_damping_coeff	Radiation damping matrix coefficients, per pulse	[Added_mass_Radiation_Damping]/[DAMPING_TERM]	Radiation_ B_mimj where i and j are the row and column numbers (from 1 to 6)	kg/s pour '1 ÿ *i, j* ÿ 3', kg.m2/s pour '4 ÿ *i, j* ÿ 6', kg.m/s sinon
wave_drift_forces	Drift forces waves, by omega pulsation and by psi incidence (M[omega][psi])	[DRIFT_FORCES_AND_MOMENTS]/[INCIDENCE_DFM_001]	Sigma_ F_drift_mi where i designates the axis (1 for X, 6 for N)	kg/m/s2 pour '1 ÿ *i* ÿ 3', kg/s2 For '4 ÿ *i* ÿ 6' s
wave_drift_periods	Periods at which wave drift forces are defined	First column of [DRIFT_FORCES_AND_MOMENTS]/[INCIDENCE_DFM_001]	Deducted from Dimensions/WAVE FREQUENCIES/waveFreq	

Champs Description	Clef HDB	Clef PRECAL_R Unit
wave_drift_psi_rad	Incidences at which defined the drift forces on swell	Value at the end of each line	Dimensions/WAVE rad DIRECTIONS/waveDir [DRIFT_FORCES_AND_MOMENTS]/[INCIDENCE_DFM_001] 0.0000

In the Python API, these fields are represented by Numpy arrays in order to facilitate their numerical processing using the functions of the Numpy library.

Warning : if the HDB and PRECAL_R files start their numbering at 1, the same does not apply to the Python code: an index i of 1 in the previous table therefore corresponds to an index of 0 in the Numpy tables of Python APIs.

The matrices of the PRECAL_R and HDB files are supposed to be expressed in a “Z up” frame: they are therefore systematically converted into the “BODY” frame of xdyn (Z down).

If *‘Mh = ((mij))1ÿi,jÿ6’* designates a matrix of a PRECAL_R or HDB file and *‘Mx’* is the representation of this matrix in the BODY reference of xdyn, xdyn performs the following calculation on the results of the potential codes before providing them to the effort models:

$$M_x=R^T\cdot M_h\cdot R = \left[\begin{array}{rrrrrr} m_{11} & -m_{12} & -m_{13} & m_{14} & -m_{15} & -m_{16}\\ -m_{21} & m_{22} & m_{23} & -m_{24} & m_{25} & m_{26}\\ -m_{31} & m_{32} & m_{33} & -m_{34} & m_{35} & m_{36}\\ m_{41} & -m_{42} & -m_{43} & m_{44} & -m_{45} & -m_{46}\\ -m_{51} & m_{52} & m_{53} & -m_{54} & m_{55} & m_{56}\\ -m_{61} & m_{62} & m_{63} & -m_{64} & m_{65} & m_{66} \end{array}\right]$$ où *‘R‘* est la matrice de passage du repère PRECAL_R/AQUA+ vers le repère xdyn et *‘RT ‘* est sa transposée. *‘R‘* a pour expression :

$$R=\left[\begin{array}{rrrrrr} 1 & 0 & 0 & 0 & 0 & 0\\ 0 & -1 & 0 & 0 & 0 & 0\\ 0 & 0 & -1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & -1 & 0\\ \end{array}\right]$$

0 & 0 & 0 & 0 & 0 & -1
\end{array}\right]

3.19.4 Access to filtered states

xdyn calculates filtered states whose definition is made in the (optional) filtered states section of the input YAML file. These filtered values are provided to the remote effort models. If we use the Python framework from the interfaces repository, these filtered values are stored in the sixth argument of the grpcforce.Model.force(self, t, states, commands, wave_information, filtered_states) method.

3.19.5 Example of use

- Tutorial 10 details the use of a remote model. • Tutorial 13 shows how to use data from a calculation potential from a distant effort model.
- Tutorial 14 gives an example of using filtered states.

4 Tutorials

4.1 Tutorial 1: free fall

python echo=False, results='raw', name='tutorial_01_load_yaml' yaml_data = load_yaml('tutorial_01_falling_ball.yml')

We start by defining the rotation conventions:

python echo=False, results='raw', name='tutorial_01_print_rotation_conventions' print_yaml(yaml_data, 'rotations convention')

Then we give environmental constants:

python echo=False, results='raw', name='tutorial_01_print_environmental_constants' print_yaml(yaml_data, 'environmental constants')

No environmental model (swell, wind, etc.) is required for this simulation:

python echo=False, results='raw', name='tutorial_01_print_environment_models' print_yaml(yaml_data, 'environment models')

We define the position of the reference “body” compared to the mesh:

python echo=False, results='raw', name='tutorial_01_print_position_of_body_frame' print_yaml(yaml_data, 'bodies/0/position of body frame relative to mesh')

