MoodyCore Manual
version 3.0

MOODY
Mooring Dynamics

JOHANNES PALM
CLAES ESKILSSON

Göteborg, Sweden 2023

Contact information: info@moodymarine.se

CONTENTS

1 Introduction **1**
1.1 General description 1
1.1.1 Versions 2
1.1.2 Acknowledgement of support 3
1.2 Installation 3
1.3 Basic usage 4
1.4 Debug tip 4

2 The input file **7**
2.1 Overview 7
2.2 Simulation 9
2.2.1 Time 9
2.2.2 Print 9
2.2.3 API 10
2.2.4 Numerical settings 10
2.2.5 Statics 10
2.3 Environment 12
2.3.1 Ground 12
2.3.2 Waves 14
2.3.3 Current 17
2.3.4 Wind 18
2.4 Vertices 19
2.5 Rigid bodies 20
2.6 Hydrobodies 25
2.7 Boundary Conditions 28
2.7.1 Translational modes 28
2.7.2 Rotational modes 30
2.8 Cable types 31

2.9 Material models . . . 33
2.10 Cables . . . 35
2.11 Initial Conditions . . . 36
2.12 Components . . . 38
2.13 Using `source` inheritance . . . 40

3 Running the code **41**
3.1 User input . . . 41
3.1.1 Flags . . . 41
3.1.2 Examples . . . 42

4 Pre processing **43**
4.1 Mesh manipulation . . . 43
4.2 Nemoh case preparation . . . 44
4.2.1 Input file format . . . 44
4.3 Wave elevation . . . 45
4.3.1 Handling output time . . . 46
4.3.2 Output format . . . 46

5 Post processing **47**
5.1 Moody post . . . 47
5.1.1 Usage . . . 47
5.2 MATLAB routines . . . 48
5.3 Python . . . 49
5.3.1 Dependencies . . . 49
5.4 Output file structure . . . 49
5.4.1 Cable output . . . 49
5.4.2 Rigid body output . . . 49
5.4.3 Hydrobody output . . . 49
5.4.4 Component output . . . 51

6 API documentation **53**
6.1 Introduction . . . 53
6.2 Interface functions . . . 53
6.2.1 Description . . . 54
6.3 Input file entries . . . 55
6.4 Fortran, Matlab and Python interface functions . . . 56
6.5 Testing the API . . . 56

REFERENCES **59**

A Theory manual **61**
A.1 Background . . . 61
A.2 Cable dynamics . . . 61
A.2.1 External Forces . . . 62

A.2.2 Shear Force Modelling 63
A.2.3 Tension-Strain Relations 64
A.3 Finite Element Method 64
A.3.1 Boundary Conditions 66
A.4 Components 66
A.4.1 Spring-dampers 67
A.5 Rigid body dynamics 67
A.5.1 Coordinate systems 67
A.5.2 Equations of motion 68
A.6 Wave kinematics and wave-body interaction 71
A.6.1 Wheeler stretching 71
A.6.2 First order potential flow forces 71
A.7 Static solver and relaxation 72
A.7.1 Solve 72
A.7.2 Relax 73
A.8 API boundary conditions 73

1

Introduction

1.1 General description

Moody was originally a module for computing cable dynamics in marine applications. It has been extended with functionality for floating bodies subject to linear potential theory with the radiation-diffraction formulation of Cummins equation. This manual explains the usage of – and theory behind – MoodyCore, as a stand-alone solver, as a plug-in module to hydrodynamic codes via the MoodyAPI library, and as the backend simulation engine to MoodyMarine - the graphical user interface of MoodyCore. MoodyCore is released as freeware on Mac, Linux and Windows. More information on `www.moodymarine.se`.

When using the different functionalities in MoodyCore, please make sure to reference publications accordingly. Below is a list of publications hosting the main feature developments and validations of MoodyCore.

Fundamental theory and numerical implementation of cable dynamics:
J. Palm, C. Eskilsson, and L. Bergdahl. An *hp*-adaptive discontinuous Galerkin method for modelling snap loads in mooring cables. *Ocean Engineering*, 144:266–276, 2017

Bending:
J. Palm and C. Eskilsson. Influence of bending stiffness on snap loads in marine cables: A study using a high-order discontinuous Galerkin method. *Journal for Marine Science and Engineering*, 8(10):795, 2020

Submerged bodies:
J. Palm and C. Eskilsson. Mooring systems with submerged buoys: influence of floater geometry and model fidelity. *Applied Ocean Research*, 102:102302, 2020

Floating bodies:
J. Palm and C. Eskilsson. Verification and validation of MoodyMarine - A free simulation

tool for modelling moored MRE devices. In J.M. Blanco Ilzarbe (ed.): *Proceedings of the 15th European Wave and Tidal Energy Conference*, Bilbao, 3-7 September 2023.

Coupling to OpenFOAM:
J. Palm, C. Eskilsson, G. Paredes, and L. Bergdahl. Coupled mooring analysis for floating wave energy converters using CFD: Formulation and validation. *Int. J. of Marine Energy*, 16:83–99, 2016

MoodyCore is based on a discontinuous Galerkin finite element method with high-order Legendre polynomial expansion bases. The formulation is stated in conservative form, employing a local Lax-Friedrich type Riemann solver for the numerical fluxes between the elements. For more information on the numerical schemes and the equations of motion, the reader is referred to the theory in Appendix A.

The stand-alone solver of MoodyCore, `moody.exe`, is typically operated from the terminal window or executed via MoodyMarine. To use MoodyCore as a mooring module with an external hydrodynamic code package such as OpenFOAM or Matlab, please see the API Chapter 6.

1.1.1 Versions

The current release (v3.0) features many improvements to the performance and handling of the code. New features in the current version are described below. MoodyCore has been released in the following previous versions (up until v2.0 under the name Moody):

Moody-1.0 Original release following the implementation in [8]. Released 2018-05-21.

Moody-2.0 Rigid body library introduced, enabling submerged buoys and clump-weights according to [6]. Released 2019-09-20.

MoodyCore-3.0 [Present release] Released 2023-10-06.

- Floating wave-body interaction supported as `hydroBodies` using linear potential radiation-diffraction theory according to Cummins equation [2]. Supports nonlinear Froude-Krylov forces and Nemoh v3 reader/writer.
- Major improvements to the static-equilibrium solver, including nonlinear statics of hydrobodies. Still under development, but system statics are now tunable from the input file.
- A native mesh-manipulation included supporting read/write from/to .stl, .gdf and .dat (nemoh) formats including panel splitting at the still water level.

- A major renaming and reorganization of the input file has been implemented, applying a more stringent format and supporting nested objects. The format has been modified to support compatibility with MoodyMarine.
- Introducing bending stiffness in the DG formulation according to [5].
- Linear translating mass included as a special type of rigid body, and components introduced to model linear spring/dampers according to [7] (command-line only).
- Post-processing module in Python provided (rudimentary).
- Many stability improvements and error messages introduced. Restructuring of the code base into Core and API repositories. The coupling to OpenFOAM is thus no longer shipped with the moodyCore.

1.1.2 Acknowledgement of support

The initial development of Moody was supported by the Western Region of Sweden through the Ocean Energy Centre at Chalmers University of Technology, Sweden (2013-2014). The formulations and code up until the present release has been funded by the Swedish Energy Agency under grants P40428-1, P42246-1, P47264-1 and P50196-1. The development up until September 2020 was done at Chalmers University of Technology, and has since been done at (and has been co-funded by) Sigma Energy and Marine AB. The financial support received is gratefully acknowledged.

1.2 INSTALLATION

MoodyCore is included in the installation of MoodyMarine, for use through the GUI. The following instructions pertain to installing MoodyCore separately.

MoodyCore is released as freeware, however the API coupling code is released open source under the collective name *MoodyAPI*. As of v.2.0.1, the compressed files include the `bin`, `lib`, `include` and `API` folders for each operating system (Windows, OSX and Linux). The tutorials, the manual and the API are all cross-platform and are accessed directly as sources when cloning the repository. Note that the automated shell scripts are developed primarily for Linux and MacOSX users. For windows, they are more guidelines than actual rules. Installation instructions:

1. Download the tar or zipped folder from `www.moodymarine.se/downloads`.
2. Select your installation directory and unpack the appropriate tar or zip binaries **in the same location**. Technically, this is only relevant for all tutorials and Matlab-processing paths to work as intended.

3. In Linux or OSX environments, source the environment variables. For this, use `moodyCoreEnvironment`. This will set, among others, the moody home directory variable `moodyDir` which we use frequently in the following manual.

   Ex:

```
cd moodyAPI
tar -xzvf moody-Linux.tar.gz
source moodyCoreEnvironment
```

> **NOTE:** There is no Windows PATH-setting script provided at this time. Use the native environment variable editor to add MoodyCore to your path.

1.3 Basic usage

Apart from running the MoodyMarine interface, MoodyCore is most easily operated through the terminal window. Alternatively, both MATLAB and Python interfaces (system calls) are included. The simulation is controlled through an input file and standard flag-value pairs e.g. `-f <fileName>`.

MoodyCore is also shipped with a suite of MATLAB® and Python scripts mainly used for post-processing. These are optional to use and are simply provided as a way of loading and processing the output data from MoodyCore. They are not fully documented and commented but are supplied to help users interact with MoodyCore. The Matlab scripts also work in GNU Octave, except for moodyMovie.m which is not compatible with Octave at present. The output data format of Moody is presented in chapter 5. Please note that any text-file can be used as input file, and the ".m" in the examples is only for convenience if the Matlab post-processing suite is used.

As of version 3.0, MoodyCore also supports .json input files, which is the format of input files from MoodyMarine. A copy of the input file is written in both .json and .txt format in each result directory from MoodyCore.

1.4 Debug tip

If there are any problems running the software, a good starting point is to check the output of the following commands (in a Linux environment):

```
echo $moodyPath
echo $moodyDir
which moody.exe
ldd libmoodyAPI.so
echo $LD_LIBRARY_PATH
echo $PATH
```

2

The input file

2.1 Overview

MoodyCore is based on an input file for setting up the mooring system and the floating structures. The input file is based on a number of top level keywords: simulation, environment, vertexLocations and bem. In addition, objects are added as numeric id naming, e.g. bc1, cable1, cable2 etc. Here we use `X` to mark the id of an object. There are five types of objects in MoodyCore: cables, components, rigid bodies, hydro bodies, and boundary conditions. These are all inter-connected at vertices. A boundary condition (bc) can connect to multiple cables and components, but not to a rigid body. A rigid body and a hydro body can have several slave vertices, but a vertex which is enslaved cannot be the main vertex of another body. The input data is grouped in the following main categories.

Simulation
: Controls time, output, API and numerical settings. Keyword: `simulation`

Environment
: Hosts gravitation, fluid properties, ground model, waves, wind and current. Keyword: `environment`

Vertices
: Specifies the number and location of vertices (points) in the simulation., Keyword: `vertexLocations`.

Boundary conditions
: Specify type and behaviour for all boundary conditions. Keyword: `bcX`

Cable types
: Specify the cross-sectional and material parameters of a mooring cable. Keyword: `cableTypeX`

Cables

Specify cable properties like end-points, cable type and length. Also specified initial condition type. Keyword: `cableX`

Components

Specify component properties like end-points, stiffness and damping. Keyword: `componentX`

Rigid bodies

Dynamic rigid bodies are submerged Morison bodies, like buoys and clump-weights. Keyword: `rigidBodyX`

Hydro bodies

Derived from the rigid body library, hydrobodies support linear potential flow wave-forces through the radiation-diffraction formulation of the Cummins equation [2]. Keyword: `hydroBodyX`

The input file can have any text-file format, with or without extension. It follows the format of a Matlab script, where assignment is made with =. Hence, `simulation.time.end = 5` will set the end time of the simulation to 5 s. Multiply defined variables are allowed, but be aware that only the first instance of a keyword will be used. Sub-structures and fields of information are assigned in nested curly brackets {} or through the `.` sign. The following sections describe each input category for a MoodyCore simulation.

2.2 Simulation

2.2.1 Time

Keyword: `time`, handles simulation time.

`start` [s]
Start time of the simulation.

`end` [s]
End time of the simulation.

`dt` [s]
Time step size.

`scheme` (RK3)
Choose integration method. Choices are:

`RK3`
Third order explicit Strong Stability Preserving Runge-Kutta scheme. This is the default.

`RK45`
Five stage fourth order explicit Runge-Kutta scheme.

`cfl`
Use adaptive time step size. Set as a fraction of the smallest CFL condition. Typically `cfl=0.9` works for most applications, but in cases dominated by ground, high bending stiffness or large internal damping, a lower value may be needed for stability. If field `cfl` is present, field `dt` is not used. It is also possible that cfl=0.9 is too restrictive. Hence, best performance is probably achieved by testing out a stable constant time.dt.
.

2.2.2 Print

Keyword: `print` controls the output folder content and format. (x) indicate default value.

`dt` (time.dt) [s]
The time between each output time in the result directory.

`mode` (1)
The `mode=1` prints all cable values in both modal and nodal space. `mode=0` prints only modal coefficients. Use moodyPost.exe to extract nodal output, see chapter 5.

`format` ('bin')
: Moody prints in binary format by default. `format='ascii'` prints in ASCII format. Binary format is significantly faster and more efficient for storage. It is also required for MoodyMarine compatibility.

2.2.3 API

The API input structure is located under `simulation`. Please see Section 6 for detailed information on how to set the `API` keyword.

2.2.4 Numerical settings

Numerical settings control the quadrature order of the simulations. The top-level keyword is `numLib`.

`qPointsAdded` (2)
: The number of quadrature points more than the polynomial order P, to use in the simulation. Default is 2, meaning that $Q = P + 2$ points will be used in each element. Moody uses Gauss-Legendre-Lobatto quadrature which is exact with $P+2$ points for the Legendre polynomial of order P. However, in cases of nonlinear external forces (such as stiff ground interaction) a higher value might be needed to sample the force properly.

2.2.5 Statics

NOTE: The `relax` stage in the static solver is still experimental and has its limitations.

