MAP++ Documentation

Release 1.15

Marco Masciola

Mar 16, 2018

Contents


MOORING
ANALYSIS PROGRAM

CHAPTER 1

Release Notes

Note: Example input files are provided to demonstrate MAP++. These fake examples do not represent realistic, practical moorings for permanent installations.

1.1 License

MAP++ is licensed under Apache v 2.0 *License*.

1.1.1 License

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

      i. You must cause any modified files to carry prominent notices stating that You changed the files; and

         A. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

         B. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices

normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

4. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
5. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
6. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
7. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
8. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

1.2 Disclaimer

This software is provided as-is and without warranty. There are no guarantees it is bug free or provides the correct answers, even if it is used for the intended purpose. By using this software and as a condition of the Apache license, you agree to not hold any MAP++ developer liable for damages.

1.3 Dependencies

Third party dependencies are distributed with the MAP++ archive on BitBucket. Required libraries include the following:

Library	Version Distributed with MAP++
LAPACK	Version 3.5.0
C/C++ Minpack	Version 1.3.3
SimCList	Version 1.6
Better String Library	Version 0.1.1

1.4 Change Log

v1.20.00 – First release.

v1.20.10 – Repaired the linearized stiffness matrix function and improved the input file parsing in python.

CHAPTER 2

Definitions

2.1 What MAP++ Solves

Mooring models can be classified into two groups: static models and dynamic models. Static models ignore the inertia forces and fluid drag loads, and only account for the mean forces in the mooring system, including elasticity, weight (in fluid), and geometric nonlinearities. Static models are the type concerned in MAP++. Extra steps are taken to reformulate the classic single line, closed–form solution *[1]* into a piece-wise, multi-sgemented system. This piece wise system is composed of a collection of nodes and elements.

Each element in *Fig. 1* is expressed as a single line counterpart in two configurations. One configuration has the line hanging freely, whereas the second orientation account for friction and contact at the bottom boundary.

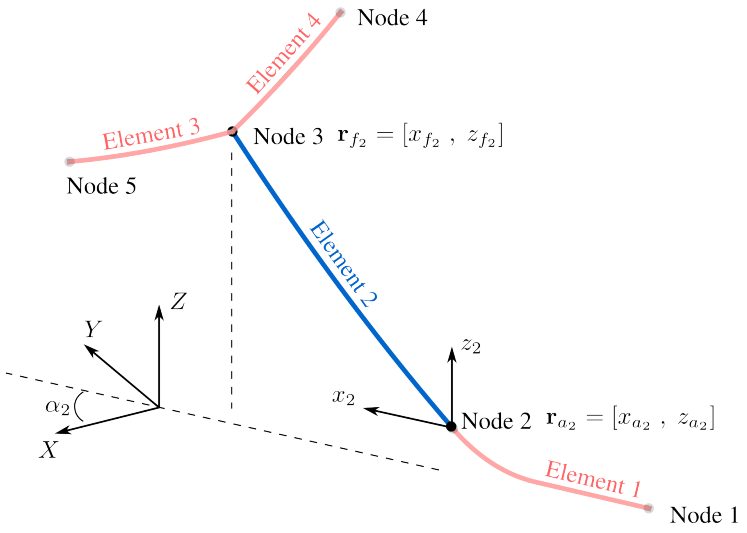


Fig. 2.1: Fig. 1

2.2 Nomenclature

Variable	Definition	Units
A	Cable cross-sectional area	[m^2]
C_B	Seabed contact friction coefficient	–
E	Young's modulus	[N/m^2]
g	Acceleration due to gravity	[m/s^2]
h	Vertical fairlead excursion	[m]
H	Horizontal fairlead force	[N]
H_a	Horizontal anchor force	[N]
l	Horizontal fairlead excursion	[m]
L	Unstretched line length	[m]
L_B	Line length resting on the seabed	[m]
M_i	Point mass applied to the ith node	[kg]
r_i	Node position vector [xi ; yi ; zi]	[m]
R_i	Rotation angle between the x_i and X axis	–
s	Unstretched distance from the anchor ($0 \leq s \leq L$)	[m]
T_j	Cable tension vector for the jth elemetn	[N]
$Te(s)$	Cable tangential tension at distance s	[N]
V	Vertical fairlead force	[N]
V_a	Vertical anchor force	[N]
w	Cable weight-per-unit length in fluid	[N/m]
x_0	Horizontal force transition point for $H(s) > 0$	[N]
ρ	Fluid density	[kg/m^3]

CHAPTER 3

Theory

The solution process begins by evaluating the two continuous analytical catenary equations for each element based on l and h values obtained through node displacement relationships. An element is defined as the component connecting two adjacent nodes together. Once the element fairlead (H, V) and anchor (H_a, V_a) values are known at the element level, the forces are transformed from the local $x_i z_i$ frame into the global XYZ coordinate system. The force contribution at each element's anchor and fairlead is added to the corresponding node it attaches to.

The force-balance equation is evaluated for each node, as follows:

$$\{\mathbf{F}\}_X^j = \sum_{i=1}^{\text{Element i at Node } j} [H_i \cos(\alpha_i)] - F_{X_j}^{ext} = 0$$

$$\{\mathbf{F}\}_Y^j = \sum_{i=1}^{\text{Element } i \text{ at Node } j} [H_i \sin(\alpha_i)] - F_{Y_j}^{ext} = 0$$

$$\{\mathbf{F}\}_Z^j = \sum_{i=1}^{\text{Element } i \text{ at Node } j} [V_i] - F_{Z_j}^{ext} + M_j g - \rho g B_j = 0$$

Node forces are found based on the connectivity geometry between element and external forces applied at the boundary conditions. This *is initiated by defining a series* of $\mathcal{F}_i$ local frames at the origin in which the individual line elements are expressed in. Frame $\mathcal{F}_0$ is an arbitrary global axis, but it is usually observed as the vessel reference origin.

Note: Simplistic way to think of MAP++'s dichotomy between nodes and elements: Nodes define the force at connection points. Elements define the mooring geometry.

Clearly, this process requires two distinct sets of equations, one of which must be solved within the other routine, to find the static cable configuration. The first set of equations are the force{balance relationships in three directions for each node; the second set of equations are the catenary functions proportional to the number of elements. Interactions between solves is captured in the *flowchart below to summarize the solve procedure*. This method was first proposed in *[4]*.