The initial conditions are described as follows:

python echo=False, results='raw', name='tutorial_01_print_initial_conditions' print_yaml(yaml_data, 'bodies/0/initial position of body frame relative to NED') print_yaml(yaml_data, 'bodies/0/initial velocity of body frame relative to NED')

Dynamic data includes mass, inertia matrix, added inertias, and center of inertia position:

python echo=False, results='raw', name='tutorial_01_print_dynamics_section' print_yaml(yaml_data, 'bodies/0/dynamics')

Only gravity acts on the solid:

python echo=False, results='raw', name='tutorial_01_print_external_forces_section' print_yaml(yaml_data, 'bodies/0/external forces')

In the end, we get the following file:

python echo=False, results='raw', name='tutorial_01_print_full_yaml' print_yaml_file('tutorial_01_falling_ball.yml')

4.1.1 Launching the simulation

The simulation can run as follows:

python echo=False, results='raw', name='tutorial_01_launch_simulation_csv_output' execCommand('xdyn tutorial_01_falling_ball.yml --dt 0.01 --tend 1 -o out.csv')

To have output on the console, you can do:

python echo=False, results='raw', name='tutorial_01_launch_simulation_console_output' execCommand('xdyn tutorial_01_falling_ball.yml --dt 1 --tend 5 -o tsv')

tsv stands for “tab-separated values” here.

You can also change the initial time (it being understood that the initial conditions defined in the YAML file apply to this initial time, whatever it is, and not to t = 0):

python echo=False, results='raw', name='tutorial_01_sun_with_modified_initial_conditions' execCommand('xdyn tutorial_01_falling_ball.yml --dt 0.01 --tstart 2 --tend 3 -o out.csv')

We can choose the solver:

python echo=False, results='raw', name='tutorial_01_change_solver' execCommand('xdyn tutorial_01_falling_ball.yml --dt 0.01 --tend 1 -s rk4 -o out.csv')

The list of all options is available by running:

python echo=False, results='raw', name='tutorial_01_xdyn_help'
execCommand('xdyn -h', echo_output=True)

4.1.2 Results

Here is a plot of elevation over time:

python echo=False, results='raw', name='tutorial_01_plot_results' data =
csv('out.csv') plot = prepare_plot_data(data, x='t', y='z(ball)', name='
Result') g = cartesian_graph([plot], x='t (s)', y='Elevation (m)') create_layout(g,
title='Elevation over time') ## Tutorial 2: oscillations in immersion

This tutorial aims to illustrate the use of hydrostatic models and to briefly compare non-linear hydrostatic (exact) and non-linear hydrostatic (fast) models .

python echo=False, results='raw', name='tutorial_02_load_yaml'
yaml_data_exact_hs = load_yaml('tutorial_02_exact_hydrostatic.yml')
yaml_data_fast_hs = load_yaml('tutorial_02_fast_hydrostatic.yml')

4.1.3 Description of the problem

In this example, we consider a ship subjected only to gravity and hydrostatic forces, without damping. The ship is released without initial speed above the free surface (assumed flat) and will therefore perform undamped oscillations in immersion.

4.1.4 Writing the simulator configuration file

We document here only the changes compared to tutorial 1.

The environment is defined as follows:

python echo=False, results='raw', name='tutorial_02_print_environment_models'
print_yaml(yaml_data_exact_hs, 'environment models')

As described in the documentation of the input file, this means that the free surface is perfectly flat and horizontal, at the height *'z* = 0' in the NED coordinate system.

Compared to tutorial 1, the position of the “body” marker relative to the mesh is important here since an STL file is provided:

python echo=False, results='raw', name='tutorial_02_print_position_of_body'
print_yaml(yaml_data_exact_hs, 'bodies/0/position of body frame relative to mesh')

We describe in the initial conditions the fact that the boat is released at 5 m above the water level (the z axis of the NED frame being oriented downwards, negative values correspond to points above the free surface):

python echo=False, results='raw', name='tutorial_02_print_initial_conditions' print_yaml(yaml_data_exact_hs, 'bodies/0/initial position of body frame relative to NED') print_yaml(yaml_data_exact_hs, 'bodies/0/initial velocity of body frame relative to NED')

Dynamic data includes mass, inertia matrix, added inertias, and center of inertia position:

python echo=False, results='raw', name='tutorial_02_print_dynamics_section' print_yaml(yaml_data_exact_hs, 'bodies/0/dynamics')

We first use the approximate hydrostatic model whose documentation is described here:

python echo=False, results='raw', name='tutorial_02_print_external_forces' print_yaml(yaml_data_fast_hs, 'bodies/0/external forces')

In the end, we get the following file:

python echo=False, results='raw', name='tutorial_02_print_full_yaml' print_yaml(yaml_data_fast_hs)