The static solver (`statics`) option enables a number of Euler fwd time steps before the time loop begins. The `solve` option works well to find system equilibrium, however it is deactivated by default. Input options are:

`solve` (0)
: Switch to 1 to enable the dynamic solve of the initial condition.

`maxIter` (5000)
: An `int` denoting the number of maximum iterations of the Euler step during the solve stage.

`timeStep` (0.1)
: The time step size used to step the bodies forward towards equilibrium. Body velocity is truncated after each step. (s)

`tol` (0.001)
: Tolerance of maximum body velocity (in any dof) of the system allowed before breaking the solve-stage of the static solver. (m/s)

`relax` (0)
Switch to 1 to enable the dynamic relaxation of the initial condition.

`relaxFactor` (0.1) Factor to scale the resulting cable velocity after each step.

`relaxIter` (5000) How many time steps to relax over.

`relaxTimestep` (0.001) Time step size of relaxing stage. (s)

2.3 Environment

Keyword: `environment` controls the environmental properties in the simulation. The following are the top-level keywords available.

`gravity` (-9.81)
: Gravitational acceleration specified as a scalar in m/s^2.

`waterLevel` (inf) [m]
: Set still water z-coordinate. Default is fully submerged.

`waterDensity` (1000.0) [kg/m^3]
: The water density applies to all fluid below the water level.

`airDensity` (1.0) [kg/m^3]
: The air density applies to all fluid above the water level.

`ground`
: Ground model. See 2.3.1.

`wave`
: Wave model. See 2.3.2.

`current`
: Current model. See 2.3.3.

`wind`
: Wind model. See 2.3.4.

2.3.1 Ground

LIMITATION: The ground level is assumed to be constant.

Keyword: `ground` specifies the ground interaction properties of all cables in the model. MoodyCore uses a combined linear spring and bilinear damper as ground model, `springDamp`. The contact force acting on each quadrature point of the cable from the ground is a function of the penetration depth and the vertical velocity. Dynamic friction (Coulomb damping) is also included. The input parameters listed below refer to the equations stated in the theory manual of Appendix A.

`level`
: This sets the level of the ground in the global coordinate system. The ground is assumed to be horizontal and thus have its normal vector aligned with the global z-axis. Goes into (A.12) as z_0.

`type=springDamp`
: This is the only one available at present.

`stiffness` [Pa/m]
: The stiffness (bulk modulus) of the ground. Goes into (A.12) as K.

`damping` (0)
: The damping input sets vertical linear damping (Pa s/m). Goes into (A.12) as ξ.

`dampingCoeff` (1)
: The damping coefficient sets vertical linear damping as ratio of critical damping. (-). Goes into (A.12) as ξ.

`oneWayDamping` (1)
: Switch to apply vertical damping only during penetration ($v_z < 0$). If set to 0, then pure linear damping is applied.

`frictionCoeff` (0)
: This is the friction coefficient between any cable and the ground. Goes into (A.13) as μ.

`frictionVelocity` (0.01) [m/s]
: The cut off speed for dynamic friction specifies the horizontal speed at which the friction force reaches its full value. This is a numerical relaxation of the friction force for the times when the cable is changing direction. Goes into (A.13) as v_μ.

```
ground = {
    level            = -1e3; % z-coordinate of ground [m]
    type             ='springDamp'; % type name
    stiffness        = 1e6;  % vertical stiffness [Pa/m]
    damping          = 0;    % vertical damping [Pas/m]
    dampingCoeff     = 1;    % ratio of critical damping [-]
    frictionCoeff    = 0;    % horizontal friction coeff. (-)
    frictionVelocity = 0.01;% horizontal threshold [m/s]
}
```

2.3.2 *Waves*

LIMITATION: Waves and current can coexist, but their effects are superposed. No modification of the linear wave theory is made to consider the effect of the current at present.

MoodyCore supports regular (Airy) waves, as regular waves, JONSWAP spectrum and custom wave trains. Keyword in environment: `wave`.

`type`
: Options are: `regular`, `JONSWAP`, `custom`. See further individual settings.

`direction` (0) [deg]
: Wave direction angle. Zero degrees is a wave propagating in the positive x-direction. θ in Eq. (2.1).

`depth` [m]
: The water depth used for wave modelling may be set independently. If not specified, it is computed from waterLevel - ground.level.

`rampTime` (0) [s]
: The ramp time of the wave amplitude, using a cosine ramp function.

`type=regular`
: Regular waves in deep and intermediate waters are simulated using the airy wave theory. The free surface is computed as

$$\zeta = a \cos\left(k_x x + k_y y - \omega t + \phi\right) , \tag{2.1}$$

from the amplitude a, directional wave numbers $k_x = k \cos\theta$, and $k_y = \sin(\theta)$, angular frequency ω and phase ϕ. θ is then the wave direction, defined as an angle to the global x-axis. Input fields:

`amplitude` (0) [m]
: Wave amplitude.

`period` ($2\pi/\omega$) [s]
: Wave period. Precedence over k, L and ω input.

`omega` ($\omega(k)$) [rad/s]
: Wave frequency period. Precedence over k and L input.

`L` ($2\pi/k$) [m]
: Wave length, single input. Precedence over k input.

`k` ($k(\omega)$) [rad/m]
: Wave number, single input. used only if period, omega and L are missing inputs.

`phase` (0) [deg]
Wave phase shift acc. to ϕ in Eq. (2.1). θ in (2.1).

`useRadians` (0)
If set to 1, it interprets phase and direction input as radians instead of degrees.

`type=custom`
Use user specified wave component input. can be used to model any seastate, and is required to use for multi-directional wave environment simulations.

`file`
File name of the wave component file. If present, explicit input of direction,amplitude,phase and frequency are unused. Expected file format has four lines per wave direction. L1: waveDirection (angle relalative to the X-axis), L2: amplitudes (list of N values, where N is number of components), L3: frequencies (list of N values), and L4: phases (list of N values).

`direction` S
ingle wave train wave direction. Unit subject to `useRadians` switch.

`amplitude` [m]
List of wave amplitudes.

`frequency` L
ist of wave frequencies. Unit subject to `useRadians` switch.

`phase` L
ist of wave phases. Unit subject to `useRadians` switch.

`useRadians` (0)
Switch to use radians in file reading or not. 0 means direction and phases in degrees, frequencies in Hz. 1 means direction and phases in radians, frequencies in rad/s.

`type=JONSWAP`
Long-crested irregular waves are modelled using synthesized waves from the JONSWAP amplitude spectrum. Inputs are named after the JONSWAP spectrum formula

$$S(\omega) = \alpha 2\pi g^2 \omega^{-5} \exp\left(-1.25 \frac{\omega}{\omega_p}^{-4}\right) \gamma^b \tag{2.2}$$

$$b = -\sigma^{-2} \left(\frac{\omega}{\omega_p} - 1\right)^2, \text{with } \sigma = \sigma_1 \text{ if } \sigma < \omega_p, \; \sigma = \sigma_2 \text{ otherwise,} \tag{2.3}$$

where w is in rad/s, w_p is the peak frequency, γ is the peakedness factor, $\alpha = 0.0081$ is a scale factor for significant wave height, and g is the gravitational acceleration.

`Hs` [m]
: Significant wave height.

`components` (
: 100) Number of wave components in the spectrum.

`fp` (0.1)
: Peak frequency. Unit subject to `useRadians`.

`f0` (0.01)
: Lowest frequency. Unit subject to `useRadians`.

`f1` (10)
: Highest frequency. Unit subject to `useRadians`.

`gamma` (
: 1) Peakedness factor of the spectrum.

`sigma1` (0.07)
: Low frequency shape factor.

`sigma2` (0.09)
: High frequency shape factor

`seed` (-1)
: Seed number. Use -1 for random seed generation, use positive int for repeatability.

```
wave = {
    type = 'regular';
    amplitude = 0.5;
    period = 6;
    phase = 90;
    rampTime = 10;
    direction = 180;
}
```

2.3.3 Current

Keyword in environment: `current`.

`direction [deg]` H
orizontal direction of current.

`rampTime` (0) [s]
Ramp time of current. A sine ramp is applied from simulation start to `rampTime`. No acceleration forces generated (assumed small).

`speed` (0) [m/s]
Uniform speed of current. Used if `depths` is NOT found.

`depths` [m]
List of depths at which `speeds` list is defined. A lookup table of [depth,speed] is used to interpolate current value. Depths are positive down using still water level as origin.

`speeds` [m/s]
List of speeds corresponding to the depths in `depths`.

```
current = {
    direction = 90;
    speed = 0.5;
}
```

2.3.4 Wind

Keyword in environment: `wind`. The power law wind profile is computed at each height z as

$$U(z) = U_r[\cos\theta \ \ \sin\theta]^T (z/z_r)^\alpha \,. \tag{2.4}$$

Here, U_r is the reference speed, z_r is the reference height and α is the power exponent.

`direction` [deg]
: Horizontal wind direction θ.

`speed` (0) [m/s]
: Reference speed U_r of wind at $z = z_r$.

`zRef` (10) [m]
: Reference height z_r at which `speed` applies.

`rampTime` (0) [s]
: Ramp time of wind. A sine ramp is applied from simulation start to `rampTime`. No acceleration forces generated (assumed small).

```
wind = {
    direction = 90;
    speed = 10;
    zRef = 10;
    rampTime = 20;
}
```

2.4 VERTICES

MoodyCore's geometry is defined by `vertexLocations`, which specify a numbered list of points in three dimensional space. A vertex number is needed to assign a physical location to any dynamic object in the simulation. All points are given in 3D coordinates.

The following example creates vertex `1` and `2` at points `[0 0 -50]` and `[50 0 0]` respectively. Note that the `[]` brackets are optional.

```
vertexLocations  = {
1 [0 0 -50];
2 [50 0 0]
};
```

2.5 Rigid bodies

NOTE: The motion of submerged rigid bodies is assumed to be following Morrison's equation, and no check is made in MoodyCore that this is a valid approximation. Therefore one should take care when simulating large bodies in relation to the flow.

Mooring elements such as submerged floaters and sinkers can be modelled using rigid body type objects, defined by keyword `rbX` for each number X. These are divided into two categories. One is the simple point mass with an assigned mass and volume but with neglected rotational motion. This is modelled using the `rigidPoint` type object. The other rigid body type also takes the rotation of the body into account and computes the motion from the force and moment exerted by the attached cables and the surrounding fluid.

The force and position fluxes between rigid bodies and the attached mooring cable(s) are such that the position of the body acts as a position boundary condition on the cable, while the cable acts as a force boundary condition to the rigid body solver. Some fields are common to all rigid bodies while other are object type specific.

`vertex`
: The vertex number of the body. It is assumed to be the center of mass of the body.

`mass` [kg]
: The mass of the body. May not be 0.

`slaves`
: List of slave vertices. Each slave is defined by 8 entries:

$$V\ ,\ x\ ,\ y\ ,\ z\ ,\ clampSwitch\ ,\ tx\ ,\ ty\ ,\ tz\ ,$$

where V is a vertex no, (x, y, z) is the vertex location in the body local coordinate system, *clampSwitch* is 0 for a pinned (moment-free) connection and 1 for a clamp at V directed in (tx, ty, tz) direction.

NOTE: Assigning a slave vertex overrides the information supplied to the vertex position in `vertexLocations`. It is treated as a local coordinate forced to follow the rigid body motion of the master body.

NOTE: Only connected cables with bending stiffness are affected by the *clampSwitch*. Torsional moments are NOT transmitted from cables.

`volume` (0) [m^3]
: The volume of the body.

`constraints`
: Vector list of constrained (locked) degrees of freedom. Only translation and rotation about the global coordinate system is supported. DoF 1-3 are translations and DoF 4-6 are rotations. Example of heave only simulation: [1,2,4,5,6].

`type=rigidPoint`
: The rigid point is the object type used to simulate point masses and point floatation forces in Moody. It computes the motion of the body in the three translational degrees of freedom. The local coordinate system is therefore always aligned with the inertial system. Any cables connected to clamped slaves are affected by the directional clamp, but the reaction moments acting on the body are not affecting the body motion. This body type has a buoyancy model which (based on a sperical geometry) computes the submerged volume at each timestep.

  `type`
  : The `type` field must be set to `'rigidPoint'`.

  `CDn and CM` (0)
  : The hydrodynamic coefficients of drag and added mass, `CDn` and `CM` respectively, are both specified as single input values and used according to the Morison equation.

  `initialState` (0)
  : A substructure with subfield like `r`, and `v`. The body can be given a start velocity and a startposition. The input is a 3 by 1 vector in [m/s] of the global coordinate system. The default is the position from vertexLocations at rest.

  `area` [m^2]
  : Area of body, used in drag force computation. If not specified, it is computed from volume assuming a sphere.

```
rb1={
    type = 'rigidPoint';
    vertex = 2;
    mass = 1; \% [kg]
    \% Optional \%
    volume = 0;
    CDn = 1;
    CM = 2.5;
    initialState.v = [1 0 0]; % 1m/s x speed
}
```

`type=rigidCylinder`

A submerged, vertical cylinder is modelled, including both translation and rotation. The rotation is modelled using quaternions and the variation of the surrounding fluid is taken into account by integration of the forces on each slice.

> **LIMITATION:** A `rigidCylinder` object is not aware of the free surface and thus assumed to be fully submerged always.

`height` [m]
: The height of the cylinder must be specified. The input is in [m].

`diameter` [m]
: The diameter of the cylinder.

`momentOfInertia, I` [kg m^2]
: This is the mass moment of inertia of the body, around the center of gravity. If not specified, the default value is that of the solid cylinder. The input should be in [kg m^2] and formatted as a vector of three values: `rigidBodyX.I = [Ixx Iyy Izz]`, where z is along the symmetry axis.

`centreOfBuoyancy, cob` (0) [m]
: Use cob to offset the center of buoyancy (along the symmetry axis only).

`CM` ([0 0])
: A vector of two values `rigidBodyX.CM=[CMn CMt]`, where `CMn` is the normal direction coefficient. `CMt` is the cylinder lid coefficient.

`CD` ([0 0 0])
: A vector of three values `rigidBodyX.CD=[CDn CDt CDyaw]`, where `CDn` is the normal direction coefficient. `CDt` is the cylinder lid coefficient, and `CDyaw` is the total yaw drag coefficient.