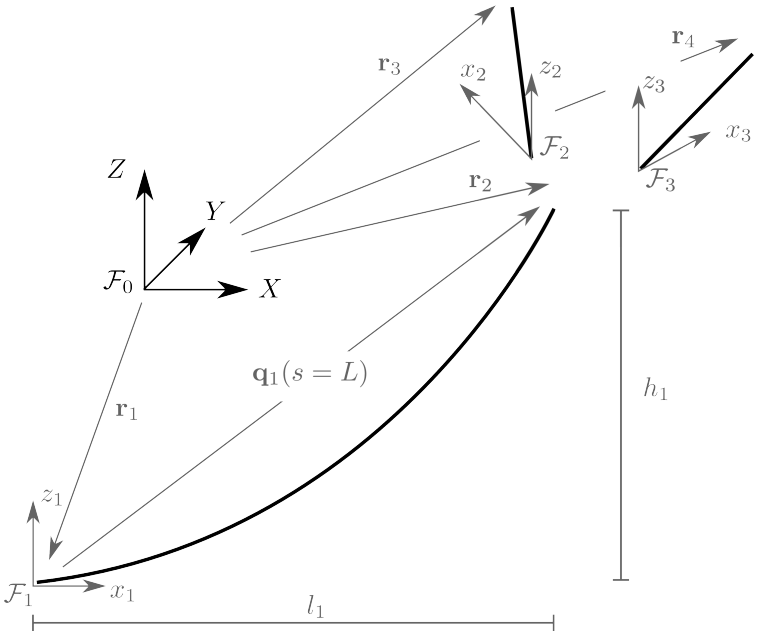


Fig. 3.1: Fig. 2

3.1 Line Theory

3.1.1 Free–Hanging Line

The equations used to describe the shape of a suspended chain illustrated in *Fig. 4* have been derived in numerous works *[1]*. For completeness, a summary of the governing equations used inside the MSQS model are presented. Given a set of line properties, the line geometry can be expressed as a function of the forces exerted at the end of the line:

$$x\left(s\right)=\frac{H}{\omega}\left\{\ln\left[\frac{V_a+\omega s}{H}+\sqrt{1+\left(\frac{V_a+\omega s}{H}\right)^2}\right]-\ln\left[\frac{V_a}{H}+\sqrt{1+\left(\frac{V_a}{H}\right)^2}\right]\right\}+\frac{Hs}{EA}$$

$$z\left(s\right)=\frac{H}{\omega}\left[\sqrt{1+\left(\frac{V_a+\omega s}{H}\right)^2}-\sqrt{1+\left(\frac{V_a}{H}\right)^2}\right]+\frac{1}{EA}\left(V_a s+\frac{\omega s^2}{2}\right)$$

where:

$$\omega=gA\left(\rho_{\text{cable}}-\rho\right)$$

and x and z are coordinate axes in the local (element) frame, *Fig. 2*. The following substitution can be made for V_a in the above equations:

$$H_a=H$$

$$V_a=V-\omega L$$

which simply states the decrease in the vertical anchor force component is proportional to the mass of the suspended line. The equations for $x(s)$ and $z(s)$ both describe the catenary profile provided all entries on the right side of the equations are known. However, in practice, the force terms H and V are sought, and the known entity is the fairlead

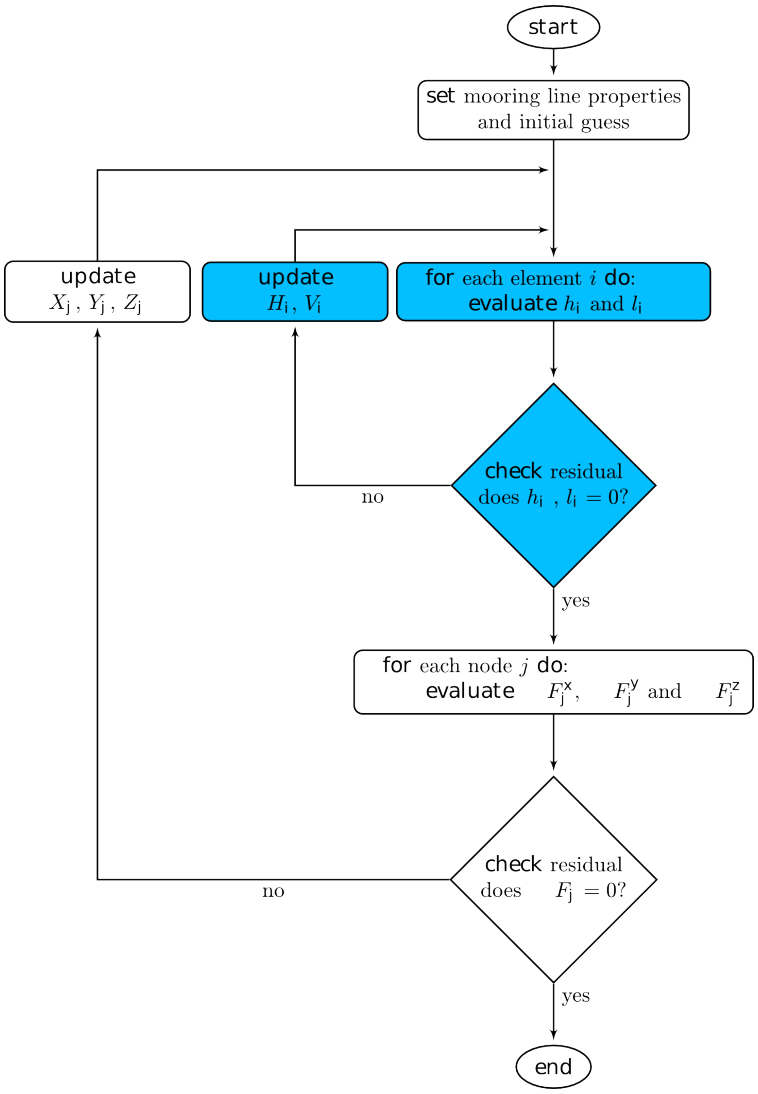

start
set mooring line properties and initial guess
update X_j, Y_j, Z_j
update H_i, V_i
for each element i do: evaluate h_i and l_i
check residual does h_i , $l_i = 0$?
no
yes
for each node j do: evaluate F_j^x, F_j^y and F_j^z
check residual does $F_j = 0$?
no
yes
end

Fig. 3.2: Fig. 3

excursion dimensions, l and l. In this case, the forces H and V are found by simultaneously solving the following two equations:

$$l = \frac{H}{\omega}\left[\ln\left(\frac{V}{H} + \sqrt{1+\left(\frac{V}{H}\right)^2}\right) - \ln\left(\frac{V-\omega L}{H} + \sqrt{1+\left(\frac{V-\omega L}{H}\right)^2}\right)\right] + \frac{HL}{EA}$$

$$h = \frac{H}{\omega}\left[\sqrt{1+\left(\frac{V}{H}\right)^2} - \sqrt{1+\left(\frac{V-\omega L}{H}\right)^2}\right] + \frac{1}{EA}\left(VL - \frac{\omega L^2}{2}\right)$$