4.1.5 Launching the simulation

The simulation can now be started as follows:

python echo=False, results='raw', name='tutorial_02_run_simulation' execCommand('xdyn tutorial_02_fast_hydrostatic.yml --dt 0.1 --tend 10 -o fast.csv') execCommand('xdyn tutorial_02_exact_hydrostatic.yml --dt 0.1 --tend 10 -o exact.csv')

4.1.6 Results

Here are the results :

python echo=False, results='raw', name='tutorial_02_plot_elevations' fast_data = csv('fast.csv') exact_data = csv('exact.csv') fast_plot = prepare_plot_data(fast_data, x='t', y='z(TestShip)', name='Modèle hydrostatique rapide') exact_plot = prepare_plot_data(exact_data, x='t', y='z(TestShip)', name='Modèle hydrostatique exact') g = cartesian_graph([fast_plot, exact_plot], x='t (s)', y='Élévation (m)') create_layout(graph=g, title='Élévation au cours du temps')

We can also represent displacements along the *'y'* axis :

python echo=False, results='raw', name='tutorial_02_plot_y' fast_plot = prepare_plot_data(fast_data, x='t', y='y(TestShip)', name='Modèle hydrostatique rapide') exact_plot = prepare_plot_data(exact_data, x='t', y='y(TestShip)', name='Modèle hydrostatique exact') g = cartesian_graph([fast_plot, exact_plot], x='t (s)', y='y (m)')

create_layout(graph=g, title='Yaw over time') ##
Tutorial 3: wave generation on a mesh

The simulator is intended to represent the behavior of solids in a fluid environment, but it can also be used to simulate an environment, without any solid. This can be interesting, for example, to generate wave fields in order to test wave motion prediction algorithms. This tutorial explains how to use the simulator for this type of simulation.

4.1.7 Description of the problem

In this example, we will simulate an Airy swell consisting of the sum of two directional spectra:

- one with JONSWAP spectral density and mono-directional •
  the other monochromatic with cos2s dispersion

It is also assumed to have a depth of 100 m.

In this example, we are limited to two spectra, but the simulator makes it possible to sum as many as we wish (we are only limited by the memory of the machine and by the time available).

4.1.8 Writing the simulator configuration file

python echo=False, results='raw', name='tutorial_03_load_yaml' yaml_data = load_yaml('tutorial_03_waves.yml')

The environment models section is much more extensive than for previous tutorials .

We start by defining the discretization. Currently, the number of pulses is equal to the number of directions: this is a limitation of the code.

python echo=False, results='raw', name='tutorial_03_print_wave_discretization' print_yaml(yaml_data, 'environment models/0/discretization')

We will therefore sum <% yaml_data['environment models'][0]['discretization']
['n'] %> pulsations and <% yaml_data['environment models'][0]
['discretization']['n '] %> directions, i.e. <% yaml_data['environment models'][0]['discretization']
['n']yaml_data['environment *models'][0]['discretization']['n'] % > dots.*
However, the spatial discretization of monochromatic spectra and monodirectional dispersions is reduced to one point. We also specify that we want to represent <%yaml_data['environment models'][0]['discretization']['energy fraction']100 %> % of the total energy, the other components not being retained.

The first spectrum is defined as follows:

python echo=False, results='raw', name='tutorial_03_print_first_spectrum' print_yaml(yaml_data, 'environment models/0/spectra/0')

For the second spectrum, we write:

python echo=False, results='raw', name='tutorial_03_print_second_spectrum'
print_yaml(yaml_data, 'environment models/0/spectra/1')

The outputs are defined as follows:

python echo=False, results='raw', name='tutorial_03_print_outputs_section'
print_yaml(yaml_data, 'environment models/0/output')

Ultimately, the environment is defined as follows:

python echo=False, results='raw', name='tutorial_03_print_environment_yaml'
print_yaml(yaml_data, 'environment models')

As we do not simulate a body, the input file is reduced to:

python echo=False, results='raw', name='tutorial_03_print_full_yaml'
print_yaml_file('tutorial_03_waves.yml')

4.1.9 Launching the simulation

The simulation can now be started as follows:

python echo=False, results='raw', name='tutorial_03_launch_simulation'
execCommand('xdyn tutorial_03_waves.yml --dt 1 --tend 1 -w tutorial_03_results.h5')

The result file is here tutorial_03_results.h5.

4.1.10 Results

We get an hdf5 file that can be opened with different software like HDFView. In the “outputs” group, there is a “waves” group which contains four data sets named t, x, y and z.

- t gives the time steps of the simulation • x
  gives the coordinates according to x of the points where the elevation is calculated. Each line corresponds to a time step.
- y gives the coordinates along y of the points where the elevation is calculated. Each line corresponds to a time step.
- z gives the elevation at the points defined by x and y. Each slice corresponds at a time step.

The description of this file is made in the documentation of YAML files.