`initialState` (zeros(13,1))
: All instances of initial state can be set here. Subfields are: r-positions, v-velocity, ω-angular velocity and q-quaternion object.

`decoupledDrag` (0)
: Set to 1 to use the drag force in decoupled mode, i.e. to compute rotational drag moment based on ω of body alone, and compute the translational force based on the relative velocity of body and fluid. Default is 0, i.e. the strip theory is used to compute the drag forces and moments at each time-step using 7-point numerical quadrature along the symmetry axis. See [6] for more information on the differences between rotational drag methods.

```
rb1={
    type=rigidCylinder
    vertex = 2
    mass =5
    height = 0.5
    diameter = 0.2
    I = [10 5]
    CDn = [1 1 0.1]
    CM = [1.2 1]
}
```

`type=translator`

A special purpose type to model linearly translating bodies. Also includes models of end-stop forces when a damping force and limited stroke-length is desired.

`type`
: The `type` field must be set to `'translator'`.

`V` [m^3]
: The volume of the body used for buoyancy forces. Unused if `rhoSeal` = 0.

`midPoint` [m]
: Position of center of the stroke-length.

`direction` [m]
: Direction of the linear motion. Input as a vector.

`damping` [Ns/m]
: Linear damping is applied to the body motion.

`friction`(0) (N)
: Dynamic friction force amplitude.

`vFriction`(0.01) (m/s)
: Define linear transition region for friction. Friction force amplitude is constant for $|v| >= vFriction$, with value `friction`.

`strokeLength` [m]
: One-sided distance to end-stop springs. i.e, the amplitude of free motion around the `midPoint`.

`endStopStiffness` [N/m]
: Stiffness of linear end stop spring. Activated as restoring spring when distance from `midPoint` exceeds `strokeLength`.

`endStopDamping` [Ns/m]
: Linear damping of the end stop.

`endStopLength` [m]
: Length of end-stop spring.

`endStopFullCompressionStiffness` [N/m]
: Activated as a second layer of stiffness when the `endStopLength` is exceeded.

`sealed`
: Switch to model a sealed container, where buoyancy is not affecting the translating body.

`rhoSeal` [kg/m^3]
: Density of seal fluid. Defines buoyancy force when `sealed`.

See example in tutorial T4.

2.6 HYDROBODIES

LIMITATION: There is no body-to-body interaction implemented in MoodyCore v-3.0.

Wave-body interaction of floating bodies in moodyCore is done through a `hydroBody`. A `hydroBody` implements Cummin's equation [2] with nonlinear additives in the form of Morison drag, mesh-based nonlinear Froude-Krylov pressure force (static and dynamic) and external forces. Presently [v-3.0], moodyCore supports diffraction forces through numerical integration of the impulse-response function.

`vertex`
: The body vertex. The vertex position is the centre of gravity of the body.

`slaves`
: List of slave vertices. The same as for a rigid body.

`mass` [kg]
: Dry mass of body

`I` [kgm^2]
: Moment of inertia. Vector of length 9 or 3 corresponding to the full mass moment of inertia tensor, or its diagonal terms. Should be computed about the centre of gravity of the body.

`constraints`
: Vector list of constrained (locked) degrees of freedom. Only translation and rotation about the global coordinate system is supported. DoF 1-3 are translations and DoF 4-6 are rotations. Example of heave only simulation: [1,2,4,5,6]

`meshName`
: Name of mesh to use for Froude-Krylov pressure calculations. Preferably with full path (for full compatibility with moodyMarine). WAMIT (.gdf), Nemoh (.dat) and stl (.stl) formats supported. All are read as triangular meshes. See documentation on `moodyPre.exe` for mesh conversion usage. Used when `nlfk` switch is turned on, or when no stiffness matrix is supplied.

`cogInMesh` [m] (0,0,0)
: Location of centre of gravity in the *mesh* coordinate system. MoodyCore will internally translate the mesh geometry to be centered around the CoG.

`C` [N/m,Nm/rad]
: Linear stiffness matrix. Used if present and if `nlfk` is turned off. Input is

a list of values with some shorthand input syntax supported. Let L be the length of the `C` input:

$L = 1$
: Input is interpreted as the heave stiffness, i.e. `C=[c33]`. All other stiffnesses are 0.

$L = 3$
: Input is interpreted as the heave, roll and pitch stiffness, i.e. `C=[c33,c44,c55]`. All other stiffnesses are 0.

$L = 36$
: Full stiffness matrix.

`volume` [m^3] (0)
: Submerged volume at initial position. Used in cases where linear stiffness matrix is found. If not present in such cases, a 0 displacement body is simulated.

`bemData`
: Location of folder with bemData results. Expects an output in Nemoh (v3) format with nested folders `mesh` and

`type`
: Presently, this has to be `linearIRF`.

`IRF`
: Substructure governing the impulse response function (IRF) settings.

`time` [s]
: Duration of IRF function integration.

`dt` [s]
: Resolution of the IRF integration.

`nlfk` (0)
: Switch to turn on (1) the nonlinear Froude-Krylov forces. Default is off (0).

> **NOTE:** A triangulated non-symmetric body mesh will give rise to erroneous yaw moment if the nonlinear Froude-Krylov option is used. Work-arounds include to import a quadrilateral (gdf) mesh to MoodyCore, use mirrored (symmetric) triangular mesh, or simply restrain the body in yaw mode.

`CD` [0 0 0 0 0 0] (Ns^2/m^2, Nms^2)
: Effective drag force coefficient for quadratic drag. Note, that it is given as a six dof vector where index 1-3 is computed as relative motion at the cog, and rotations (index 4-6) are computed based on body rotation alone.

`CDlinear` [0 0 0 0 0 0] (Ns/m, Nms)
: Effective drag force coefficient for linear drag. Note, that it is given as a six dof vector where index 1-3 is computed as relative motion at the cog, and rotations (index 4-6) are computed based on body rotation alone.

2.7 Boundary Conditions

Boundary conditions are created through the input file using top-level syntax bc1, bc2 etc.

`vertex`
: Each boundary condition can only be applied to one vertex, specified by `vertex`. The externalRigidBody mode can be used to also control slave vertices (see API section).

`startTime` (time.start) [s]
: Specify the start time of a time-variation of the bc.For $t <$`bcX.startTime`, the boundary value is held fixed at that of `bcX.startTime`. The input is either a single value or a 3 by 1 vector specifying the start time of each coordinate direction independently.

`endTime` (time.end) [s]
: Defines the time at which the boundary condition stops to be active. For $t >$`bcX.endTime`, the boundary value is held fixed at that of `bcX.endTime`. The input is either a single value or a 3 by 1 vector specifying the end time of each coordinate direction independently.

`rampTime` (0) [s]
: Dynamic boundary conditions are ramped up from static start time conditions over a given time interval $\Delta\tau$=`bcX.rampTime`. The ramp factor Q is defined as a function of $\tau = t-$`startTime` as

$$Q = 0.5\left(\sin\left(\pi\frac{\tau}{\Delta\tau} - \frac{\pi}{2}\right) + 1\right) . \tag{2.5}$$

`dampTime` (0) [s]
: Analogous to `rampTime` for smoothly stopping the motion of a boundary condition over a period of time (`dampTime`) as the end time is approached.

`mode` (fixed)
: Specify BC mode for translational degrees of freedom.

`rotationMode` (pinned)
: Specify BC mode for rotational degrees of freedom.

2.7.1 *Translational modes*

`mode=fixed`
: Model a fixed point.

  `value` (vertexLocation) [m]
  : Specify coordinate to override default vertexLocation.

`mode=force`
Model a constant force. Default models a free cable end.

`value` (0,0,0) [N]
Specify constant force amplitude in outward-pointing normal direction.

`mode=sine and mode=sineForce` T
hese modes are used to generate sinusoidally varying values at the boundary. The offset, amplitude, frequency and phase of the sine motion can be specified for each coordinate direction. Scalar valued input is allowed and will be applied to all directions. The amplitude and the centerValue are interpreted as [m] for mode=`sine` and as [N] for mode=`sineForce`.

`amplitude` ([0,0,0]) [m alt. N]
The amplitude of the sinusoidal excitation.

`frequency` ([0,0,0]) [Hz]
The frequency of oscillation.

`phase` ([0,0,0]) [deg.]
The phase of the excitation.

`centerValue` (vertexLocation alt. [0,0,0]) [m alt. N]
If defined it determines the offset around which the oscillation takes place. Must be vector valued with length 3.

```
bc1={
    type = 1; % position
    vertex = 2;
    mode = sine;
    amplitude = [1;0;1];
    frequency = [0.5,0,0.5];
    phase = [90 0 0]
    rampTime = 5
    }
```

`mode=externalPoint`
This mode is used in MoodyAPI. It uses quadratic interpolation to generate intermediate boundary conditions in between coupling times of the external solver. When used for mooring through MoodyAPI, no additional inputs are needed. For stand-alone MoodyCore, a no-input BC is equivalent to mode=`fixed`, but a quadratic interpolation between a start and end point can be set using the following fields, in combination with `bc.startTime` and `bc.endTime`:

`startValue` (vertexLocation alt. [0,0,0]) [m alt. N]
The start value defines the value of the boundary at the beginning of the simulation.

`endValue` (vertexLocation alt. [0,0,0]) [m alt. N]
The end value defines the value of the boundary at the `endTime`.

`startVelocity` ([0,0,0]) [m/s alt. N/s]
The initial rate of change of the boundary at `startTime`.

`mode=externalRigidBody`
This mode expects a 6 dof position state of a rigid body as input to the API. The total mooring forces and moments on the rigid body frame are returned to the external solver.

`slaves`
A list of vertices and locations that specifies slave vertices. Format is the same as slaves for rigidBodies and hydroBodies. Slave positions should be given in the body local coordinate system.

2.7.2 Rotational modes

`rotationMode=pinned`
Models a moment-free point connection.

`rotationMode=moment`
Models a constant moment vector BC condition.

`rotValue` (0,0,0) [Nm]
Specify moment vector amplitude.

`rotationMode=clamped`

`rotValue` (0,0,1) [-]
Direction vector for clamp. The axial direction of attached cables in the global coordinate system. Does not have to be unit-normalized, but needs non-zero norm.

2.8 Cable types

LIMITATION: MoodyCore neglect the torsional rigidity of the cable. If torsion is important to you then MoodyCore is not the correct tool.

A cable type is used to define the material properties of a cable. Each cable refers to a cable type. It is analogous to cutting a nylon rope into three pieces. The nylon rope properties are contained in the cable type, and the three pieces become three cables.

Cable types are grouped for each type under keywords `cableTypeX`. The required and optional fields of the cable type structure are described below.

`diameter` [m]
: The diameter of the cable. If not specified, the default value is the equivalent diameter, computed from `gamma0` and `rho`, assuming a constant density in a circular cross-section. After initial dependencies, diameter is only used as drag diameter base and as contact to the ground.

`gamma0` [kg/m]
: The dry mass per meter of the cable. If not specified, the default value is computed from `diameter` and `rho`, assuming a constant density in a circular cross-section.

`rho` [kg/m^3]
: The density of the cable material. If not specified, the default value is computed from `diameter` and `rho`, assuming a constant density in a circular cross-section.

`materialModel`
: The material model is a substructure of the cable type. It contains information about the constitutive relations of the cable. See section 2.9 for choices and detailed info.

`CDn` (0)
: The drag force coefficient in the normal direction of the cable. Is applied to the diameter.

`CDt` (0)
: The drag force coefficient in the tangential direction of the cable. Is applied to the diameter.

`CM` or `CMn` (0)
: The added mass coefficient of the cable in the normal direction of the cable. Is applied to the cross-sectional area.

`CMt` (0)
: The added mass coefficient in the tangential direction of the cable.

```
cableType1={
    diameter = 0.05;
    gamma0 = 1.5; % (default: rho*area)
    rho = 7800; % (default: gamma0/area)
    materialModel = {
        type = 'bilinear';
        EA = 1e4;
        xi = 10;
    }
    CDn = 1; % (default: 0)
    CDt = 0.1; % (default: 0)
    CM = 2; % (default: 0)
}
```

2.9 Material models

The material model is a substructure of the `cableType` and contains the information about the constitutive relation of the cable. Moody uses the cable elongation ($\delta L/L_0 - 1$) as strain input to the force-strain relation. Several non-linear responses are implemented for the axial force, but bending stiffness is always linear.

`type`
: The type name. Several types exist. See individual keywords further down.

`EA` ($E * A$) [N]
: The mean axial stiffness of the cable. In nonlinear materials, this value is used as an approximation when computing the initial condition of the cable analytically. If not present, it is computed from E input and cable area.

`xi` (0) [Ns]
: The linear internal damping coefficient of the material. The internal damping force is added to the tension force from the material type and is computed as a linear strain-rate dependence as $T_\xi = \xi\dot{\epsilon}$.

`EI` (0) [kNm^2]
: The bending stiffness of the cable.

`compressionScale` (0)
: Scale factor for the compression stiffness. Negative tension is computed as compressionScale*EA*strain. Needs to be set explicitly to model beams with high bending stiffness. Reduce value if buckling occurs.

`type=bilinear`
: Computes the axial tension force of the cable according to Hooke's Law, but disallows compressive forces. Thus $T = \max(EA\,\epsilon, 0)$.

`type=exponential`
: This type uses an exponential strain-force relation as:

$$T = \max\left(0, K\ \left(e^{a\epsilon} - 1\right)\right)\ , \tag{2.6}$$

where K and a are defined in additional fields.

`K` [N]
: Basic stiffness.

`a`
: The growth-rate of the exponential function.

`type=polynomial`
: Implements strain-force relation as a polynomial according to

$$T = \max\left(0, \sum_{i=1}^{m} C_i \epsilon^i\right)\ , \tag{2.7}$$

where C_i is the polynomial coefficient of order i as defined by the `C` field below. Additional fields of input are:

`C` [N]
A vector set of coefficients. First value is the linear coefficient, second is the quadratic strain dependence and so on. The length of the vector specifies the order of the polynomial strain-force curve. Ex:

```
materialModel.C = [1.0e3 2.0e3 3.0e3 4.0e3];
```

2.10 CABLES

Cables are defined with naming convention following `cable1`, `cable2`, etc. They are used to define the cable objects in the input file.