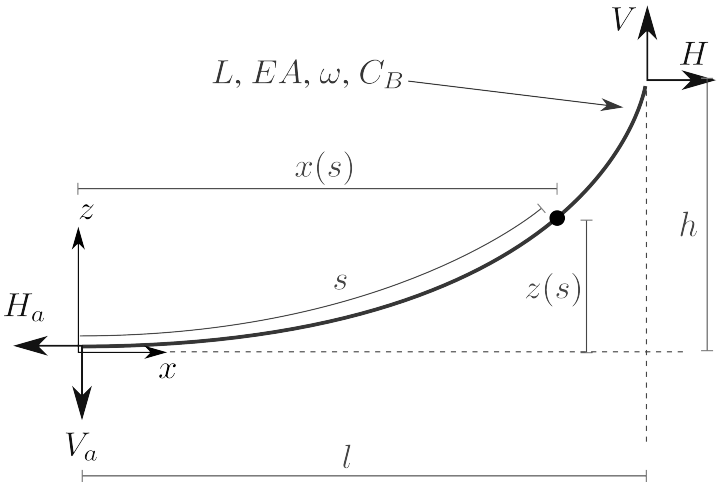


Fig. 3.3: Fig. 4

3.1.2 Line Touching the Bottom

The solution for the line in contact with a bottom boundary is found by continuing $x(s)$ and $z(s)$ beyond the seabed touch–down point $s = L_B$. Integration constants are added to ensure continuity of boundary conditions between equations:

$$x(s) = \begin{cases} s & \text{if } 0 \le s \le x_0 \\ s + \frac{C_B\omega}{2EA}\left[s^2 - 2x_0 s + x_0\lambda\right] & \text{if } x_0 < s \le L_B \\ L_B + \frac{H}{\omega}\ln\left[\frac{\omega(s-L_B)}{H} + \sqrt{1+\left(\frac{\omega(s-L_B)}{H}\right)^2}\right] + \frac{Hs}{EA} + \frac{C_B\omega}{2EA}\left[x_0\lambda - L_B^2\right] & \text{if } L_B < s \le L \end{cases}$$

where λ is:

$$\lambda = \begin{cases} L_B - \frac{H}{C_B\omega} & \text{if } x_0 > 0 \\ 0 & \text{otherwise} \end{cases}$$

Between the range $0 \le s \le L_B$, the vertical height is zero since the line is resting on the seabed and forces can only occur parallel to the horizontal plane. This produces:

$$z(s) = \begin{cases} 0 & \text{if } 0 \le s \le L_B \\ \frac{H}{\omega}\left[\sqrt{1+\left(\frac{\omega(s-L_B)}{H}\right)^2} - 1\right] + \frac{\omega(s-L_B)^2}{2EA} & \text{if } L_B < s \le L \end{cases}$$

The equations above produce the mooring line profile as a function of s. Ideally, a closed–form solution for l and h is sought to permit simultaneous solves for H and V, analogous to the freely–hanging chase in the previous section. This is obtained by substituting $s = L$ to give:

$$l = L_B + \left(\frac{H}{\omega}\right) \ln\left[\frac{V}{H} + \sqrt{1 + \left(\frac{V}{H}\right)^2}\right] + \frac{HL}{EA} + \frac{C_B\omega}{2EA}\left[x_0\lambda - L_B^2\right]$$

$$h = \frac{H}{\omega}\left[\sqrt{1 + \left(\frac{V}{H}\right)^2} - 1\right] + \frac{V^2}{2EA\omega}$$

Finally, a useful quantity that is often evaluated is the tension as a function of s along the line. This is given using:

$$T_e(s) = \begin{cases} \text{MAX}\left[H + C_B\omega(s - L_B)\ ,\ 0\right] & \text{if } 0 \le s \le L_B \\ \sqrt{H^2 + [\omega(s - L_B)]^2} & \text{if } L_B < s \le L \end{cases}$$

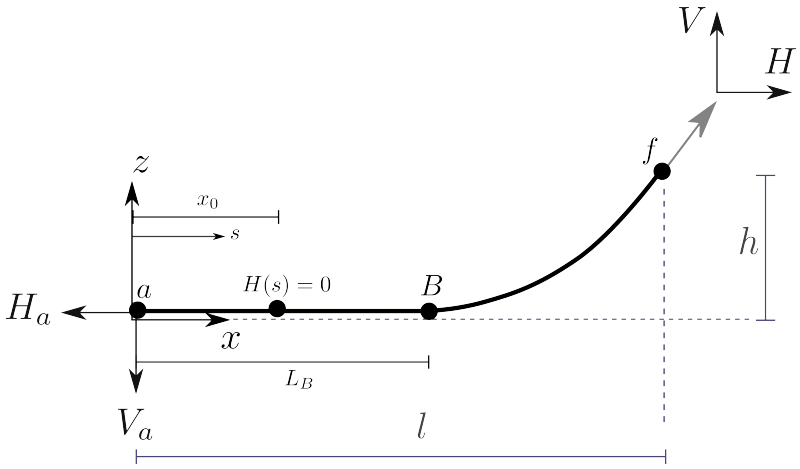


Fig. 3.4: Fig. 5

3.2 Vessel

3.2.1 Reference Origin

$$\mathbf{R} = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\phi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