Elevations can be obtained in any xdyn frame (NED or linked to a solid). If the coordinate system is linked to a solid, we obtain x and y coordinates that change over time. ## Tutorial 6: propulsion

So far we have only simulated environmental stresses. In this tutorial, we simulate a thruster.

4.1.11 Description of the problem

The vessel is operating in a swell-free environment. It is subject to the following five efforts:

• Gravity •
Hydrostatic forces (rapid and not exact) • Viscous
damping • Propulsion force
due to a propeller • Resistance to progress

We can also be satisfied with only 2 efforts of resistance and propulsion.

4.1.12 Writing the simulator configuration file

Changes from tutorial 2 are the addition of damping and resistance forces, an external forces section and a commands section.

We start by defining the characteristics of the thruster:

python echo=False, results='raw', name='tutorial_06_load_yaml' yaml_data = load_yaml('tutorial_06_1D_propulsion.yml')

python echo=False, results='raw', name='tutorial_06_print_external_forces_section' print_yaml(yaml_data, 'bodies/0/external forces')

The commands are defined at the root of the YAML:

python echo=False, results='raw', name='tutorial_06_print_commands_section' print_yaml(yaml_data, 'commands')

Ultimately, the input file is:

python echo=False, results='raw', name='tutorial_06_print_full_yaml' print_yaml(yaml_data)

4.1.13 Launching the simulation

The simulation can now be started as follows:

python echo=False, results='raw', name='tutorial_06_launch_simulation' execCommand('xdyn tutorial_06_propulsion.yml --dt 0.1 --tend 20 -o out.csv')

4.1.14 Results

Here is the time evolution of the forward speed:

python echo=False, results='raw', name='tutorial_06_plot_results' data = csv('out.csv') plot = prepare_plot_data(data, x = 't', y = 'u(TestShip)', name="Vitesse d'avance") g = cartesian_graph([plot],

x='t (s)', y='U (m/s)') create_layout(graph=g, title="Longitudinal advance velocity") ## Tutorial 9 : Using a remote wave model

This tutorial explains how to use an external swell model in xdyn.

We will use Docker compose to launch the client (xdyn) and the swell server. This is not mandatory (you can do without Docker and Docker Compose to make the whole thing work), but using Docker greatly simplifies the implementation.

4.1.15 Overview

For this tutorial, you need:

- a swell model • an xdyn input

The wave model can be implemented in Python. In order to simplify its implementation , one can use the repository https://gitlab.sirehna.com/sirehna/demo_ docker_grpc which already contains a sample swell server in Python.

4.1.16 Setting xdyn data

In a YAML file (named tutorial_09_gRPC_wave_model.yml in this example) we write:

python echo=False, results='raw', name='tutorial_09_load_yaml' yaml_data = load_yaml('tutorial_09_gRPC_wave_model.yml')

python echo=False, results='raw', name='tutorial_09_print_yaml' print_yaml(yaml_data)

4.1.17 Writing the wave model

In a Python file (named airy.py in this example) we write:

“‘python evaluate=False, results=‘hidden’ """“Airy wave model. As implemented in xdyn.”"""

import math import yaml import numpy as np import waves

def pdyn_factor(k, z, eta): return 0 if (eta != 0 and z < eta) else math.exp(-k * z) class

Airy(waves.AbstractWaveModel): def init(self): self.psi0 = None self.jonswap_parameters = {‘sigma_a’: 0.07, ‘sigma_b’: 0.09} self.directional_spectrum = {}