`typeNumber`
: The cable type number.

`vertex0`
: The start vertex number of the cable. This will be used as $s = 0$, where s is the unstretched cable coordinate from 0 to L.

`vertex1`
: The end vertex number of the cable. This will be used as $s = L$, where s is the unstretched cable coordinate from 0 to L.

`length` [m]
: The unstretched length (L) of the cable. For some types of initial condition, e.g. `PreStrain`, the field is unused and is overwritten by the computed value.

`IC`
: The `IC` field is a substructure that defines the initial conditions of the cable. There are several options for the type of initial conditions and they are described in section 2.11.

`mesh.N`
: The number of elements in the cable.

`mesh.P` (4)
: The polynomial order of the Legendre polynomial basis functions. In this release of Moody, this option is not available. A constant P=4 is always used.

```
cable1 = {
    typeNumber = 1;
    vertex0 = 1;
    vertex1 = 2;
    length  = 100;
    IC = {
        type = 'PreStrain';
        eps0 = 0.05; % prestrain
    }
    mesh.N = 10; % no. of elements
}
```

2.11 Initial Conditions

The initial conditions are specified individually for each cable object, as a sub-structure of `cable`.

`type`
Choose the type. Each type has its own additional input fields, described below.

`type=preStrain`
The cable is set to a straight line between the two end points. The length of the cable is recomputed based on the distance between the points and the specified initial pre-strain. Gravity is not taken into account..

`eps0`
The initial pre-strain of the cable. Input is dimensionless and can be either a single value or a vector of values. If more than one value is specified, the length of the vector must match the length of `parts`, see below. Each part of the cable will then be given the matching pre-strain.

`parts` (1.0)
The relative part of the cable subject to the strain specified in `eps0`. The input is a vector of the same size as `eps0` and the sum of all the values must be 1. The values of `parts` relates to the unstretched length fractions of the cable.

```
IC = {
    type = 'PreStrain'
    eps0 = [0.01; 0.02]; % different prestrains
    parts = [0.5,0.5]; % corresponding sections
}
```

`type=straightLine`
Similar to prestrain, but uses cable length by default. This option should be used to avoid re-definition of cable length during static analysis.

`eps0`
The pre-strain of the cable, overrides length input if present.

`type=catenary`
The cable is set to be an elastic catenary shape at rest. This option only works when gravity is non-zero. Ground interaction is detected analytically, so the cable is initially set to lie still exactly at the ground level. No further input is required.

`type=sine`

Sine type is an extension of type `preStrain`. A sinusoidally varying displacement is imposed on a pre-strained cable. The sinusoidal displacement is in the vertical direction only.

`amplitude` (0) [m]
: The amplitude of vertical displacement. When set to 0, the result is the same as for type `'preStrain'`.

`periods` (0.5)
: The number of sinusoidal periods to use along the cable. Default is a single bubble displacement, i.e. 0.5 periods.

`eps0` (0.0)
: The desired pre-strain in the cable before the sinusoidal displacement. The input is a single value that is applied to the whole cable.

```
IC = {
    type = 'Sine';
    amplitude = 1; % [m]
    eps0 = 0.01;
    periods = 0.5;
}
```

2.12 COMPONENTS

As of Moody 2.1, axial springs and dampers can be defined. They are defined by keyword `component1`, `component2`, etc.

`vertex0`
Start vertex.

`vertex1`
End vertex.

`type=spring`
Models a massless spring-damper between two points. Force is constant along the spring and is reported in outward-pointing normal direction at its ends. It expects a vector of input variables interpreted as polynomial coefficients. See the theory manual for more information. A self-explaining example is provided below.

`stiffness` [0] (N/m^i)
Vector of stiffness. Each entry corresponds to a polynomial order. [a] is in N/m, [a,b] is in [N/m, N/m^2] etc.$F_c = \sum_{i=1}^{N} C_i(L - L_0)^i$, with N being length of stiffness input and L_0 the restlength.

`damping` [0] (Ns^i/m^i)
Vector of damping. Each entry corresponds to a polynomial order. [a] is in Ns/m, [a,b] is in [Ns/m, Ns^2/m^2] etc. $F_d = \sum_{i=1}^{N} D_i v^i$, with N as the length of damping input, and v as the axial velocity.

`restLength` (0) [m]
Restlength of the component, at which point it gives no stiffness force.

`constantForce` (0) [N]
Constant force offset applied by the component. Positive values are tension forces, negative are compressive forces.

`allowCompression` (0) [-]
Switch to allow compressive forces or not in the component.

```
component1 = {
    type = "spring"
    vertex0 = 2; % start vertex
    vertex1 = 4; % end vertex
    stiffness = [1e4];  % Linear stiffness (N/m)
    damping = [26.341]; % Linear damping (Ns/m)
    restLength = 0;    %  Rest length (m)
    allowCompression = 0; % Switch on compression
    }
```

`type=tabulated`

A two-dimensional lookup table controls the force as a function of length (x) and extensional velocity (v). Linear two-factor interpolation is used to compute F(x,v) for each length and velocity combination. Extrapolation is discouraged. Please ensure that values are within the table limits. The example below explains the input table format. It describes a constant 5N spring with 0 force at 0 length, with a 10N constant damper at 10m/s extension, not activated in compression.

`table`

Expects an N+1 by M+1 matrix. First row contains M component lenghts, first column contains N component extensional velocities (m/s), and the remaining matrix contains the force values at the given length and velocity coordinates. Force input is in [N].

`file`

Filename of table input, used instead of `table` for larger tables. No comments are allowed in external file input style. Should be data only.

```
component2={
    vertex0 = 1;
    vertex1 = 5;
    table = [ 0 0 10 15 20; % row1: x (m)
             -10 0 5 5 5;  % col1: v (m/s).
              0  0 5 5 5;  % rest is force (N)
              10 10 15 15 15];
}
```

2.13 Using source inheritance

MoodyCore has a special keyword `source` which can be used in any input object to link input data to a new object, or to a user-defined data label. This is used frequently in the tutorials to copy data from one cable or body to another. All data that is NOT specified in the object containing `source` will be added in the input file parsing stage. See example from tutorial 5 below.

```
%--- Rigid bodies ---%
rigidBody1 = {
    type = point
    mass = 2000
    V      = 1
    vertex = 2
}
rigidBody3={
    source = rigidBody1
    vertex = 4
}
%--- Cables ---%
cable1={
    type = 1
    vertex0 = 1
    vertex1 = 2
    IC = {
        type = 'prestrain'
        eps0 = 0.2
    }
    mesh.N = 3
}
cable2={
    source = cable1
    vertex0 = 2
    vertex1 = 3
}
cable3={
    source = cable1
    vertex0 = 3
    vertex1 = 4
}
```

3

Running the code

3.1 USER INPUT

MoodyCore is operated though the terminal window. Generally, the `-<flagName> <value>` syntax is used to provide run-time information to the computation. All settings apart from the output file name prepends information in the input file.

3.1.1 Flags

`-f` (moodyInput.m)
: Filename. Must be followed by the filename of the input file.

`-o` (input filename without extension)
: Output filename. Must be followed by the name of the output folder.

`-time.start`
: Specifies the start time of the simulation. Overrides information specified in input file.

`-time.end`
: Specifies the end time of the simulation. Overrides information specified in input file.

`-startState`
: This is the reboot flag of MoodyCore. If followed by a value, the value should be an existing results folder name. If no value is specified, the default is to use the output folder if it exists. MoodyCore will start from the mooring state at the start time of the simulation. If no results are found, MoodyCore will start from static equilibrium, and print a warning message.

`-statics`
This is a shorthand flag for solving only the static solution of the system. It automatically sets the end time to the start time and enables the statics.solve stage of the simulation.

`-addInput`
A way to specify additional changes to the input file as value pairs: `-addInput <name1> <value1> <name2> <value2>`. All information prepends the input file and thereby overrides the information in the input file.

3.1.2 Examples

```
moodyCore.exe -f caseFile.m
moodyCore.exe -f moodyInput.m -o outFolder -time.start 10 \
-time.end 20  -addInput simulation.time.dt 0.01
```

4

Pre processing

4.1 Mesh manipulation

The `moodyPre.exe` utility has a direct link to the mesh manipulation functions in MoodyCore. Using the `-meshConvert` flag enables reading/writing from/to mesh formats of .stl, .gdf (WAMIT format) and .dat format (Nemoh format). Symmetric mesh files are supported as input, however they are mirrored and printed without symmetry flag in the output. Two additional flags are available:

`-origin <x> <y> <z>`
When present, expects three following input values specifying desired origin coordinate in the output mesh. Naturally, it has to be given in the input mesh coordinate system. This option translates the mesh by -[x,y,z] m and prints it with the new origin.

`-wet <swl>`
When specified, the `-wet` flag keeps only wet panels in the output mesh. `<swl>` input is a single value specifying the position of the still water level.

> NOTE: The `<swl>` is given relative to the origin of the output mesh.

Usage:

```
moodyPre.exe -meshConvert in.gdf out.stl -
moodyPre.exe -meshConvert in.stl outMesh.dat -origin 0 0 2 -wet 0
```

4.2 NEMOH CASE PREPARATION

MoodyMarine has an input form interface for building a Nemoh project case from pure MoodyCore input. This is done by running `moodyPre.exe` with the -nemoh flag. It is provided as a stage 0 execution, prior to the preProc.exe, solver.exe and postProc.exe executions of Nemoh (v3).

The bem input is stored in two different places of the input file. The top key `bem` contains the overall settings of the simulation (no of frequencies, directions etc.), while each hydrodynamic body has its own input section for the boundary element method (BEM) stage, with keyword `hydroBodyX.bemInput`. In addition, `moodyPre.exe` reads required environmental settings from the normal moody input format. The input format is described below. Usage:

```
moodyPre.exe -nemoh -f moodyInput.m
```

4.2.1 *Input file format*

Nemoh case generation is integrated into the settings of the standard MoodyCore input file.

`environment`
The following Nemoh input values are used in order of priority. (X) indicates default value:

- density (1000) : `waterDensity`
- gravity (9.81) : -1*`gravity`
- depth (1000) : `wave.depth`, `waterLevel`-`ground.level`

`bem`
The main input setting of the Moody prepropressor interface.

`body`
Body id (int) to include in analysis.

> **LIMITATION:** MoodyCore only supports single bodies IRF simulations at present.

`output`
The output folder of the results. It is created if it does not exist. Only last folder level is created.

`w0` (0.01) [rad/s]
Start frequency.

`w1` (20) [rad/s]
End frequency.

`nFreqs` (30) [-]
Number of wave frequencies in analysis.

`dir0` (0) [deg]
Starting wave direction.

`dir1` (180) [deg]
End wave direction.

`nDirs` (5) [-]
Number of wave directions in analysis.

`hydroBodyX.bemInput`
Body specific data is nested in the substructure bemInput within each hydroBody. The chosen id is controlled by the `bem.body` input (required).

`meshName`
Name of mesh file to use in analysis. .stl, .gdf and .dat (Nemoh mesh) formats are supported. moodyPre.exe is used to translate mesh according to Nemoh coordinate requirements ($z = 0$ at the waterline).

`keepMesh` (0)
Switch that when 1 turns OFF any MoodyCore manipulation of the mesh, assuming the meshName is a well-prepared Nemoh mesh file for the analysis.

`cogInMesh` (0,0,0) [m]
Specify point of CoG in the raw input mesh coordinate system. Unused if `keepMesh=1`.

`position` (0,0,0) [m]
The position of the CoG in the global coordinate system. For best results, this should be equivalent to the static equilibrium position of the body (when all external loads are accounted for). This will appear as (x,y,-z) in the Nemoh.cal file as the analysis point for all radiation and diffraction problems.

4.3 Wave elevation

The wave elevation is not printed together with the results by default in Moody. It can be reprocessed through the `-waveElevation` flag sent to `moodyPre.exe`. Two input formats are supported:

`-domain <X> <Y> <nx> <ny>` An `X*Y` area surface grid with `nx` by `ny` panels is generated. Wave elevation reported at all vertices of the surface. The domain setting is saved in `waveDomain.dat`. The domain is centered around (0,0,0).

-xyFile <filename> File input format of wave probe locations. Expects file format ascii with N lines of 2 columns, specifying the [x y] coordinates of the probe. Wave elevation is computed at each probe.

-positions <x1> <y1> <x2> <y2> ... Output wave elevation at a given list of probe locations. Wave elevation is computed at each probe.

-o <outputDirectory> The output is written to `outputDirectory/` as `waveElevation.dat` and (for `-domain` flag only) as `waveDomain.dat`. Default is to output in the result directory of the input file, i.e. `fileName` (without extension).

4.3.1 Handling output time

The wave elevation feature was developed to provide wave elevation animation data to MoodyMarine. If `time.dat` is present in the same directory as the input file location, then `waveElevation.dat` will provide elevation at each probe at each timestamp of `time.dat`. If it is not present, output time is governed by input file `simulation.time` and printed accordingly.

4.3.2 Output format

At present `waveElevation.dat` is printed in ascii format. One line per time stamp, and each line has Nprobe+1 entries. First column is the time (s).

5

Post processing

5.1 MOODY POST

MoodyCore has a post-processing utility named `moodyPost.exe`. Although nodal values are printed to the output directory, `moodyPost.exe` can be used to create a smaller set of output data for post-processing. It can also be used to generate VTK-files of the cable lines and component and their tension force magnitude for visualisation in e.g. Kitware's Paraview.

5.1.1 Usage

`moodyPost.exe` is a command line tool. The first parameter must be a moody result directory. Default is to process all times in output/time.dat. To control which times to process and output there are three flag options.