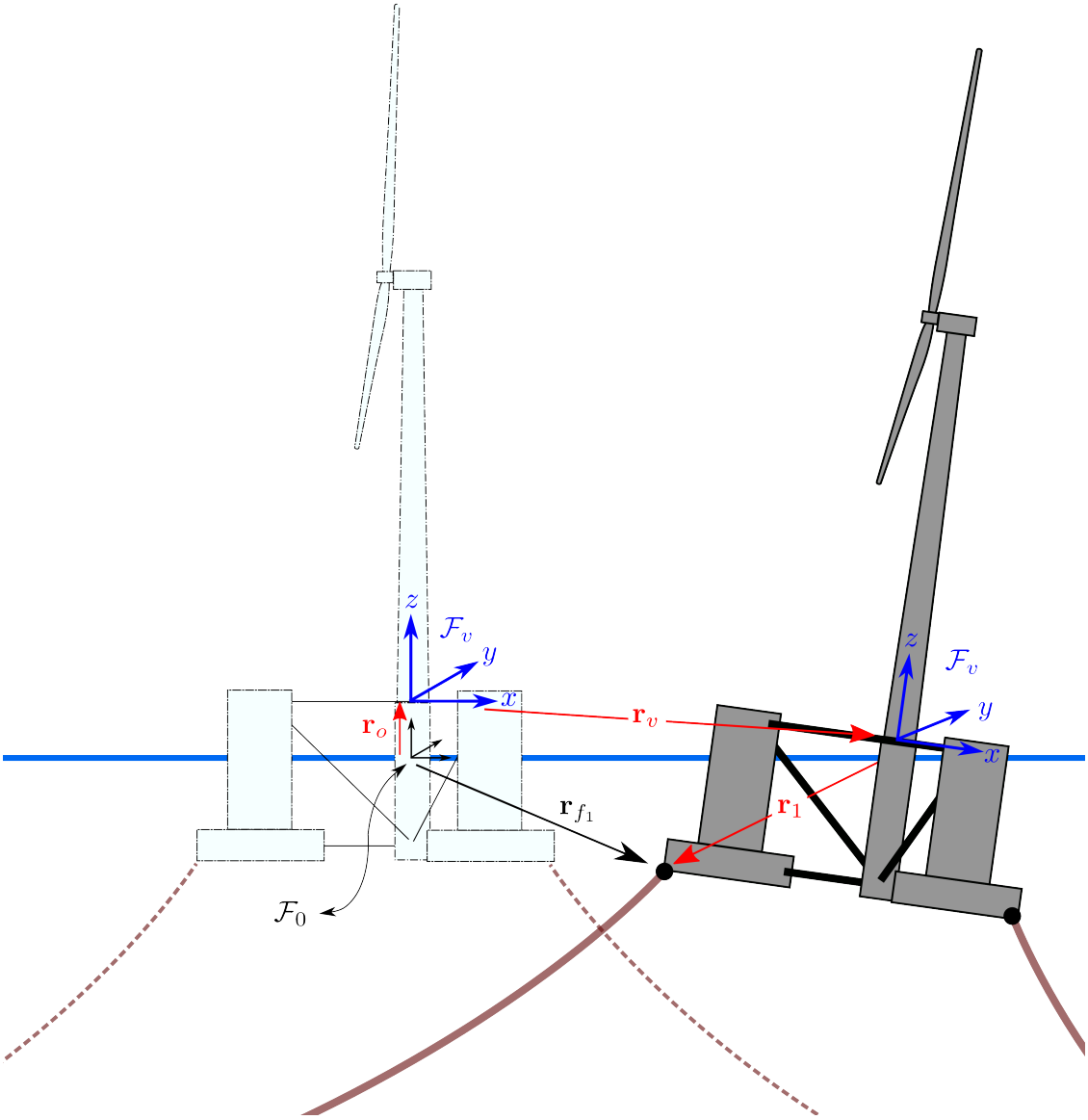

z
$\mathcal{F}_v$
y
x
$\mathbf{r}_o$
$\mathbf{r}_v$
$\mathbf{r}_{f_1}$
$\mathcal{F}_0$
z
$\mathcal{F}_v$
y
x
$\mathbf{r}_1$

Fig. 3.5: Fig. 6

CHAPTER 4

Input File

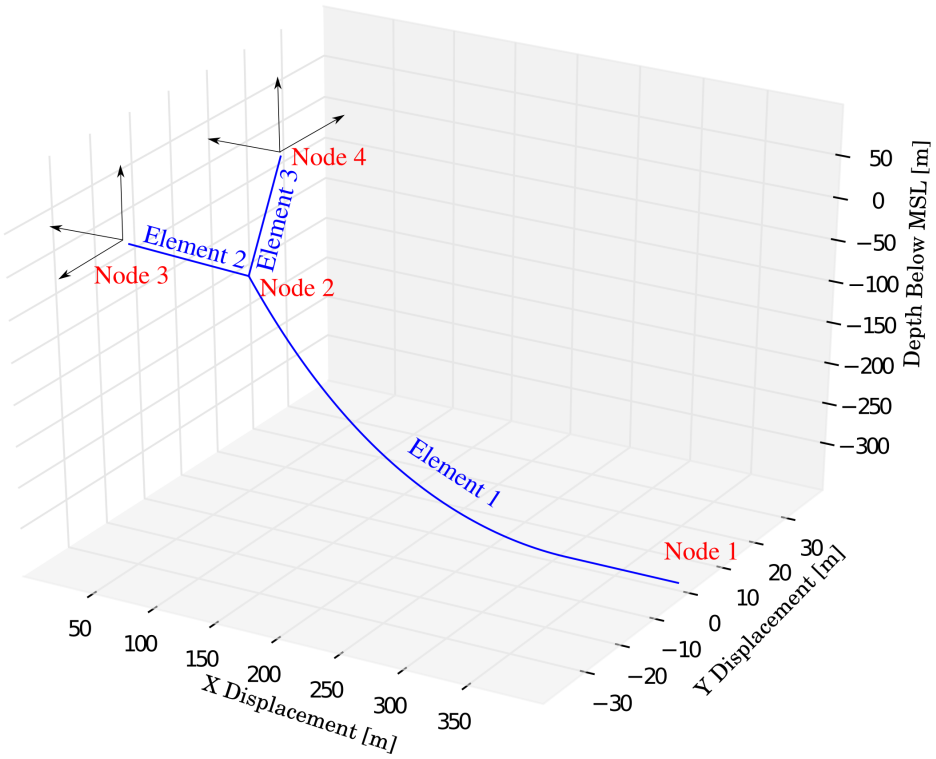
The MAP++ input file define the mooring properties, material definitions, connections between lines, and identify lines anchored or attached to a vessel. We use the extension <*.map> to identify the MAP++ input file. The sample MAP++ input deck and relevant commands are defined on this page.

4.1 Baseline Example

The baseline example below is a template on how properties are defined in MAP++:

```
--------------- LINE DICTIONARY ----------------------------------------------
LineType  Diam      MassDenInAir    EA         CB   CIntDamp  Ca     Cdn     Cdt
(-)       (m)       (kg/m)         (N)        (-)    (Pa-s)   (-)    (-)     (-)
mat_1     0.25       320.0     9800000000   1.0    -999.9 -999.9 -999.9 -999.9
mat_2     0.30       100.0     980000000    1.0    -999.9 -999.9 -999.9 -999.9
--------------- NODE PROPERTIES ----------------------------------------------
Node Type       X       Y       Z      M     B     FX    FY    FZ
(-)  (-)       (m)     (m)     (m)    (kg)  (m^3) (N)   (N)   (N)
1    fix     400        0     depth    0     0      #     #     #
2    connect #90       #0    #-80      0     0      0     0     0
3    vessel   20        20    -10      0     0      #     #     #
4    vessel   20       -20    -10      0     0      #     #     #
--------------- LINE PROPERTIES ----------------------------------------------
Line    LineType  UnstrLen  NodeAnch  NodeFair  Flags
(-)      (-)       (m)       (-)       (-)       (-)
1       mat_1      450        1         2     altitude     x_excursion
2       mat_2      90         2         3     tension_fair
3       mat_2      90         2         4
--------------- SOLVER OPTIONS-----------------------------------------------
Option
(-)
outer_tol 1e-5
 repeat 120 240
```

The space preceeding `repeat 120 240` indicates that line is a comment and is ignored by MAP++. Executing this input file produced the mooring geoemtry illustrated here:



Note: Environmental properties like water depth, sea density, and gavity constant are set by the calling program. They are purposely absent in the MAP++ input file to prevent force imbalances from coefficient mismatches.