def set_parameters(self, parameters): param = yaml.safe_load(parameters) self.jonswap_parameters['t_p'] = param['Tp']

```
        self.jonswap_parameters['gamma'] = param['gamma']
        self.directional_spectrum['omega'] = param['omega']
        self.directional_spectrum['psi'] = \ [param['waves
                propagating to']*math.pi/180] self.jonswap_parameters['hs_square']
        = param['Hs']*param['Hs'] self.jonswap_parameters['omega0'] = 2*math.pi/param['Tp']
        self.jonswap_parameters['coeff'] = 1-0.287*math.log(param['gamma'])
        self.directional_spectrum['si'] = [self.jonswap(omega) for omega in param['omega']]


        self.directional_spectrum['dj'] = [1]
        self.directional_spectrum['psi'] = [1]
        self.directional_spectrum['k'] = [omega*omega/9.81 for omega in
                                                              param['omega']]
        phases = np.random.uniform(low=0,
                                                   high=2*math.pi,
                                                   size=(len(param['omega']),))
        self.directional_spectrum['phase'] = phases

    def jonswap(self, omega): sigma_a
        = self.jonswap_parameters['sigma_a'] sigma_b =
        self.jonswap_parameters['sigma_b'] omega0 =
        self.jonswap_parameters['omega0'] hs_square =
        self.jonswap_parameters['hs_square'] coeff =
        self.jonswap_parameters['coeff'] gamma =
        self.jonswap_parameters['coeff'] sigma = sigma_a if
        omega <= omega0 else sigma_b ratio = omega0/omega alpha =
        ratio*ratio*ratio*ratio awm_5
        = coeff*5.0/16.0*alpha/omega*hs_square
        bwm_4 = 1.25*alpha kappa = (omega-omega0)/(sigma*omega0)
        return awm_5*math.exp(-
        bwm_4)*math.pow(gamma, math.exp(-0.5*kappa*kappa))



    def elevation(self, x, y, t): zeta = 0

        dir_spec = self.directional_spectrum psi =
        dir_spec['psi'][0] for s_i, k, omega,
        phase in zip(dir_spec['si'], dir_spec['k'], dir_spec['omega'],
                                                    dir_spec['phase']):
                                                    k_x_cos_psi_y_sin_psi =
                                                    k * (x * math.cos(psi) + y *
                math.sin(psi)) zeta -= s_i * math.sin(-omega*t + k_x_cos_psi_y_sin_psi + phase)

        return zeta
```

```
def dynamic_pressure(self, x, y, z, t):
    dir_spec = self.directional_spectrum eta =
    self.elevation(x, y, t) acc = 0

    psi = dir_spec['psi'][0] for s_i, k,
    omega, phase in zip(['si'], dir_spec['k'],
                                        dir_spec['omega'],
                                            dir_spec['phase']):
                                            k_x_cos_psi_y_sin_psi =
        k * (x * math.cos(psi) + y * math.sin(psi)) acc -= s_i *
                                              pdyn_factor(k, z,
        eta)*math.sin(-omega*t
                                                                  + k_x_cos_psi_y_sin_psi +
                                                                  phase)
    return 1000*9.81*acc

def orbital_velocity(self, x, y, z, t):
    dir_spec = self.directional_spectrum eta =
    self.elevation(x, y, t) v_x = 0 v_y = 0
    v_z = 0
    psi =

    dir_spec['psi'][0] for s_i, k,
    omega, phase in zip(['; si'], dir_spec['k'],
                                         dir_spec['omega'],
                                            dir_spec['phase']):
                                            pdyn_factor =
        self.pdyn_factor(k, z, eta) pdyn_factor_sh = pdyn_factor
        k_x_cos_psi_y_sin_psi = k*(x*math.
        cos(psi) + y * math.sin(psi))

        theta = -omega * t + k_x_cos_psi_y_sin_psi + phase cos_theta =
        math.cos(theta) sin_theta =
        math.sin(theta) a_affected = s_i * k /
        omega a_affected_pdyn_factor_sin_theta
        = a_affected * pdyn_factor \ * sin_theta

        v_x += a_involved_pdyn_factor_sin_theta * math.cos(psi) v_y +=
        a_involved_pdyn_factor_sin_theta * math.sin(psi) v_z += a_involved *
        pdyn_factor_sh * cos_theta

    return {'vx': v_x, 'vy': v_y, 'vz': v_z}

def angular_frequencies_for_rao(self):
    return self.directional_spectrum['omegas']
```

```
def directions_for_rao(self): return
        self.directional_spectrum['psis']

def spectrum(self, x, y, t): return
        self.directional_spectrum

if name == 'main': waves.serve(Airy())
```

Launching the simulation

We start by retrieving the sample swell model:

```bash
git clone git@gitlab.sirehna.com:sirehna/demo_docker_grpc.git
```

We then write a docker-compose.yml file:

```
version: '3'
services:
    waves-server:
        build: waves_grpc/python_server entrypoint:
        ["/bin/bash", "/entrypoint.sh", "/work/airy.py"] working_dir: /work

        volumes:
        - .:/work - ./
        waves_grpc:/proto xdyn:
    image:
        sirehna/xdyn working_dir: /
        data entrypoint: xdyn
        tutorial_09_gRPC_wave_model.yml --dt 0.1 --tend 1 -o tsv volumes:

        - .:/data
        depends_on:
        - waves-server
```

We can then run the simulation as follows:

```
docker-compose up
```

4.1.18 Without Docker

If you don't use Docker, you have to start the swell server manually:

```
python3 airy.py
```

Then you have to edit the xdyn input YAML file by replacing:

```
environment models:
    - model: grpc
      url: waves-server:50051
```

about

```
environment models:
  - model: grpc
    url: localhost:50051
```

We can then launch xdyn normally:

```
./xdyn tutorial_09_gRPC_wave_model.yml --dt 0.1 --tend 1 -o tsv
```

4.2 Tutorial 10: Using a remote effort model

This tutorial explains how to use an external effort model in xdyn.

We will use Docker compose to launch the client (xdyn) and the server (the effort model). This is not mandatory (you can do without Docker and Docker Compose to make the whole thing work), but using Docker greatly simplifies the implementation.