`-times`
A list of output times.
Ex. `moodyPost.exe outName -vtk -times '0,1,2,4.5'`

`-timeList`
Use times from a file. File should be in a single column vector format with no header.
Ex. `moodyPost.exe outName -vtk -timeList timeFile.txt`

`-dt` [s]
Use time step size `dt` to step through the output history.
Ex. `moodyPost.exe outName -vtk -dt 0.5`

Output options are:

`-vtk`
Print VTK files in sub-directory `VTK`.
Ex. `moodyPost.exe moodyResults -vtk -times '0,1,2,4.5'`

`-p`
Print nodal values in sub-directory `processed`. Optional flag `-var` is used to control which parameters to output. Default is to use all.

`-var` Should be followed by a list of integer values in 1 to 7, where

1. tension
2. strain
3. strain rate
4. position
5. velocity
6. tangent
7. end point values.

Ex. `moodyPost.exe moodyResults -p -var '[1,4]'`, prints tension and position.

`-clean`
The clean flag (or `-c`) is used to repair incomplete output. So, for cases that have crashed or aborted, `moodyPost.exe` makes the output causal in time and prints the `sPlot.dat` file required by the post-processing routines in Matlab. It also truncates results to the last completed time stamp.

> **NOTE:** It is strongly advised to backup the simulation results prior to executing `moodyPost.exe -clean`, at least for time-consuming simulations, as the command will modify the results in-place.

5.2 MATLAB ROUTINES

A set of Matlab routines and functions are provided in `$moodyDir/API/matlab`. These can be used to load and visualise the results. Each cable result is analysed separately. The most important routine is `readCase.m` which returns a cell array of data structures containing the data of all cable objects and rigid bodies. Please note that readCase assumes there exists a `sPlot.dat` file. If moody quit unexpectedly or for any other reason stopped, the `sPlot.dat` may be missing. In API runs this also may mean that the time order is non-causal (due to multiple calls of the same time-window). Prior to loading, MoodyCore output then needs to be cleaned using: `moodyPost.exe` with the `-clean` option. See the help texts in each function for usage instructions.

5.3 PYTHON

Two Python scripts are included in API/python/post for those who want to view moody results through Python. `moodyPlt.py` presents examples of plotting information, and is based on `moodyReader.py` which imports the data. These files are released for convenience; they are not a fully documented API or post-suite.

5.3.1 Dependencies

- numpy
- matplotlib.pyplot

5.4 OUTPUT FILE STRUCTURE

The result time stamps are logged in `time.dat` (ascii). The first column in each data file contains the same time data. Hence all data files have the same number of rows. The file `info.json` shows the number of objects of each type in the simulation, as well as the number of columns in each file.

5.4.1 Cable output

Nodal values are presented along the unstretched cable coordinate s for each n output quadrature points $[s_1, s_2, \ldots, s_n]$ of the cable. Output is separated into vector valued output and scalar output. For scalars such as tension, strain and strain rate, the output goes from the start to the end of the cable. For the vector valued output, such as position and velocity, the data is stored consecutively for each coordinate direction of the simulation, in the order x, y, z. The structure is described in table 5.1.

5.4.2 Rigid body output

Rigid bodies are calculated directly in the physical domain. The state vector is printed at the center of gravity acc. to Table 5.2.

5.4.3 Hydrobody output

The main output file of the body state has the same format as the rigid body output. In the case of hydrobody `linearIRF` type however, the four quaternion columns instead indicate the roll, pitch and yaw angle (radians) of the body. The fourth column is a dummy output (0). Additionally a `hydroBodyX_forces.dat` file output is available. It lists the force components acting on the hydroBody. The format is displayed in Table 5.3.

Table 5.1: Output structure description for cable object data. Times are indexed as $t_1, t_2, \dots, t_m$, vector components as $\overline{f} = [f_x, f_y, f_z]$ and the cable unstretched coordinate s is indexed as $s_1, s_2, \dots, s_n$. The moment file only exists if bending stiffness is used.

File suffix	Output structure
_position.dat	$t_1 \quad x(t_1, s_1 : s_n) \quad y(t_1, s_1 : s_n) \quad z(t_1, s_1 : s_n)$ $\vdots \quad \vdots \quad \vdots \quad \vdots$ $t_m \quad x(t_m, s_1 : s_n) \quad y(t_m, s_1 : s_n) \quad z(t_m, s_1 : s_n)$
_velocity.dat	$t_1 \quad v_x(t_1, s_1 : s_n) \quad v_y(t_1, s_1 : s_n) \quad v_z(t_1, s_1 : s_n)$ $\vdots \quad \vdots \quad \vdots \quad \vdots$ $t_m \quad v_x(t_m, s_1 : s_n) \quad v_y(t_m, s_1 : s_n) \quad v_z(t_m, s_1 : s_n)$
_tension.dat	$t_1 \quad T_x(t_1, s_1 : s_n) \quad T_y(t_1, s_1 : s_n) \quad T_z(t_1, s_1 : s_n)$ $\vdots \quad \vdots$ $t_m \quad T_x(t_m, s_1 : s_n) \quad T_y(t_m, s_1 : s_n) \quad T_z(t_m, s_1 : s_n)$
_moment.dat	$t_1 \quad M_x(t_1, s_1 : s_n) \quad M_y(t_1, s_1 : s_n) \quad M_z(t_1, s_1 : s_n)$ $\vdots \quad \vdots$ $t_m \quad M_x(t_m, s_1 : s_n) \quad M_y(t_m, s_1 : s_n) \quad M_z(t_m, s_1 : s_n)$
_strain.dat	$t_1 \quad \epsilon(t_1, s_1 : s_n)$ $\vdots \quad \vdots$ $t_m \quad \epsilon(t_m, s_1 : s_n)$
_sPlot.dat	$t_1 \quad s_1 : s_n$

Table 5.2: Output structure of rigidBodyX.dat. The 14 data columns are time (1), position (3), quaternion (4), velocity (3) and angular velocity (3). The velocity and angular velocity are in the body local coordinate system.

t_1	$\vec{p}(t_1)$	$\vec{q}(t_1)$	$\vec{v}(t_1)$	$\vec{w}(t_1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_m	$\vec{p}(t_m)$	$\vec{q}(t_m)$	$\vec{v}(t_m)$	$\vec{w}(t_m)$

Table 5.3: Output structure of hydroBodyX_forces.dat. The 37 data columns are: time (1), excitation (or diffraction) force (6), radiation force (6), added mass force (6), restoring force (6), quadratic drag force (6), and linear drag force (6). The drag forces are in the body local coordinate system.

t_1	$\vec{F}_{exc}(t_1)$	$\vec{F}_{rad}(t_1)$	$\vec{F}_{add}(t_1)$	$\vec{F}_{res}(t_1)$	$\vec{F}_{quad}(t_1)$	$\vec{F}_{lin}(t_1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_m	$\vec{F}_{exc}(t_m)$	$\vec{F}_{rad}(t_m)$	$\vec{F}_{add}(t_m)$	$\vec{F}_{res}(t_m)$	$\vec{F}_{quad}(t_m)$	$\vec{F}_{lin}(t_m)$

Table 5.4: Output structure of componentX.dat. The file is stored in ascii format.

t_1	$\vec{p}_0(t_1)$	$\vec{p}_1(t_1)$	$\vec{v}_0(t_1)$	$\vec{v}_1(t_1)$	$\vec{F}_1(t_1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_m	$\vec{p}_0(t_m)$	$\vec{p}_1(t_m)$	$\vec{v}_0(t_m)$	$\vec{v}_1(t_m)$	$\vec{F}_1(t_m)$

5.4.4 Component output

The component output file (componentX.dat) is presently only implemented in ascii format. It stores the positions, $\vec{p}$ and velocities $\vec{v}$ at the end points 0 and 1 respectively. It also prints the force vector at `vertex1` (outward-pointing tangential direction of component. See Table 5.4.

6

API documentation

6.1 Introduction

MoodyCore as a mooring module can be used as an add-on for external solvers of the the fluid problem. We call this usage API-mode. When MoodyCore is started up in API-mode, an external solver is guiding the time evolution of some of the boundary conditions. In this way MoodyCore can be used as a mooring module for hydrodynamic simulations of wave-body interaction. The interaction between the codes follows the schematics of Figure 6.1.

When the time step size of the external solver is larger than the internal time step of Moody, a sub-step is used internally in MoodyCore. MoodyCore then interpolates in time between the old and the new boundary values received. Quadratic interpolation with optional time staggering is used. See the theory sections A.8 for a description of the interpolation. A discussion on how different choices of interpolation can affect the solution can be found in [4]. We recommend to use `staggeredTimeFraction =0.5`, which gives good and stable results for most applications, provided that a reasonably high sample rate is used.

6.2 Interface functions

The program interface is made up of global functions:

(i) `moodyInit` - initialisation routine

(ii) `moodySolve` - compute and return the mooring forces

(iii) `moodyClose` - close the moody objects and clean the output files

(iv) `moodyGetNumberOfPoints` - return the number of quad-points in the mooring system.

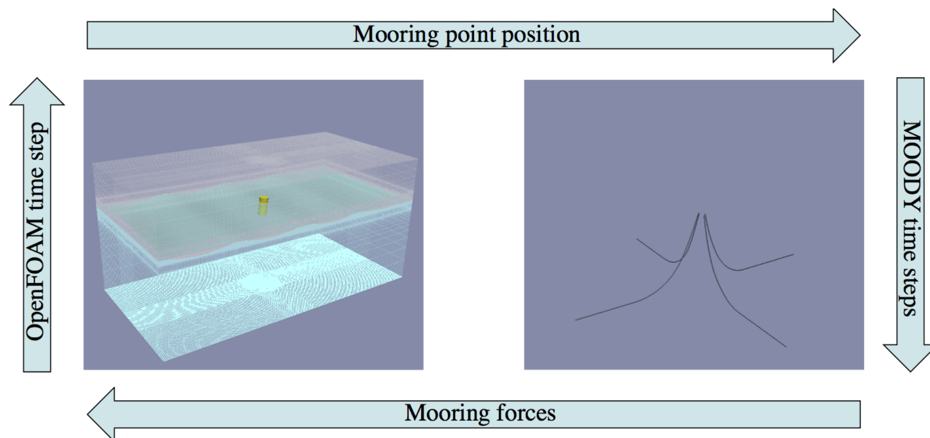

Figure 6.1: Schematic description of the information loop between Moody and an external software, represented by an OpenFOAM CFD simulation.

(v) `moodyGetPositions` - return the coordinates of all quad-points in the mooring system.

(vi) `moodySetFlow` - sets the flow velocity, acceleration and density at each of the quadpoints of Moody. Used for one-way coupling of high-fidelity fluid simulations.

6.2.1 Description

The interface is contained in `$moodyDir/include/moodyWrapper.h`, together with a description of all interface parameters required. The following is an excerpt from `moodyWrapper.h`, showing the definition of the interface functions.

```
 /** Initialisation call to moody. Required to setup the API.
   Parameter fName:          input file name (including path and extension)
   Parameter nVals:          number of values in parameter initialValues
   Parameter initialValues: array of initial values of the boundary conditions (
    global frame)
   Parameter startTime:      time at start of simulation (s)
*/
void moodyInit(const char* fName, int nVals,double initialValues[], double
    startTime );

/** Solves the internal system of mooring dynamics between time t1 and t2.
    Parameter X:     Boundary condition values at time t2.
    Parameter F:     Returned as the outward pointing mooring forces for each
        boundary condition dof.
    Parameter t1:    Time of last call, start time of time integration.
    Parameter t2:    End time of present time step simulation.

    Note: t1 and t2 are stored internally, with corresponding states of the
        mooring dynamics. If t1 has increased since the last call, moody will
        move forward in time and store the state at t1 as the new starting state
        of the mooring dynamics. Several iterations can be made with varying t2
        but the same t1 without loosing the backup starting state. Compatible
        with predictor/corrector type time stepping schemes.
```

```
*/
void moodySolve(const double X[], double F[], double t1, double t2);

/** Closes the moody simulation and stores data for post processing */
void moodyClose();

/** Interface function to collect mooring points from moody. Returns the number
    of points. XYZ is 3*nPoints. */
int moodyGetNumberOfPoints();

/** Collect mooring quadrature points in xyz list. xyz needs to be a container of
     at least nPointsIn*3 doubles. */
void moodyGetPositions(int nPointsIn, double xyz[] );

/** Set flow velocity (vF) acceleration (aF) and density (rhoF) at time t in all
    nPoints of the mooring system. nPoints should be output from
    moodyGetNumberOfPoints() and containers vF, and aF must be at least nPoints*3
     in size. rhoF must be at least nPoints. */
void moodySetFlow(const double t, int nPoints, const double vF[] ,const double aF
    [],const double rhoF[]);
```

The input file for an API simulation requires some additions for the code coupling to work.

6.3 Input file entries

Two modes of boundary condition are supported for control via the API keywords: `externalPoint` and `externalRigidBody`. See section 2.7 for information on their individual setup. Other input file entries for the API are grouped under the `simulation.API` keyword. The following fields are available to control the simulation behaviour, where some are simply input-file versions of the command-line flags described in chapter 3:

`bcNames`

This is the only required field of the simulation.API input structure. It lists the boundary conditions that are to be externally controlled. The order of the input tells MoodyCore how to interpret the input in the `initialValues` parameter of `moodyInit` and in the `X` parameter in `moodySolve`. By extension it also controls the order of the forces returned in parameter `F` in `moodySolve`.
Ex: `API.bcNames = {'bc4'; 'bc5'; 'bc1'};`

`staggerTimeFraction` (0.0)

Used to control the API time staggering for a smoother mooring behaviour. Value α should be a scalar $\alpha \in [0, 1]$, specifying a fraction of the latest time step at which to stop the mooring simulation and return the mooring force. Default is $\alpha = 0.0$, but recommended value is $\alpha = 0.5$. End time of each coupling step is then computed from $t_{\text{end}} = (1 - \alpha)t_2 + \alpha t_1$.
Ex: `API.staggerTimeFraction = 0.5`

`output` (input file name)
The name of the MoodyCore output directory of the simulation. Equivalent to the `-o` flag, but read from input file in API mode.
Ex: `API.output = 'moodyResults'`

`reboot` (yes)
This flag controls the startup behaviour of MoodyCore in API mode. Default value is `'yes'` and tells MoodyCore to attempt a reboot from a dynamic state at `startTime` of `moodyInit` call in the output directory. New output is then appended to the previous results. To disable use of old results use `API.reboot='no'`. If API.reboot takes any other string value, this should be a result directory that can be used as a source state for the simulation. The source directory is then unchanged by the simulation.
Ex: `API.reboot = 'no'`

`syncOutput` (1)
Integer value 0 or 1 allowed. If 1, MoodyCore results are printed at each coupling time (in addition to at each `saveInterval`). If 0, output is only printed based on Moodys' internal `print.dt`.
Ex: `API.syncOutput = 1`

6.4 Fortran, Matlab and Python interface functions

There is a Fortran module with interface functions to Moody, to facilitate coupling to Fortran codes. There is also a Matlab API which allows for easy prototyping of coupled simulations from Matlab. An API to python, which is mostly for post processing at the present time. All interfaces are located in the `API/<language>` location.