The MAP++ input file is divided into four sections:

- **LINE DICTIONARY:** Defines the material properties of the line.
- **NODE PROPERTIES:** Defines boundary constraints and extensional limits.
- **LINE PROPERTIES:** Associates a line with material properties and connectivity between nodes.
- **SOLVER OPTIONS:** Run-time options to engage different solve strategies.

4.2 Line Dictionary

Variable	Definition
`LineType`	User–defined name [-]
`Diam`	Material diameter [m]
`MassDenInAir`	Mass density in air [kg/m^3]
`EA`	Element axial stiffness [N/m]
`CB`	Cable/seabed friction coefficient [-]
`CIntDamp`	Unused
`Ca`	Unused
`Cdn`	Unused
`Cdt`	Unused

4.3 Node Properties

Variable	Definition
NODE	Node number (sequential)
TYPE	Type of node, which can be one of FIX, CONNECT, or VESSEL. Vessel implied the node motion is prescribed.
X	Global x coordinate if node is FIX or CONNECT [m]. Local x cooridinate relative to vessel if node is VESSEL [m]. Connect nodes must be preceeded by a # is indicate this is as an initial guess.
Y	Global y coordinate if node is FIX or CONNECT [m]. Local y cooridinate relative to vessel if node is VESSEL [m]. Connect nodes must be preceeded by a # is indicate this is as an initial guess.
Z	Global z coordinate if node is FIX or CONNECT [m]. Local z cooridinate relative to vessel if node is VESSEL [m]. Connect nodes must be preceeded by a # is indicate this is as an initial guess.
M	Point mass applied to the node [kg]. The force appled to the node is $M \times g$ applied in the direction of gravity.
B	Displaced volume applied to node [m^3]. The force applied is $B \times \rho \times g$ applied opposite of gravity.
FX	x direction external force applied to CONNECT node [N]. VESSEL and FIX must use # to indicate iterated value. # can be preceeded by user–suplied initial guess to speed convergence.
FY	y direction external force applied to CONNECT node [N]. VESSEL and FIX must use # to indicate iterated value. # can be preceeded by user–suplied initial guess to

4.4 Line Properties

Variable	Definition
`Line`	Line number (sequential).
`LineType`	Line type. Must be one type defined in `LineType` from dictionary.
`UnstrLen`	Unstretched line length [m].
`NodeAnch`	Anchor node number
`NodeFair`	Fairlead node number
`Flags`	Line flag. Can include any command included in *Flags*

4.5 Flags

Flags are applied to individual lines as indicated in the 'LINE PROPERTIES' section of the input file above. These flags control the output text stream:

Variable	Definition
`GX_POS`	global X fairlead position [m]
`GY_POS`	global Y fairlead position [m]
`GZ_POS`	global Z fairlead position [m]
`GX_A_POS`	global X position of anchor [m]
`GY_A_POS`	global Y position of anchor [m]
`GZ_A_POS`	global Z position of anchor [m]
`GX_FORCE`	global X fairlead force [N]
`GY_FORCE`	global Y fairlead force [N]
`GZ_FORCE`	global Z fairlead force [N]
`H_FAIR`	horizontal (XY plane) fairlead force [N]
`H_ANCH`	horizontal (XY plane) anchor force [N]
`V_FAIR`	vertical (Z axis) fairlead force [N]
`V_ANCH`	vertical (Z axis) anchor force [N]
`TENSION_FAIR`	fairlead force magnitude, [N]
`TENSION_ANCH`	anchor force magnitude, [N]
`X_EXCURSION`	line horizontal excursion [m]
`Z_EXCURSION`	line veritical excursion [m]
`AZIMUTH`	line azimuth angle with respect to the inertial reference frame [deg]
`ALTITUDE`	angle of declination at the fairlead [deg]
`ALTITUDE_ANCH`	line liftloff angle at the anchor [deg]

The follow flags enable/disable features for each line they are applied to:

Variable	Definition
`LINE_TENSION`	line tension force magnitude at fairlead [N]
`OMIT_CONTACT`	ignore seabed boundary and treat line as freely hanging
`LINEAR_SPRING`	model the line as a linear spring. Intended for taut lines
`LAY_LENGTH`	amount of line laying on the seabed [m]
`DIAGNOSTIC`	run diagonostics on line for each solve iteration
`DAMAGE_TIME`	time [sec] to disconnect fairlead from node. Not used

4.6 Solver Options

Solver options are applied to the entire model domain.

Variable	Definition
HELP	prints a list of options on the command line when MAP++ initializes
INNER_FTOL	inner loop function tolerance
INNER_GTOL	desired orthogonality between the function evaluations and Jacobian column
INNER_XTOL	inner loop consecutive iterate tolerance
INNER_MAX_ITS	maximum inner loop iterations
OUTER_MAX_ITS	maximum outer loop iterations
OUTER_TOL	outer loop tolerance
OUTER_EPSILON	Not used
INTEGRATION_DT	Not used
KB_DEFAULT	Not used
CB_DEFAULT	Not used
OUTER_CD	central difference Jacobian (outer loop solve only)
OUTER_BD	backward difference Jacobian (outer loop solve only)
OUTER_FD	forward difference Jacobian (outer loop solve only)
LM_MODEL	Not used
PG_COOKED	use the relaxation algorithm developed in *[4]*
KRYLOV_ACCELERATOR	use the Krylov accelerator algorithm developed in *[5]*
REPEAT	repeat the element/nodes defined in the input file by mirroring the mooring pattern with a rotation about the Z-axis
REF_POSITION	reference position

Todo: The REF_POSITION options is disabled in MAP++ until this feature can be fully integrated into the program. The reference position is fixed at $< 0, 0, 0 >$ until then.