4.2.1 Overview

For this tutorial, you need:

- a force model (implemented in Python in this example) • an xdyn data entry (a YAML file)

4.2.2 Setting in xdyn data

In a YAML file (named tutorial_10_gRPC_force_model.yml in this example) we write:

python echo=False, results='raw', name='tutorial_10_load_yaml' yaml_data = load_yaml('tutorial_10_gRPC_force_model.yml')

python echo=False, results='raw', name='tutorial_10_print_yaml' print_yaml(yaml_data)

We create a file containing the commands for the gRPC model (tutorial_10_gRPC_force_model_commands.yml in this example):

python echo=False, results='raw', name='tutorial_10_load_yaml_commands' yaml_cmds = load_yaml('tutorial_10_gRPC_force_model_commands.yml')

python echo=False, results='raw', name='tutorial_10_print_yaml_commands' print_yaml(yaml_cmds)

The remote effort model connection is defined in the following section:

python echo=False, results='raw', name='tutorial_10_print_yaml_subsection' print_yaml(yaml_data, 'bodies/0/external forces')

- model: grpc tells xdyn that this is a remote effort model

- url: force-model:9902 gives the network address at which the force model can be reached. Using docker-compose here allows us to specify an address equal to the template name
- name: parametric oscillator is an arbitrary name that the user gives in his YAML file in order to be able to match any commands (commands section of the YAML file) to this force model.

All other lines are sent to the effort model as a parameter, without being interpreted by xdyn. In the present case, the model has two parameters *'k'* and *'c'* whose value is given once and for all at the start of the simulation.

4.2.3 Writing of the model of effort

This is a damped harmonic oscillator model:

$$F_x = -k\cdot x - c\cdot u$$
$$F_y = c\cdot \overline{v}$$
$$F_z = 0$$
$$M_x = 0$$
$$M_y = 0$$
$$M_z = 0$$

The force on the Y axis is proportional to the filtered speed. The definition of this filtering is done in the filtered states section of the xdyn YAML file.

In a Python file (named harmonic_oscillator.py in this example) we write:

```
“‘python evaluate=False, results=‘hidden’ """Damped harmonic oscillator model."""

import yaml import grpcforce

class HarmonicOscillator(grpcforce.Model): def init(self): self.k = None self.c = None

def model_needs_wave_outputs(self):
    return False

def set_parameters(self, parameters): param =
    yaml.safe_load(parameters) self.k = param['k']
    self.c = param['c']

def force(self, t, states,                    _, _, filtered_states): # The
    index in brackets corresponds to the position in # the history of states (starting
    with the most recent # value) and the corresponding instant is given by #
    states.t(i) .
```

```
        force = {'Fx': -self.k*states.x(0) - self.c*states.u(0), 'Fy': self.c*filtered_states.v  return {'forces': forces, 'extra outputs': {}}

if name == 'main': grpcforce.serve(HarmonicOscillator())
```

Launching the simulation

We start by retrieving the sample swell model:

```bash
git clone git@gitlab.sirehna.com:root/xdyn.git
```

We then write a docker-compose.yml file:

```
version: '3'
services:
  force-model:
    build: xdyn/grpc_force_python_server entrypoint: ["/
    bin/bash", "/entrypoint.sh", "/work/harmonic_oscillator.py"] working_dir: /work volumes: - .:/work - ./xdyn:/
    proto xdyn: image: sirehna/
    xdyn
    working_dir: /
    data entrypoint: xdyn

  tutorial_10_gRPC_force_model.yml

    tutorial_10_gRPC_force_model_commands  volumes:

    - .:/data
    depends_on: -
    force-model
```

We can then run the simulation as follows:

```
docker-compose up
```

4.2.4 Without Docker

If you don't use Docker, you have to start the swell server manually:

```
python3 harmonic_oscillator.py
```

Then you have to edit the xdyn input YAML file by replacing:

```
external forces:
  - model: grpc
    url: force-model:9002
```

about

```
external forces:
  - model: grpc
    url: localhost:50051
```

We can then launch xdyn normally:

```
./xdyn tutorial_10_gRPC_force_model.yml tutorial_10_gRPC_force_model_commands.yml --dt 0.1
```

changequote({{',}}')

4.3 Tutorial 11: using a remote controller

This tutorial explains how to use an external controller in xdyn.

We will use Docker compose to launch the client (xdyn) and the server (the effort model). This is not mandatory (you can do without Docker and Docker Compose to make the whole thing work), but using Docker greatly simplifies the implementation.