6.5 Testing the API

Finally, there is a `testAPI.x` program for testing a MoodyCore API setup. The following example simulates the `mooringSystem.m` run via the time series specified in the `positions.txt` file. The time series should be in simple text-file format with no header line. The first column specifies the time, and the remaining columns specify the values of each api input dof respectively. See the Matlab API tutorial for an example of `positions.txt`. The utility supports an optional flag `-startTime` expecting a value in seconds at which to start the new simulation. Please note that if `API.reboot='yes'` is set in the input file, MoodyCore will attempt to reload and dynamically start from the previous results (as named by

API.output in the input file) at the specified -startTime. Default is to start at the first time of `positions.txt`.

```
testAPI.exe mooringSystem.m positions.txt -startTime 5
```

As of version 3.0, `testAPI.exe` also works with input from a moody result folder (if from an API run). The `API_values.dat` file is of the same format as expected by `testAPI.exe`. A coupled simulation can therefore be re-simulated in uncoupled mode using new MoodyCore inputs by the following syntax:

```
testAPI.exe mooringSystem.m oldAPIResults/API_values.dat
```

REFERENCES

[1] B. Cockburn and C.W. Shu. The local discontinuous Galerkin method for time-dependent convection-dominated systems. *SIAM J. Numer. Anal.*, 35(6):2440–2463, 1998.

[2] W.E. Cummins. The impulse response function and ship motions. *Schiffstechnik*, 9:101–109, 1962.

[3] J.R. Morison, M.P. O'Brien, J.W. Johnson, and S.A. Schaaf. The force exerted by surface waves on piles. *Petroleum Transactions, Amer. Inst. Mining Engineers*, 186:149–154, 1950.

[4] J. Palm. *Mooring Dynamics for Wave Energy Applications*. PhD thesis, Chalmers University of Technology, 2017.

[5] J. Palm and C. Eskilsson. Influence of bending stiffness on snap loads in marine cables: A study using a high-order discontinuous Galerkin method. *Journal for Marine Science and Engineering*, 8(10):795, 2020.

[6] J. Palm and C. Eskilsson. Mooring systems with submerged buoys: influence of floater geometry and model fidelity. *Applied Ocean Research*, 102:102302, 2020.

[7] J. Palm and C. Eskilsson. On endstope. In *EWTEC*, September 2021.

[8] J. Palm, C. Eskilsson, and L. Bergdahl. An *hp*-adaptive discontinuous Galerkin method for modelling snap loads in mooring cables. *Ocean Engineering*, 144:266–276, 2017.

[9] A. Tjavaras. *The Dynamics of Highly Extensible Cables*. PhD thesis, Massachusetts Institute of Technology, 1996.

A

Theory manual

A.1 Background

Moody is built around an hp-adaptive cable solver based on the discontinuous Galerkin method, with an approximate Riemann solver of local Lax-Friedrich type, see.This manual outlines the governing equations and highlights the modelling approach used in the implementation. The section on cable dynamics is mostly compiled from Palm and Eskilsson [5] where bending stiffness was introduced in the modelling, while the rigid body section originates from Palm and Eskilsson [6] where submerged buoy dynamics was presented. In addition, the section on API boundary condition interpolation is collected from the PhD thesis of Palm [4].

A.2 Cable dynamics

The cable equation of motion is the balance between inertial, internal and external forces on the cable:

$$\frac{\mathrm{d}\vec{\nu}}{\mathrm{d}t} = \frac{\mathrm{d}\vec{T}}{\mathrm{d}s} + \vec{f}_{\mathrm{e}}\,, \tag{A.1}$$

in which $\vec{\nu} = \gamma_0 \vec{v}$ is the cable momentum per meter (mass per meter γ_0 times velocity $\vec{v}$), $\vec{T}$ is the internal tension force and $\vec{f}_{\mathrm{e}}$ represents the external forces.

We denote the axial cable strain (elongation) by ϵ with elongation factor $l_\epsilon = 1 + \epsilon$. The cable tension force can be divided into axial and transversal forces as:

$$\vec{T} = T\,(\epsilon, \dot{\epsilon})\,\hat{\imath} + \vec{T}_{\mathrm{s}}\,, \tag{A.2}$$

where T is the axial force magnitude and $\vec{T}_{\mathrm{s}}$ is the shear force vector. The axial tangent vector $\hat{\imath}$ and the strain ϵ can be written in terms of the cable position vector

$\vec{r}$ as:

$$\vec{q} = \frac{d\vec{r}}{ds}, \tag{A.3}$$

$$\epsilon = \sqrt{\vec{q} \cdot \vec{q}} - 1, \tag{A.4}$$

$$\hat{\imath} = \frac{\vec{q}}{l_\epsilon}. \tag{A.5}$$

We use $\vec{v}$ and $\vec{q}$ as independent variables in the inertial frame and use the time derivative of Equation (A.3) to arrive at an expression for $\dot{\vec{q}}$,

$$\frac{d\vec{q}}{dt} = \frac{d\vec{v}}{ds} = \frac{d}{ds}\left(\frac{\vec{v}}{\gamma_0}\right). \tag{A.6}$$

Equations (A.1) and (A.6) can be written in conservative form as:

$$\frac{d}{dt}\begin{bmatrix} \vec{q} \\ \vec{v} \end{bmatrix} = \frac{d}{ds}\begin{bmatrix} \vec{v}/\gamma_0 \\ \vec{T} \end{bmatrix} + \begin{bmatrix} \vec{0} \\ \vec{f}_{ex} \end{bmatrix}, \tag{A.7}$$

which transforms to:

$$\frac{d\vec{u}}{dt} = \frac{d\vec{F}(\vec{u})}{ds} + \vec{G}(\vec{u}), \tag{A.8}$$

in terms of a single state vector $\vec{u}^T = \begin{bmatrix} \vec{q} & \vec{v} \end{bmatrix}$; a flux vector $\vec{F}$; and a source term $\vec{G}$.

A.2.1 External Forces

The $\vec{f}_{ex}$ in Equation (A.1) represents the total body force on the cable segment. The force can be divided into:

$$\vec{f}_{ex} = \vec{f}_a + \vec{f}_b + \vec{f}_c + \vec{f}_d, \tag{A.9}$$

where $\vec{f}_a$ is the force from the added mass and the Froude–Krylov effect, $\vec{f}_b$ is the buoyancy force, $\vec{f}_c$ represents contact forces and $\vec{f}_d$ stands for the drag force.

Added mass $\vec{f}_a$: In the Morison equation [3], the added mass force on a slender body is assumed to be proportional to the relative acceleration between the body and the flow. Hence, the added mass acting on the cable cross-section is:

$$\vec{f}_a = \rho_f A \left(\vec{a}_{w\perp} + C_{m\imath}\vec{a}^*_\imath + C_{m\perp}\vec{a}^*_\perp\right), \tag{A.10}$$

where A is the cross-section area of the cable, ρ_f is the fluid density, $\vec{a}^* = \vec{a}_w - \dot{\vec{v}}$ is the relative acceleration of the fluid with respect to the cable, and subscripts $\imath$ and $\perp$ denote the tangential and perpendicular components of a vector quantity. $C_{m\imath}$

and $C_{\mathrm{m}\perp}$ represent the added mass coefficients in the tangential and perpendicular direction of the cable, respectively. Please note that there is no dependency of cable strain as we assume a volume-preserving material, in which the cable elongation factor l_ϵ is cancelled by the same decreasing factor on the cross-section area.

Buoyancy, $\vec{f}_\mathrm{b}$: The buoyancy term is the sum of the cable weight and the buoyant Archimedes force,

$$\vec{f}_\mathrm{b} = Ag\left(\rho_\mathrm{f} - \rho_\mathrm{c}\right)\hat{z}\,, \tag{A.11}$$

where $g = 9.81\mathrm{m/s}^2$ is the gravitational constant. Elongation does not affect buoyancy, due to the assumption of a volume preservation.

Contact forces, $\vec{f}_\mathrm{c}$: Contact forces refer to the contact between the cable and the ground. This is modelled as a vertical bilinear spring-damper system with dynamic friction in the horizontal direction. The vertical force is computed from the cable penetration depth $\delta = z_\mathrm{g} - r_{\hat{z}}$ and the vertical cable velocity $v_{\hat{z}}$ as:

$$\vec{f}_\mathrm{c}^{(z)} = \sqrt{l_\epsilon}\left(K_\mathrm{g} d\delta - \xi 2\sqrt{K_\mathrm{g} d\gamma_0} v_{\hat{z}}\right), \tag{A.12}$$

where K_g is the bulk modulus of the ground, z_g is the vertical coordinate of the ground and ξ is the damping coefficient ($\xi = 1$ indicates critical damping). The horizontal friction force is proportional to the horizontal velocity of the cable, $\vec{v}_{\hat{x}\hat{y}} = \vec{v} - v_{\hat{z}}\hat{z}$, up to a threshold speed v_μ beyond which the magnitude is constant. The dynamic friction force is computed from:

$$\vec{f}_\mathrm{c}^{(xy)} = \sqrt{l_\epsilon}\mu \tanh\left(\pi\frac{|\vec{v}_{\hat{x}\hat{y}}|}{v_\mu}\right)\frac{\vec{v}_{\hat{x}\hat{y}}}{|\vec{v}_{\hat{x}\hat{y}}|}\min\left(\vec{f}_\mathrm{b}^{(z)}, 0\right), \tag{A.13}$$

where μ is the fully developed dynamic friction coefficient.

Drag forces, $\vec{f}_\mathrm{d}$: The cable drag is also modelled as in the Morison equation [3]. Based on the relative velocity between the fluid and the cable, $\vec{v}^* = \vec{v}_\mathrm{f} - \vec{v}$, we compute it as:

$$\vec{f}_\mathrm{d} = 0.5 d\rho_\mathrm{f}\sqrt{l_\epsilon}\left(C_{\mathrm{d}\iota}\left|\vec{v}_\iota^*\right|\vec{v}_\iota^* + C_{\mathrm{d}\perp}\left|\vec{v}_\perp^*\right|\vec{v}_\perp^*\right), \tag{A.14}$$

with d being the nominal diameter of the cable and $C_{\mathrm{d}\iota}$ and $C_{\mathrm{d}\perp}$ being the drag coefficients in the tangential and perpendicular directions. The nominal diameter decreases by a factor $\sqrt{l_\epsilon}$ with increasing ϵ, while the cable length factor is by definition l_ϵ. In combination, the drag force therefore scales with $\sqrt{l_\epsilon}$ as the cable stretches.

A.2.2 Shear Force Modelling

We model the cable by adapting the local Lagrangian frame formulation described in [9] to the inertial frame of reference. The main assumptions are: (i) that there

are no distributed moments acting on the cable; (ii) that the cable cross-section is axisymmetric; and (iii) that the inertial effects of rotating the cross-section can be neglected. The balance of moments equation for a linearly visco-elastic material with bending stiffness EI, bending damping ξ_b, and torsional stiffness GI_p thus reads:

$$0 = \frac{\partial}{\partial s}\left(\frac{\vec{M}}{l_\epsilon^2}\right) + l_\epsilon \hat{\imath} \times \vec{T}_\mathrm{s}\,, \tag{A.15}$$

$$\vec{M} = GI_\mathrm{p}\Omega_1\hat{\imath} + \hat{\imath} \times (EI\kappa + \xi_\mathrm{b}\dot{\kappa})\,, \tag{A.16}$$

in which we define the curvature $\vec{\kappa} = \dfrac{\partial \hat{\imath}}{\partial s}$ and its time derivative $\dot{\kappa} = \dfrac{\partial}{\partial s}\left(\dfrac{\dot{\vec{q}} - \vec{q}\dot{\epsilon}}{l_\epsilon}\right)$. The bending moment vector is thus acting in the binormal direction. Taking the cross-product from the left on Equation (A.15) allows us to solve for the shear force as:

$$\vec{T}_\mathrm{s} = \frac{1}{l_\epsilon}\hat{\imath} \times \frac{\partial}{\partial s}\left(\frac{\vec{M}}{l_\epsilon^2}\right) = \frac{1}{l_\epsilon}\hat{\imath} \times \frac{\partial \vec{M}_*}{\partial s}\,, \tag{A.17}$$

where $\vec{M}_* = \dfrac{\vec{M}}{l_\epsilon^2}$ represents the moment in the stretched domain used for boundary conditions. Finally, we note that the torsional stiffness is the only $\hat{\imath}$ component in Equation (A.15), which leads to a simplistic model for the torsional curvature Ω_1:

$$\frac{\partial}{\partial s}\left(GI_\mathrm{p}\Omega_1\right) = 0\,. \tag{A.18}$$

This is a consequence of all three assumptions (i)–(iii) stated above (see [9] for further information); however, the most important factor is the assumption of a circular cross-section. The torsional moment is constant along the cable, and Ω_1 can be globally computed from the boundary conditions of the cable system at each time step. Torsion is not supported in the present version of Moody.

A.2.3 Tension-Strain Relations

To complete the set of equations, we need to define the scalar function $T(\epsilon, \dot{\epsilon})$ relating cable strain and its rate of change to the axial tension force. See the user manual for the definition of the different choices of material model available.