4.6.1 Default Solver Options

Variable	Definition
INNER_FTOL	1.0E-6
INNER_GTOL	1.0E-6
INNER_XTOL	1.0E-6
INNER_MAX_ITS	500
OUTER_MAX_ITS	500
OUTER_TOL	1.0E-6
OUTER_EPSILON	1.0E-3
OUTER_BD	
REF_POSITION	<0.0 , 0.0 , 0.0>

CHAPTER 5

Python Example

5.1 Static Configuration

This example will be run against the *baseline input file*. The water depth is 350 meters. The space preceeding the `repeat 120 240` solver option flag is removed to enable duplication of the mooring geometry twice with 120° and 240° offsets about the Z axis.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

'''
Copyright (C) 2015
map[dot]plus[dot]plus[dot]help[at]gmail
License: http://www.apache.org/licenses/LICENSE-2.0
'''

from mapsys import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
np.set_printoptions(precision=2)
np.set_printoptions(suppress=True)


if __name__ == '__main__':
    mooring_1 = Map()

    mooring_1.map_set_sea_depth(350)       # m
    mooring_1.map_set_gravity(9.81)        # m/s^2
    mooring_1.map_set_sea_density(1025.0) # kg/m^3

    mooring_1.read_file("../test/baseline_2.map")
    mooring_1.summary_file('summary_file.txt')

    mooring_1.init( )
```

```
    epsilon = 1e-3 # finite difference epsilon
    K = mooring_1.linear(epsilon)
    print "\nLinearized stiffness matrix with 0.0 vessel displacement:\n"
    print np.array(K)

    surge = 5.0 # 5 meter surge displacements
    mooring_1.displace_vessel(surge,0,0,0,0,0)
    mooring_1.update_states(0.0,0)

    K = mooring_1.linear(epsilon)
    print "\nLinearized stiffness matrix with %2.2f surge vessel displacement:\n"
↪%(surge)
    print np.array(K)

    # We need to call update states after linearization to find the equilibrium
    mooring_1.update_states(0.0,0)

    line_number = 0
    H,V = mooring_1.get_fairlead_force_2d(line_number)
    print "Line %d: H = %2.2f [N]  V = %2.2f [N]"%(line_number, H, V)

    fx,fy,fz = mooring_1.get_fairlead_force_3d(line_number)
    print "Line %d: Fx = %2.2f [N]  Fy = %2.2f [N]  Fz = %2.2f [N]\n"%(line_number,␣
↪fx, fy, fz)

    print "These values come from the output buffer as defined in the 'LINE PROPERTIES
↪' portion of the input file"
    print "Labels : ", mooring_1.get_output_labels()[0:6]
    print "Units  : ", mooring_1.get_output_units()[0:6]
    v = mooring_1.get_output_buffer()[0:6]
    print "Values : ", ["{0:0.2f}".format(i) for i in v]

    fig = plt.figure()
    ax = Axes3D(fig)
    num_points = 20
    for i in range(0,mooring_1.size_lines()):
        x = mooring_1.plot_x(i, num_points)
        y = mooring_1.plot_y(i, num_points)
        z = mooring_1.plot_z(i, num_points)
        ax.plot(x,y,z,'b-')

    ax.set_xlabel('X [m]')
    ax.set_ylabel('Y [m]')
    ax.set_zlabel('Z [m]')

    plt.show()

    mooring_1.end( )
```

5.1.1 Output

Two outputs are produced executing the stript above. Information explicitly requested is printed to the command line:

```
MAP++ (Mooring Analysis Program++) Ver. 1.20.10 Mar-22-2016
MAP++ environment properties (set externally)...
```

```
    Gravity constant           [m/s^2]  : 9.81
    Sea density                [kg/m^3] : 1025.00
    Water depth                [m]      : 350.00
    Vessel reference position [m]       : 0.00 , 0.00 , 0.00

Linearized stiffness matrix with 0.0 vessel displacement:

[[  1.99e+04  -3.78e-03   5.19e-03  -4.89e-02  -2.00e+05  -1.77e-02]
 [  1.18e-03   1.99e+04  -1.06e-02   2.00e+05   3.50e-02  -6.17e-01]
 [  2.49e-03  -1.01e-03   2.27e+04   2.21e-03   2.23e-01  -2.12e-01]
 [  1.95e-03   2.00e+05  -7.14e-03   2.17e+08   1.10e-02  -5.23e+01]
 [ -2.00e+05   3.33e-04   4.85e-01  -4.89e-02   2.17e+08  -2.19e-02]
 [  8.83e-04  -5.59e-01   1.12e-03  -8.53e+01  -7.90e-02   1.41e+08]]

Linearized stiffness matrix with 5.00 surge vessel displacement:

[[  1.96e+04  -2.58e-05   1.17e+03   9.61e-03  -2.15e+05  -1.67e-01]
 [ -4.57e-04   2.07e+04   1.41e-03   1.81e+05  -5.24e-02   1.72e+03]
 [  1.17e+03  -3.32e-04   2.32e+04  -5.38e-03  -1.19e+04   1.12e-03]
 [  1.05e-03   2.00e+05   1.51e-03   2.17e+08   4.25e-02  -5.21e+01]
 [ -2.00e+05  -8.91e-05   4.79e-01   5.43e-03   2.17e+08   6.60e-02]
 [  2.10e-03  -5.60e-01   5.77e-03  -8.52e+01   1.07e-01   1.41e+08]]
Line 0: H = 597513.33 [N]  V = 1143438.75 [N]
Line 0: Fx = -597513.33 [N]  Fy = -0.00 [N]  Fz = 1143438.75 [N]

These values come from the output buffer as defined in the 'LINE PROPERTIES' portion␣
↪of the input file
Labels :  ['l[1]', 'alpha[1]', 'T[2]', 'l[4]', 'alpha[4]', 'T[5]']
Units  :  ['[m]', '[rad]', '[N]', '[m]', '[rad]', '[N]']
Values :  ['338.18', '1.07', '711942.60', '338.18', '1.07', '711942.39']
```

A figure is also produced to show the mooring geometry with a 5 meter vessel offset:

Note: The default units for the linearized stiffness matrix are [N/m], [N/rad], [Nm/m], and [Nm/rad]. See *the section on the linearized stiffness matrix* in the FAQ for more information.

5.2 Time-Marching for Dynamics Simulation

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

'''
Copyright (C) 2015
map[dot]plus[dot]plus[dot]help[at]gmail
License: http://www.apache.org/licenses/LICENSE-2.0
'''

from mapsys import *
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.ticker as mtick
from matplotlib import rcParams
import numpy as np
```