4.3.1 Overview

For this tutorial, you need:

- a controller (implemented in Python in this example)
- an xdyn data entry (a YAML file)

4.3.2 Setting in xdyn data

In a YAML file (named tutorial_11_gRPC_controller.yml in this example) we write:

python echo=False, results='raw', name='tutorial_11_load_yaml' yaml_data = load_yaml('tutorial_11_gRPC_controller.yml')

python echo=False, results='raw', name='tutorial_11_print_yaml' print_yaml(yaml_data)

This example contains two controllers:

- a remote PID controller (gRPC) implemented in Python and clocked at 0.3 seconds
- an internal xdyn PID controller, clocked at 0.7 seconds

These controllers are, in principle, implemented in the same way (with the differences of programming language). Two heading instructions are given:

- 30° from t = 0 to t = 250 seconds
- 45° from t = 250 seconds of simulation

The remote controller section is:

python echo=False, results='raw', name='tutorial_11_print_yaml_subsection' print_yaml(yaml_data, 'controllers/0')

• type: grpc tells xdyn that it is a remote controller • url: pid:9002 gives the network address at which the controller can be reached. Using docker-compose here allows us to specify an address equal to the name of the model • name: portside controller is an arbitrary name that the user gives in his YAML file in order to be able to access any outputs of the controller.

All other lines are sent to the controller as a parameter, without being interpreted by xdyn. In this case, the model has gains and weights assigned to states (same parameterization as xdyn's PID model).

4.3.3 Write controller

In a Python file (named pid_controller.py in this example) we write

python evaluate=False, results='hidden' include({{pid_controller.py}})

4.3.4 Launching the simulation with docker-compose

We start by retrieving the sample swell model:

git clone git@gitlab.com:sirehna_naval_group/sirehna/interfaces.git

We then write a docker-compose.yml file:

include({{{docker-compose.yml}}})

This file was created to be used in the source folder of xdyn and therefore the path must be adapted by replacing context: ../../interfaces by context: interfaces.

We can then run the simulation as follows:

CURRENT_UID=$(id -u):$(id -g) docker-compose up

The CURRENT_UID=$(id -u):$(id -g) part is simply used to ensure that any files generated are generated with the permissions of the current user.

4.3.5 Launching the simulation without Docker

If you don't use Docker, you have to start the swell server manually:

python3 pid_controller.py

Then you have to edit the xdyn input YAML file by replacing:

```
controllers:
    - type: grpc
      name: portside controller url:
      pid:9002
```

about

```
controllers:
    - type: grpc
      name: portside controller url:
      localhost:9002
```

We can then launch xdyn normally:

```
./xdyn tutorial_11_gRPC_controller.yml --dt 0.1 --tend 1 -o tsv
```

4.3.6 Launching the simulation from the xdyn repository

The tutorial can be launched directly from the xdyn repository by running make from the grpc_tests/controller directory. The simulation will then be launched and a CSV file will be generated. The convergence of the heading controller is then evaluated (at 30° between 0 and 250 seconds, then at 45° between 250 and 500 seconds). ## Tutorial 12: using PRECAL_R result files

This tutorial explains how to use files generated by PRECAL_R instead of HDB files for added masses.

4.3.7 Preparation: generation of PRECAL_R output files

The calcAmasDampCoefInfFreq flag must have the value true in the PRECAL_R input file (it is false by default). It is located in the section sim > parHYD > calcAmasDampCoefInfFreq. For more details, one can refer to the theoretical manual of PRECAL_R version 18.1.3 (Report No. 21447-7-RD, sections 2.3 and 2.4) and to its user manual (section 3.3.2, p. 25).

4.3.8 Configuration d'xdyn

If the PRECAL_R output file is called ONRT_SIMMAN.raodb.ini, the following bodies[0]/ dynamics/added mass matrix at the center of gravity and projected in the body frame section is used :

```
added mass matrix at the center of gravity and projected in the body frame:
    from PRECAL_R: ONRT_SIMMAN.raodb.ini
```

Here is the full example file:

python echo=False, results='raw', name='tutorial_12_load_yaml' yaml_data = load_yaml('tutorial_12_precal_r.yml') print_yaml(yaml_data)

In the output file of PRECAL_R, the section defining the mass matrix added to infinite frequency has the following appearance:

```
[added_mass_damping_matrix_inf_freq]

total_added_mass_matrix_inf_freq_U1_mu1 = {
```

0.110E+06,-0.888E-01,0.226E+06,-0.144E+00,0.270E+08,0.551E+01
-0.122E-01,0.344E+07,-0.563E-02,-0.113 E+07,0.157E+02,0.497E+08
0.227E+06,-0.898E+00,0.129E+08,0.763E+01,0.844E+08,0.130E+01
0.183E+00,-0.123 E+07,0.251E+01,0.498E+08,0.104E+03,0.338E+09
0.270E+08,0.106E+01,0.845E+08,-0.431E+02,0.119E+11,- 0.341E+03
0.164E+01,0.497E+08,0.101E+02,0.345E+09,-0.390E+03,0.522E+10 }

4.4 Tutorial 13: Using Potential Code Results with Remote Effort Models (gRPC)

This tutorial explains how to use the results of potential codes (HDB or PRECAL_R format) in gRPC effort models.