A.3 Finite Element Method

We use a Discontinuous Galerkin (DG) finite element formulation of Equation (A.8). The computational domain Ω is partitioned into N elements $\Omega^e \in [s_L^e, s_R^e]$ with size h^e. A function $x(s,t)$ is approximated to an arbitrary order P within Ω^e as:

$$x(s,t) \approx x^e(s,t) = \sum_{k=0}^{k=P} \phi_k(s)\tilde{x}_k^e(t)\,,$$

where $\phi_k(s)$ is the kth order trial function with expansion coefficient $\tilde{x}_k^e$. Legendre polynomials are used as test and trial functions in this study. Defining the inner product operator $()_{\Omega^e}$ as:

$$(a(s,t), b(s,t))_{\Omega^e} = \int_{\Omega^e} a(s,t)b(s,t)\mathrm{d}s\,,$$

the DG formulation expressed in strong form within Ω^e reads:

$$(\phi_l,\, \phi_m)_{\Omega^e}\, \dot{\tilde{\vec{u}}}^e = \left(\phi_l,\, \frac{\partial \phi_m}{\partial \xi}\right)_{\Omega^e} \tilde{\vec{F}}^e + \left[\widehat{\vec{F}} - \vec{F}^+\right]_{s_L^e}^{s_R^e} + (\phi_l,\, G)_{\Omega^e}\,. \tag{A.19}$$

The DG method uses a numerical flux (denoted with $\widehat{\cdot}$ in Equation (A.19)) to express the value of a quantity on an element boundary. In this paper, we use:

$$\widehat{\vec{F}} = \frac{1}{2}\left(\vec{F}^+ + \vec{F}^- + \lambda\left(n^-\vec{u}^+ + n^+\vec{u}^-\right)\right)\,, \tag{A.20}$$

$$\lambda = \sqrt{\frac{1}{\gamma_0}\frac{\partial T}{\partial \epsilon}}\,, \tag{A.21}$$

where n is the outward pointing unit normal, λ is the speed of sound in the cable and $\widehat{\vec{F}}$ is represented by the Lax–Friedrichs flux of $\vec{F}$. Superscripts $^+$ and $^-$ indicate if values are taken from the interior domain (i.e., from Ω^e in this case) or from the neighbouring element of the boundary (i.e., Ω^{e+1} or Ω^{e-1}, respectively). See [8] for details.

Bending stiffness is introduced via the shear force using a local DG (LDG) [1] approach to compute the derivatives in Equations (A.16) and (A.17). Three auxiliary variables are used in the LDG formulation for the modal shear force $\tilde{T}_\mathrm{s}$: (i) κ, the cable curvature, (ii) $\dot{\kappa}$, the curvature rate of change, and (iii) τ, the spatial derivative of the moment vector. The modal coefficients, $\tilde{\cdot}$, of these auxiliary variables are for each element found from:

$$(\phi_l,\, \phi_m)_{\Omega^e}\, \tilde{\vec{\kappa}}^e = \left(\phi_l,\, \frac{\partial \phi_m}{\partial \xi}\right)_{\Omega^e} \tilde{\hat{\imath}}^e + \left[\widehat{\hat{\imath}} - \hat{\imath}^+\right]_{s_L^e}^{s_R^e}\,, \tag{A.22}$$

$$(\phi_l,\, \phi_m)_{\Omega^e}\, \tilde{\dot{\vec{\kappa}}}^e = \left(\phi_l,\, \frac{\partial \phi_m}{\partial \xi}\right)_{\Omega^e} \tilde{\dot{\hat{\imath}}}^e + \left[\widehat{\dot{\hat{\imath}}} - \dot{\hat{\imath}}^+\right]_{s_L^e}^{s_R^e}\,, \tag{A.23}$$

$$(\phi_l,\, \phi_m)_{\Omega^e}\, \tilde{\vec{\tau}}^e = \left(\phi_l,\, \frac{\partial \phi_m}{\partial \xi}\right)_{\Omega^e} \tilde{\vec{M}}_*^e + \left[\widehat{\vec{M}}_* - \vec{M}_*^+\right]_{s_L^e}^{s_R^e}\,, \tag{A.24}$$

remembering that $\vec{M}_*$ is the stretched domain moment from Equation (A.17). $\vec{T}_\mathrm{s}$ is then recovered from $\vec{T}_\mathrm{s} = \frac{1}{l_\epsilon}\hat{\imath} \times \vec{\tau}$. The numerical fluxes are chosen as:

$$\widehat{\hat{\imath}} = \frac{1}{2}\left(\hat{\imath}^+ + \hat{\imath}^-\right) + \beta\left(\hat{\imath}^+ n^- + \hat{\imath}^- n^+\right)\,, \tag{A.25}$$

$$\widehat{\vec{M}}_* = \frac{1}{2}\left(\vec{M}_*^+ + \vec{M}_*^-\right) - \beta\left(\vec{M}_*^+ n^- + \vec{M}_*^- n^+\right)\,, \tag{A.26}$$

Table A.1: Table of fluxes required for different typical connections used in a mooring system. Index *BC* indicates a prescribed value at the boundary, and "+" indicates a value taken from the internal domain. $\vec{v}^{(}_{\mathrm{RB}}P)$ is the velocity of the connection point P of the rigid body.

Description	$\widehat{\vec{v}}$	$\widehat{\vec{T}}$	$\widehat{\hat{\imath}}$	$\widehat{\dot{\hat{\imath}}}$	$\widehat{\vec{M}_*}$
Prescribed motion, pinned	$\vec{v}_{\mathrm{BC}}$	$\vec{T}^+$	$\hat{\imath}^+$	$\dot{\hat{\imath}}^+$	$\vec{0}$
Pinned joint	$\vec{0}$	$\vec{T}^+$	$\hat{\imath}^+$	$\dot{\hat{\imath}}^+$	$\vec{M}_*^+$
Free cable end	$\vec{v}^+$	$\vec{0}$	$\hat{\imath}^+$	$\dot{\hat{\imath}}^+$	$\vec{0}$
Clamped fixed end	$\vec{0}$	$\vec{T}^+$	$\hat{\imath}_{\mathrm{BC}}$	$\vec{0}$	$\vec{M}_*^+$
Point force and moment	$\vec{v}^+$	$\vec{T}_{\mathrm{BC}}$	$\hat{\imath}^+$	$\dot{\hat{\imath}}^+$	$\vec{M}_{*\mathrm{BC}}$
Rigid body connection at point P	$\vec{v}^{(\mathrm{P})}_{\mathrm{RB}}$	$\vec{T}^+$	$\hat{\imath}^+$	$\dot{\hat{\imath}}^+$	$\vec{0}$

where the $\beta \in [-0.5, 0.5]$ parameter governs the amount of flux taken from the left and right side respectively for each equation. Moody is released with a centred scheme ($\beta = 0$) to avoid one-sided bias on the auxiliary variable $\vec{M}_*$ in the results.

A.3.1 Boundary Conditions

The boundary conditions are introduced via numerical flux values at the edges of the finite element domain. The original Lax–Friedrichs formulation without bending stiffness [8] required the numerical fluxes of velocity $\widehat{\vec{v}}$ or the tension force vector $\widehat{\vec{T}}$ to be given as the boundary condition on the domain boundaries. The implementation of the nested LDG method for bending stiffness additionally requires that fluxes for the cable tangential vector $\widehat{\hat{\imath}}$ or for the total moment vector $\widehat{\vec{M}_*}$ be defined. Finally, if bending damping is included, also the time derivative of the cable direction $\widehat{\dot{\hat{\imath}}}$ is required. Boundary conditions for the auxiliary variables $\hat{\imath}$ or $\vec{M}_*$ associated with the bending stiffness may be specified independently of the conditions set for the cable velocity and tension force at each end point. Variables for which no boundary condition is specified are reactions and are simulated by collecting the flux from the interior domain. Table A.1 describe the combinations of boundary conditions required to model some typical end point properties for marine cables. We use the term pinned to describe a point where no moments are transferred. When moments are transferred, we label the condition as a clamped one.

A.4 Components

Currently, only spring-dampers are the only component implemented in Moody.

A.4.1 Spring-dampers

Spring-dampers are mass-less components that compute axial tension force based on relative displacement and velocity between its two attachment points $\vec{r}_1$ and $\vec{r}_2$. The rest-length L_0, and the current length $L = \left|\vec{r}_2 - \vec{r}_1\right|$, gives the elongation as $x = L - L_0$ (m). The inline relative velocity is computed as $v = \left|\dot{\vec{r}}_2 - \dot{\vec{r}}_1\right| \cdot \hat{t}$, where $\hat{t} = (\vec{r}_2 - \vec{r}_1)/L$ is the unit tangential vector. The tension force is computed as a polynomial (power series), according to (A.27).

$$T = \max\left(0, \sum_{i=1}^{P_C} C_i x^i + \sum_{i=1}^{P_D} D_i v^i\right), \tag{A.27}$$

where C_i and D_i are the polynomial coefficient of order i. P_C and P_D denote the polynomial orders of the respective polynomials.

A.5 Rigid body dynamics

A.5.1 Coordinate systems

Let $\mathcal{X} = \{\hat{x}, \hat{y}, \hat{z}\}$ be the inertial frame, and $\mathcal{I} = \{\hat{\imath}, \hat{\jmath}, \hat{k}\}$ denote the body fixed coordinate system with origin in the centre of mass, see Fig. A.1. Here $\vec{O}_\mathcal{X}$ is the centre of gravity of the buoy (in the inertial frame), coinciding with the origin of $\mathcal{I}$. The buoyancy centre ($\vec{OB}$) and the attachment points of the mooring lines ($\vec{OM}_i$) are both expressed in the local frame of reference. The unit quaternion

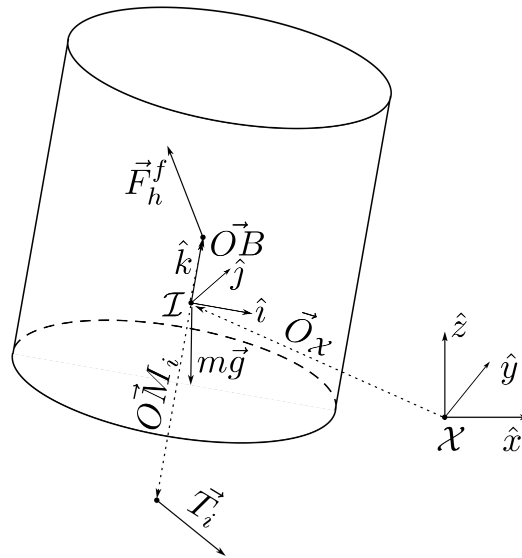

Figure A.1: Schematic view of the relation between the body-fixed coordinate system $\mathcal{I}$ and the inertial frame $\mathcal{X}$.

$\vec{b} = [b_0\, b_1\, b_2\, b_3]$ relates the orientation of the body frame to the inertial frame.

Further, the rotation matrix $\mathbf{R}(\vec{b})$ transforms vector components from $\mathcal{X} \rightarrow \mathcal{I}$ is written

$$\mathbf{R} = \begin{bmatrix} \sigma_b + B_{11} & B_{12} + B_{03} & B_{13} - B_{02} \\ B_{21} - B_{03} & \sigma_b + B_{22} & B_{23} + B_{01} \\ B_{31} + B_{02} & B_{32} - B_{01} & \sigma_b + B_{33} \end{bmatrix}, \tag{A.28}$$

in which $B_{ij} = 2\left(\vec{b} \otimes \vec{b}\right)$, and $\sigma_b = b_0^2 - b_1^2 - b_2^2 - b_3^2$. The inverse transform is described by $\mathbf{R}^{-1} = \mathbf{R}^{\mathrm{T}}$.

A.5.2 Equations of motion

A body connected to N cables is expressed as

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{M}^b \vec{u}\right) = \vec{F}_{\mathrm{a}}^b + \vec{F}_{\mathrm{b}}^b + \vec{F}_{\mathrm{d}}^b + \sum_{i=1}^{N} \vec{F}_i\,, \tag{A.29}$$

where $\mathbf{M}^b$ is the 6×6 mass matrix, $\vec{u}^b = \left[\vec{v}^b\, \vec{\omega}^b\right]^{\mathrm{T}}$ is the 6×1 state vector of velocity representing linear, $\vec{v}^b$, and rotational, $\vec{\omega}^b$, velocities of the body. The body is affected by added mass, $\vec{F}_{\mathrm{a}}^b$, buoyancy forces, $\vec{F}_{\mathrm{b}}^b$, and drag forces, $\vec{F}_{\mathrm{d}}^b$, as well as the restraining action of the mooring lines $\sum_{i=1}^{N} \vec{F}_i$.

The time derivatives of $\vec{O}_X$ and $\vec{b}$ can be updated from the velocity $\vec{v}^b$ and the angular velocity $\vec{\omega}^b$ as

$$\frac{\partial \vec{O}_X}{\partial t} = \mathbf{R}^{\mathrm{T}} \vec{v}^b\,, \tag{A.30}$$

$$\frac{\partial \vec{b}}{\partial t} = \frac{1}{2} \begin{bmatrix} b_0 & -b_1 & -b_2 & -b_3 \\ b_1 & b_0 & -b_3 & b_2 \\ b_2 & b_3 & b_0 & -b_1 \\ b_3 & -b_2 & b_1 & b_0 \end{bmatrix} \begin{bmatrix} 0 \\ \vec{\omega}^b \end{bmatrix}. \tag{A.31}$$

Here $\vec{O}_X$ is a position vector in the inertial frame $\mathcal{X}$ while all other state vectors have components in the Lagrangian frame ($\mathcal{I}$). Expressing Eq. (A.29) in $\mathcal{I}$ and separating the rotational and translational degrees of freedom gives

$$\begin{bmatrix} m^b \dfrac{\partial \vec{v}^b}{\partial t} \\ \mathbf{I} \dfrac{\partial \vec{\omega}^b}{\partial t} \end{bmatrix} = -\begin{bmatrix} \vec{\omega}^b \times m^b \vec{v}^b \\ \vec{\omega}^b \times \mathbf{I} \vec{\omega}^b \end{bmatrix} + \vec{F}_{\mathrm{a}}^b + \vec{F}_{\mathrm{b}}^b + \vec{F}_{\mathrm{d}}^b + \sum_{i=1}^{N} \vec{F}_i\,, \tag{A.32}$$

$$\vec{F}_{\mathrm{b}}^b = \begin{bmatrix} \left(m^b - V^b \rho^f\right) \mathbf{R} \vec{g} \\ -\vec{OB} \times V^b \rho^f \mathbf{R} \vec{g} \end{bmatrix}, \tag{A.33}$$

$$\vec{F}_i = \begin{bmatrix} \mathbf{R} \vec{T}_i \\ \vec{OM}_i \times \mathbf{R} \vec{T}_i \end{bmatrix}, \tag{A.34}$$

where m^b is the body mass, $\mathbf{I}$ is the 3×3 inertia matrix of the body and V^b is the volume of the body. The fluid density is denoted by ρ^f and $\vec{g}$ is the acceleration of gravity.