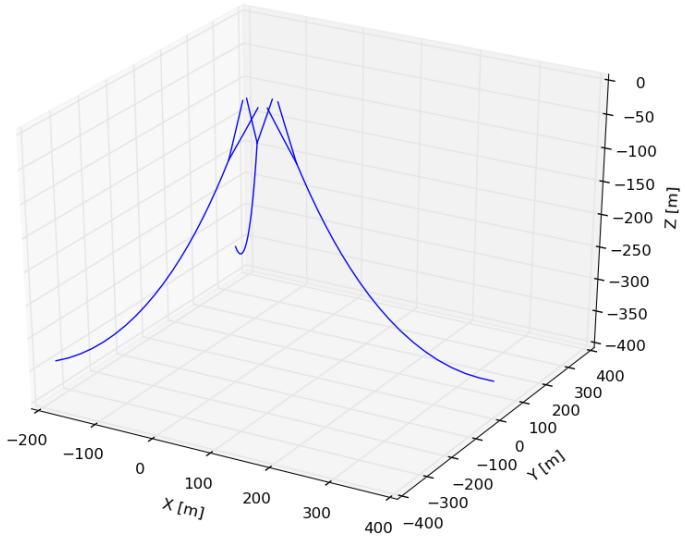


Fig. 5.1: Fig. 7

```
np.set_printoptions(precision=2)
np.set_printoptions(suppress=True)
rcParams.update({'figure.autolayout': True})


# user function to plot the mooring profile and footprint
def plot_mooring_system(mooring_data):
    # plot the mooring profile
    fig = plt.figure(1)
    ax = Axes3D(fig)
    colors = ['b','g','r','c']
    for i in xrange(mooring_data.size_lines()):
        x = mooring_data.plot_x( i, 20 ) # i is the the line number, and 20 is the␣
↪number of points plotted on the line
        y = mooring_data.plot_y( i, 20)
        z = mooring_data.plot_z( i, 20)
        ax.plot(x,y,z,colors[i]+'-')
    ax.set_xlabel('X [m]')
    ax.set_ylabel('Y [m]')
    ax.set_zlabel('Z [m]')


def start():
    """
    Step 1) First initialize an instance of a mooring system

    Step 2) Assume that (X, Y, Z, phi, theta, psi) are the translation and rotation␣
↪displacement of the vessel.
    These displacements are fed into MAP as an argument to displace the fairlead.␣
↪With the fairlead(s)
    rigidly connected to the vessel, the (X, Y, Z, phi, theta, psi) directly␣
↪manifests into the fairlead
    position in the global frame.

    For the time being, assume a sinusoidal displacement of the vessel
```

```
    Step 3) This for-loop emulates the time-stepping procedure. You want to loop␣
↪through the length of
    the arrays (X,Y,Z,phi,theta,psi) to retrieve the fairlead force

    Step 4) update the MAP state. The arguments in displace_vessel are the displace␣
↪displacements and rotations about the reference origin.
    In this case, the reference origin is (0,0,0).
    They can be set to a different potision using a run-time argument (this is an␣
↪advanced feature).

    Step 5) get the fairlead tension. The get_fairlead_force_3d returns the fairlead␣
↪force in
    X, Y Z coordinates. This must be called at each time-step, and then stored into␣
↪an array. We append
    the empty lists created on lines 84-88.

    .. Note::

       MAP does *NOT* return the mooring restoring moment, The user must calculate␣
↪this
       themself using the cross-product between the WEC reference origin and the␣
↪mooring attached
       point, i.e.,

       :math:`\mathbf{Moment} = \mathbf{r} \times \mathbf{F}`
    """

    # Step 1
    mooring = Map()
    mooring.map_set_sea_depth(120)         # m
    mooring.map_set_gravity(9.81)          # m/s^2
    mooring.map_set_sea_density(1020.0)    # kg/m^2
    mooring.read_file('../test/baseline_1.map')    # input file
    mooring.summary_file('summary_file.sum.txt') # output summary file name at the␣
↪conclusion of initialization

    mooring.init()                 # solve the cable equilibrium profile
    plot_mooring_system(mooring) # Optional: call the user function to illustrate the␣
↪mooring equilibrium profile


    # initialize list to zero (this is artificial. This would be prescribed the by␣
↪vessel program)
    X,Y,Z,phi,theta,psi = ([0.0 for i in xrange(500)] for _ in xrange(6))
    time = []

    # variable to specify the amplitude of surge oscillation and period factor
    dt = 0.1
    amplitude = 10.0

    # prescribe artificial surge and pitch displacement. Again, this should be␣
↪supplied based on the WEC motion or from time-marching routine
    for i in xrange(len(X)):
        time.append(i*dt)
        X[i] = (amplitude)*(math.sin(i*0.05))
        theta[i] = (amplitude)*(math.sin(i*0.025))
```

```
    # create an empty list of the line tension. We will store result from MAP in␣
↪these lists
    line1_fx, line1_fy, line1_fz = ([] for _ in xrange(3))
    line2_fx, line2_fy, line2_fz = ([] for _ in xrange(3))
    line3_fx, line3_fy, line3_fz = ([] for _ in xrange(3))
    line4_fx, line4_fy, line4_fz = ([] for _ in xrange(3))

    # Step 3) Time marching
    for i in xrange(len(X)):
        # Step 4)

        # displace the vessel, X,Y,X are in units of m, and phi, theta, psi are in␣
↪units of degrees
        mooring.displace_vessel(X[i], Y[i], Z[i], phi[i], theta[i], psi[i])

        # first argument is the current time. Second argument is the coupling␣
↪interval (used in FAST)
        mooring.update_states(time[i], 0)

        # Step 5)
        # line 1 tensions in X, Y and Z. Note that python is indexed started at zero
        fx, fy, fz = mooring.get_fairlead_force_3d(0) # arugment is the line number
        line1_fx.append(fx)
        line1_fy.append(fy)
        line1_fz.append(fz)

        # line 2 tensions in X, Y and Z.
        fx, fy, fz = mooring.get_fairlead_force_3d(1)
        line2_fx.append(fx)
        line2_fy.append(fy)
        line2_fz.append(fz)

        # line 3 tensions in X, Y and Z.
        fx, fy, fz = mooring.get_fairlead_force_3d(2)
        line3_fx.append(fx)
        line3_fy.append(fy)
        line3_fz.append(fz)

        # line 4 tensions in X, Y and Z.
        fx, fy, fz = mooring.get_fairlead_force_3d(3)
        line4_fx.append(fx)
        line4_fy.append(fy)
        line4_fz.append(fz)

    # Optional: plot the vessel displacement (surge=X and pitch=theta) as a function␣
↪of time
    plt.figure(2)
    plt.plot(time,X,lw=2,label='Surge displacement')
    plt.plot(time,theta,lw=2,label='Pitch displacement')
    plt.title('Vessel Translation/Rotation')
    plt.ylabel('Amplitude [m,deg]')
    plt.xlabel('Time [sec]')
    plt.legend()

    # Optional: plot line tension time history
    plt.figure(3)
    ax=plt.subplot(3,1,1)
    plt.plot(time,line1_fx,label='Line 1')
```

```
    plt.plot(time,line2_fx,label='Line 2')
    plt.plot(time,line3_fx,label='Line 3')
    plt.plot(time,line4_fx,label='Line 4')
    ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.2e'))
    plt.ylabel('X Fairlead Force [N]')
    plt.legend()

    ax = plt.subplot(3,1,2)
    ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.2e'))
    plt.plot(time,line1_fy)
    plt.plot(time,line2_fy)
    plt.plot(time,line3_fy)
    plt.plot(time,line4_fy)
    plt.ylabel('Y Fairlead Force [N]')

    ax = plt.subplot(3,1,3)
    ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.2e'))
    plt.plot(time,line1_fz)
    plt.plot(time,line2_fz)
    plt.plot(time,line3_fz)
    plt.plot(time,line4_fz)
    plt.ylabel('Z Fairlead Force [N]')
    plt.xlabel('Time [sec]')

    plt.show()

if __name__ == '__main__':
    start()
```