4.4.1 Configuration d'xdyn

python echo=False, results='raw', name='tutorial_13_load_yaml' yaml_data = load_yaml('tutorial_13_hdb_force_model.yml') yaml_data['output'][0]['data'] = yaml_data['output'][0]['data'][1:3]

To use an HDB or PRECAL_R file, add the (optional) hdb (or, respectively, raodb) key to the gRPC effort section:

python echo=False, results='raw', name='tutorial_13_grpc_yaml' print_yaml(yaml_data['bodies'][0]['external forces'][0])

The complete file is:

python echo=False, results='raw', name='tutorial_13_print_full_yaml' print_yaml(yaml_data)

xdyn does not guarantee that the HDB or PRECAL_R files used in the YAML are consistent: it is quite possible to use different files for the internal efforts and for the external efforts.

In the effort model (here a Python code), we can write:

```
class HDBForceModel(force.Model):
    """Outputs data from HDB in extra_observations."""

    def __init__(self, _, body_name, pot): """Initialize
        parameters from gRPC's set_parameters."""  self.body_name = body_name
        self.pot = pot


    def get_parameters(self):
        """Parameter k is stiffness and c is damping."""  return
        {'max_history_length': 0, 'needs_wave_outputs': False,
                'frame': self.body_name, 'x': 0, 'y': 0, 'z': 0, 'phi': 0,
```

```
                    'theta': 0, 'psi': 0, 'required_commands': []}

    def force(self, states,  _, __):
        """Force model."""
        extra_observations = {}
        extra_observations['Ma(0,0)'] = self.pot.Ma[0][0] return {'Fx': 0, 'Fy':
        0, 'Fz': 0, 'Mx': 0, 'My':
                    0, 'Mz': 0,



                    'extra_observations': extra_observations }

if __name__ == '__main__':
    force.serve(HDBForceModel)
```

The data is provided once, during model initialization, in the third parameter of the effort model constructor. All data is stored in Numpy types (ndarray) to simplify and speed up numerical processing.

This information can be written to the xdyn output files using the following output section:

python echo=False, results='raw', name='tutorial_13_outputs'
print_yaml(yaml_data['output']) changequote({{',}}')

4.5 Tutorial 14: using filtered states

This tutorial explains how to use filtered states in remote effort models and how to write them to xdyn output files.

4.5.1 Configuration d'xdyn

python echo=False, results='raw', name='tutorial_14_load_yaml' yaml_data = load_yaml('tutorial_14_filtered_states.yml')

To use filtered states, we define them in the filtered states section, defined for each body at the same level as the name or position of body frame relative to mesh keys, for example:

python echo=False, results='raw', name='tutorial_14_filtered_states_yaml' y = {'filtered states': yaml_data['bodies'][0]['filtered states']} print_yaml(y)

You can retrieve the filtered states in the xdyn output files using the following output section:

python echo=False, results='raw', name='tutorial_14_outputs'
print_yaml(yaml_data['output'])

The complete file is:

python echo=False, results='raw', name='tutorial_14_print_full_yaml' print_yaml(yaml_data)

If a state is not in the filtered states section, its filtered value will be the same as its unfiltered value.

In a Python file (named filtered_force.py in this example) we write

python evaluate=False, results='hidden' include({{filtered_force.py}})

We then write a docker-compose.yml file:

include({{docker-compose-filtered-states.yml}})

We can then run the simulation as follows:

CURRENT_UID=$(id -u):$(id -g) docker-compose up

The CURRENT_UID=$(id -u):$(id -g) part is simply used to ensure that any files generated are generated with the permissions of the current user.

Simulation results can be seen in the filtered_states.csv file:

python echo=False, results='raw', name='tutorial_14_plots' data =
csv('filtered_states.csv') xplot = prepare_plot_data(data, x='t', y='x(TestShip)',
name='État x non filtré') xfilteredplot = prepare_plot_data(data, x='t',
y='x_filtered(TestShip)', name='État x filtré') gx = cartesian_graph([xplot, xfilteredplot],
x='t (s)', y='Cavalement (m)') create_layout(graph=gx, title='Position x au cours
du temps') zplot = prepare_plot_data(data, x='t', y='z(TestShip)', name='État z non
filtré') zfilteredplot = prepare_plot_data(data, x='t', y='z_filtered(TestShip)',
name='État z filtré') gz = cartesian_graph([zplot, zfilteredplot], x='t (s)', y='Pillonement (m)')
create_layout(graph=gz, title='Position z au cours du temps')

We observe that :

- the values x_filtered and x are identical (since no filtering is defined for this degree of freedom) • the filtering is well taken into account in z