The expressions for added mass and drag damping are a bit more complicated. The following applies for a small cylindrical buoy (relative to changes in the flow field) with $\mathcal{I}$ located in the geometrical center, and $\hat{k}$ along the symmetry axis. The fluid velocity and acceleration are treated as constant over the body domain and are evaluated at the origin of $\mathcal{I}$, i.e. $\vec{v}^f = \vec{v}^f_{\mathrm{O}}$ and $\partial\vec{v}^f/\partial t = \vec{a}^f_{\mathrm{O}}$. Then, using $\partial\vec{v}^b/\partial t = \vec{a}^b$, and $\partial\vec{\omega}^b/\partial t = \vec{\alpha}^b$, the added mass force and moment vector $\vec{F}^b_{\mathrm{a}}$ (6×1) is written as

$$\vec{F}^b_{\mathrm{a}} = V^b\rho^f \begin{bmatrix} \vec{a}^f_{\mathrm{O}\perp}(1 + C_{\mathrm{M1}}) - C_{\mathrm{M1}}\vec{a}^b_{\perp} + (1 + C_{\mathrm{M2}})\, a^f_{\mathrm{O}\hat{k}} - C_{\mathrm{M2}} a^b_{\hat{k}}\hat{k} \\ -\frac{(h^b)^2}{12} C_{\mathrm{M1}}\vec{\alpha}^b_{\perp} + 0\hat{k} \end{bmatrix}. \tag{A.35}$$

The index $\perp$ denotes vector projection onto the $\hat{\imath}, \hat{\jmath}$ plane and h^b is the cylinder height. C_{M1} and C_{M2} are the coefficients of added mass perpendicular and parallel to the symmetry axis respectively. The factor $h^2/12$ comes from integrating the moment equation along the cylinder.

For drag forces, the quadratic term makes the resulting integral expressions exceedingly expensive to evaluate, and we therefore leave the expression in integral form as

$$\vec{F}^b_{\mathrm{d}} = \frac{\rho^f D^b}{2} \begin{bmatrix} C_{\mathrm{D1}}\vec{Q}^{(1)} + \left(C_{\mathrm{D2}}\frac{\pi(D^b)}{4} + C_{\mathrm{D3}}h^b\right)\left|v^*_{\hat{k}}\right| v^*_{\hat{k}}\hat{k} \\ C_{\mathrm{D1}}\vec{Q}^{(2)} - C_{\mathrm{D3}}h^b\frac{(D^b)^3}{8}\left|\omega_{\hat{k}}\right|\omega_{\hat{k}}\hat{k} \end{bmatrix}, \tag{A.36}$$

$$\vec{Q}^{(1)}(\vec{v}^*) = \int_{-h^b/2}^{h^b/2} \sqrt{\vec{v}^*_{\perp}\cdot\vec{v}^*_{\perp}}\,\vec{v}^*_{\perp}\,\mathrm{d}z\,, \tag{A.37}$$

$$\vec{Q}^{(2)}(\vec{v}^*) = \int_{-h^b/2}^{h^b/2} z\sqrt{\vec{v}^*_{\perp}\cdot\vec{v}^*_{\perp}}\left(\hat{k}\times\vec{v}^*_{\perp}\right)\mathrm{d}z\,, \tag{A.38}$$

where C_{D1}, and C_{D2} are the in-plane and out of plane drag coefficients of a circle respectively, and C_{D3} is the sectional shear coefficient of tangential drag and D^b is the cylinder diameter. $\vec{Q}^{(1)}$ and $\vec{Q}^{(2)}$ denote the integrals over the cylinder height as a function of the local section velocity $\vec{v}^* = \vec{v}^f_{\mathrm{O}} - \vec{v}^b - \vec{\omega}^b \times z\hat{k}$.

We rewrite Eqs. (A.29)–(A.31) in terms of a state vector with 13 degrees of freedom

$$\vec{U}^b = \left[\vec{O}_X,\, \vec{b},\, \vec{u}^b\right]^{\mathrm{T}}. \tag{A.39}$$

For submerged bodies, the effective mass matrix $\mathbf{M}^b_{\mathrm{e}}$ describing inertial effects of the body and the surrounding fluid is constant in the body-fixed coordinate system. Further, if the centre of mass coincides with the centre of buoyancy $\mathbf{M}^b_{\mathrm{e}}$ reduces

to a diagonal matrix written as

$$\operatorname{diag}\left(\mathbf{M}_{\mathrm{e}}^{b}\right) = \begin{bmatrix} m^b + V^b \rho^f C_{\mathrm{M1}} \\ m^b + V^b \rho^f C_{\mathrm{M1}} \\ m^b + \dfrac{2V^b \rho^f C_{\mathrm{M2}}}{h^b} \\ I_{\hat{\imath}} + \alpha_h C_{\mathrm{M1}} \\ I_{\hat{\jmath}} + \alpha_h C_{\mathrm{M1}} \\ I_{\hat{k}} \end{bmatrix}, \tag{A.40}$$

with $\alpha_h = (h^b)^2/12$, derived from Eq. (A.35). The constant inverse of $\mathbf{M}_{\mathrm{e}}^{b}$ is then trivially computed.

Forces and moments from each mooring cable are computed in $\mathcal{I}$ based on the attachment point location $\vec{OM}_i$, see Eq. (A.34). The position and velocity of point $\vec{M}_i$ in $\mathcal{X}$ are then

$$\vec{M}_i^{(\mathcal{X})} = \vec{O}_{\mathcal{X}} + R^{\mathrm{T}} \vec{OM}_i\,, \tag{A.41}$$

$$\frac{\partial \vec{M}_i^{(\mathcal{X})}}{\partial t} = R^{\mathrm{T}} \left(\vec{v}^b + \vec{\omega}^b \times \vec{OM}_i \right), \tag{A.42}$$

which are used as boundary conditions for any cable connected to attachment point $\vec{OM}_i$.

A.6 Wave kinematics and wave-body interaction

MoodyCore uses the first order velocity potential to describe the incident wave field from one or several wave components. The choice of zero phase differs between different boundary element solvers for the radiation-diffraction problem, and the formulation for the incident wave potential is therefore slightly different. To simplify notation, the following relates to a single regular wave component. The formulation used in MoodyCore is based on an intuitive regular wave component elevation η as

$$\eta = a\cos(k_x x + k_y y - \omega t + \phi) = \Re\left(ae^{k_x x + k_y y - \omega t + \phi}\right), \tag{A.43}$$

with a - amplitude, k_x and k_y the wave number components in x and y respectively, ω as the angular frequency, ϕ as the wave phase and t as the time. To the first order linearisation of the free surface condition, the elevation relates to the velocity potential as

$$\frac{\partial\Phi(z=0)}{\partial t} + g\eta = 0, \tag{A.44}$$

We therefore follow the Nemoh notation and define the complex incident wave potential Φ_I as

$$\Phi_I = -i\frac{ag}{\omega}Z(\zeta)e^{k_x x + k_y y - \omega t + \phi}, \tag{A.45}$$

$$Z(\zeta) = \frac{\cosh k(h+\zeta)}{\cosh kh}, \tag{A.46}$$

where ζ is the Wheeler stretched vertical coordinate acc. to Eq. (A.47) and $Z(\zeta)$ is the depth variation factor, which is evaluated as $e^{k\zeta}$ below the deep water limit defined in moody core as: $k\zeta \leq \pi$. By following the same definition as Nemoh and Capytaine, the resulting excitation forces from these codes can be used directly in MoodyCore.

A.6.1 Wheeler stretching

The Wheeler stretching method is applied to the vertical depth variation function $Z(\zeta)$ to approximate wave pressure and kinematics up to the instantaneous wave elevation $\eta(t)$. The vertical coordinate of the inertial frame z is transformed to ζ as

$$\zeta(z,\eta) = h\left(\frac{h+z}{h+\eta} - 1\right). \tag{A.47}$$

A.6.2 First order potential flow forces

MoodyCore uses the following definitions of the first order potential flow forces on a floating body:

F_{rad} Radiation force from the radiation potential Φ_{rad}, realised through the radiation Kernel (also known as the impulse response function) convolution integral with body velocity. Linearised at the initial body position.

F_{df} Diffraction force from the diffraction potential Φ_{df} (sometimes referred to as the scattered potential).

$F_{fk}^{(d)}$ Dynamic Froude-Krylov force from the integral of dynamic pressure in the incident wave potential Φ_I over the wetted body surface.

$F_{fk}^{(s)}$ Static Froude-Krylov force, being the hydrostatic pressure integrated over the wetted body surface.

F_{fk} Total Froude-Krylov force, the sum of static and dynamic Froude-Krylov forces $F_{fk} = F_{fk}^{(s)} + F_{fk}^{(d)}$.

F_{ex} Complex-valued excitation force, $F_{ex} = F_{fk}^{(d)} + F_{df}$

In the fully linear simulation of the Cummins equation, F_{ex} is used to as external wave forcing for any wave amplitude, and the linearised hydrostatic force $F_{fk}^{(s)}$ is transformed into a constant restoring stiffness matrix **C** times the body displacement $\vec{x}$. When the nonlinear Froude-Krylov formulation is activated, F_{df} is instead used as linearised external wave force, and the restoring force is evaluated as the total Froude Krylov force $F_{fk} = F_{fk}^{(s)} + F_{fk}^{(d)}$. The total FK force is reported in the restoring force output of the body, and the excitation force output covers only the linearised diffraction force.

A.7 Static solver and relaxation

The static equilibrium of the total system of floating bodies, submerged buoys, cables and components is in MoodyCore approximated through a two-step sequence: stage 1 - solve, stage 2 - relax.

A.7.1 Solve

The initial conditions of cables are defined as different choices of analytical quasi-static mooring solvers with given end-points for each cable individually. That is, for each set of vertex positions, an analytical estimate of the cable forces can be quickly obtained. Each body (floating or submerged) is solved for at $t = t_0$ by a sequence of i Euler steps of step size Δ_i. The velocity of each body is removed after each step so that consistent quasi-static velocities are obtained for the tension force of the cables.
The algorithm takes wind and ocean current steady forces on bodies into account

unless they have a ramp-time specified. Hydrodynamic forces on cables are however neglected in the equilibrium due to the analytical nature of the elastic catenary or straight line initial conditions specified. A closer estimate of the complete equilibrium of the system is instead obtained by the `relax` stage of the static solver.

A.7.2 Relax

The relax stage is simply a fixed point iteration of the dynamic equations, with a relaxation factor applied to the velocity states of all cables after each step. The algorithm is experimental and prone to instabilities for high-celerity cables. Manual tuning of parameters `relaxIter`, `relaxFactor`, and `relaxTimestep` may be required. The relax stage is typically needed mostly for cases where no initial transients are acceptable. In most practical applications however, an initial ramping stage is used to build up the wave field, in which case the initial transients in the simulation can be allowed in the startup of the dynamic evaluation instead.

A.8 API BOUNDARY CONDITIONS

When coupled to an external solver (i.e. in API mode), the timestep requirement on the cable dynamics is potentially orders of magnitude smaller than that required by the external solver. To save computational effort, the boundary conditions that are sent to Moody at each coupling time are interpolated to achieve intermediate boundary conditions. The position of the attachment point is interpolated based on constant acceleration over the time step, and a staggered process is used to maintain smoothness.

To explain, we let t_k and t_{k+1} be two consecutive coupling times, with corresponding mooring point positions r_k and r_{k+1}. We introduce the staggered-time fraction $\phi \in [0, 1]$ and identify a corresponding mooring time $t^{\mathrm{m}}_{k+1} \in [t_k, t_{k+1}]$ as

$$t^{\mathrm{m}}_{k+1} = \phi t_k + (1 - \phi)\, t_{k+1}\,. \tag{A.48}$$

The mooring boundary conditions $r_{\mathrm{D}}(t)$ and $v_{\mathrm{D}}(t)$ are interpolated over the mooring time step interval $t \in \left[t^{\mathrm{m}}_k, t^{\mathrm{m}}_{k+1}\right]$ as

$$r_{\mathrm{D}}\left(t^{\mathrm{m}}_k + \tau\right) = r^{\mathrm{m}}_k + v^{\mathrm{m}}_k \tau + 0.5 a_k \tau^2\,, \tag{A.49}$$

$$v_{\mathrm{D}}\left(t^{\mathrm{m}}_k + \tau\right) = v^{\mathrm{m}}_k + a_k \tau\,, \tag{A.50}$$

where $\tau \in \left[0, t^{\mathrm{m}}_{k+1} - t^{\mathrm{m}}_k\right]$, while $r^{\mathrm{m}}_k = r_{\mathrm{D}}\left(t^{\mathrm{m}}_k\right)$ and $v^{\mathrm{m}}_k = v_{\mathrm{D}}\left(t^{\mathrm{m}}_k\right)$ are taken from the previous coupling interval. To close the system, we only need to define a_k. Here, we choose a_k as the constant acceleration needed to satisfy $r_{\mathrm{D}}\left(t_{k+1}\right) = r_{k+1}$, i.e.

$$a_k = \frac{r_{k+1} - r^{\mathrm{m}}_k - v^{\mathrm{m}}_k \Delta_k}{0.5\Delta_k^2}\,, \tag{A.51}$$

with $\Delta_k = t_{k+1} - t^{\mathrm{m}}_k$.