5.2.1 Output

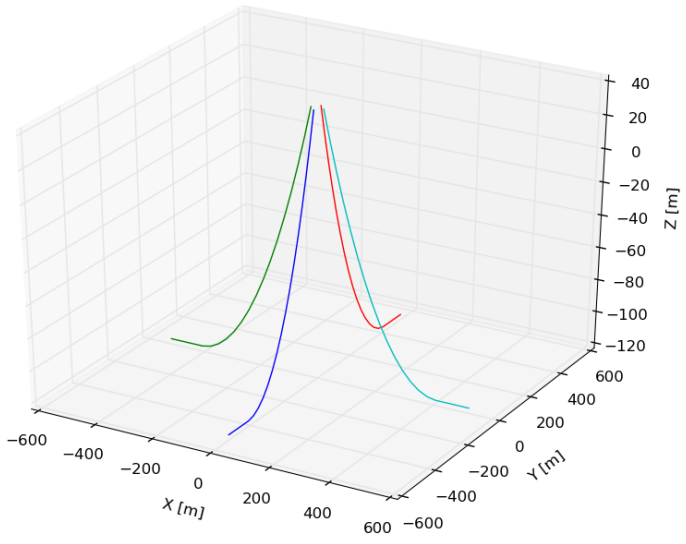


Fig. 5.2: Fig. 8

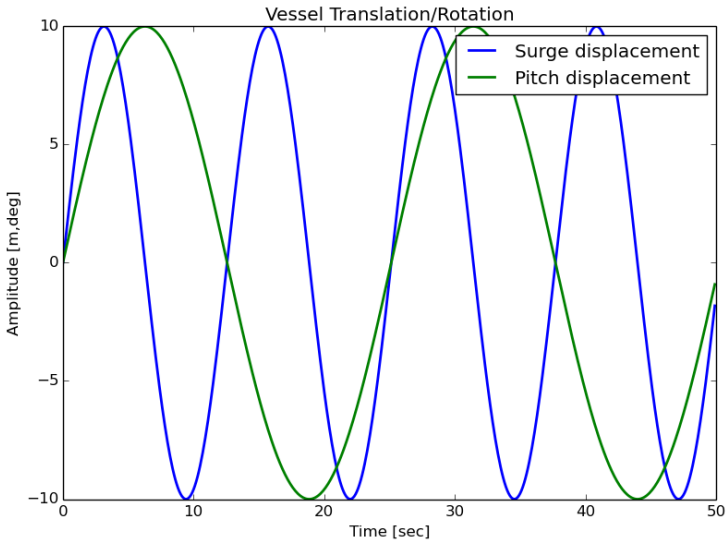


Fig. 5.3: Fig. 9

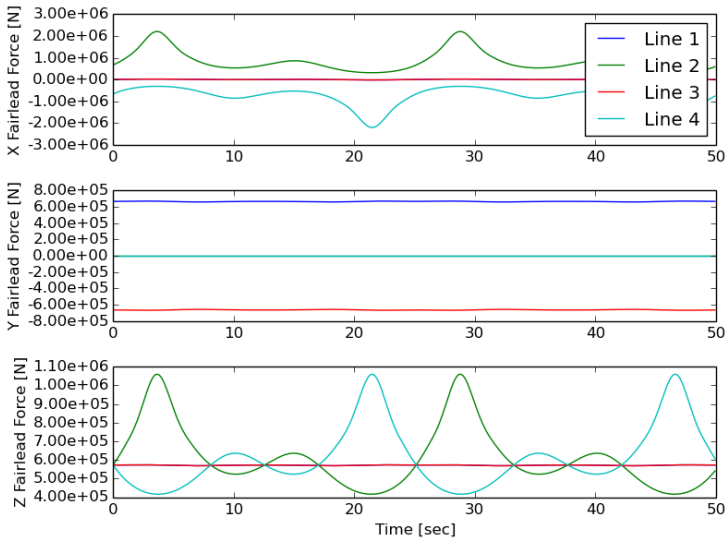


Fig. 5.4: Fig. 10

CHAPTER 6

API Documentation

6.1 Python API

CHAPTER 7

FAQ

7.1 Using with Python

7.1.1 Python doesn't load MAP

If importing MAP++ into Python leads to this error on Linux/OSx:

```
OSError: ../src/libmap-1.20.00.so: cannot open shared object file: No such file or␣
↪directory
```

or on Windows:

```
WindowsError: [Error 126] The specified module could not be found
```

then the issue is the MAP++ shared object/dll can't be located by the MAP++ module. This is corrected by changing the path where the .so/.dll is picked up in `mapsys.py`:

```
lib = cdll.LoadLibrary("/directory/to/map/libmap-1.20.00.so")
```

On Windows, it would look something like this:

```
lib = cdll.LoadLibrary("C:\User\local\directory\map_x64.dll")
```

If the error still persists, the solution could be one of the answers on this post at Stack Overflow.

7.2 Initialization Errors

7.2.1 I get WARNING [5] or FATAL [41] and can't emerge from initialization

The error thrown at initialization is either:

```
MAP_WARNING[5] : Cable density is approaching the density of seawater.
                 This may result in the problem becoming poorly conditioned.
```

or

```
MAP_FATAL[41] : Cable mass density is zero. Neutrally buoyant cables cannot
                be solved using quasi-statis model
```

We insert a check to warning the user if the cable is nearly neutrally buoyant. This causes the solver to go on the fritz, and sometimes fail, because the algebraic equation is close to dividing by zero; see *the horizontal cable equation*. The straightforward way to fix this is to change ω tolerance levels in `mapinit.c` The default tolerance levels are omega $\omega < 1.0$ for a warning, and omega $\omega < 10^{-3}$ for fatal. In some cases, this is unavoidable and a different mooring program is needed.

```
1     if (fabs(library_iter->omega)<=1.0) {
2       set_universal_error_with_message(map_msg, ierr, MAP_WARNING_5,
3                                        "omega = %f <= 1.0", library_iter->omega);
4     };
5   };
6   list_iterator_stop(&domain->library); /* ending the iteration "session" */
7
8   if (fabs(library_iter->omega)<=1.0E-3) {
9     return MAP_FATAL;
10  }
```

Todo: Include a run-time tolerance override option in the input file.

7.2.2 Maximum iterations are exceeded with taut mooring

The mooring system probably has a connect node. I'd start by increasing `OUTER_TOL` by an order of magnitude more than the *default option*. It doesn't take much displacement error in the inner solve to cause a large difference in the outer loop, so there's this constant game of playing catch-up. Alternatively, you can decrease the inner loop solve tolerance, but you might be already approaching machine precision. Taut lines are strange like that.

A good solver option/initial guess strategy should converge on a solution in under 100-500 total iterations in the first solve.

7.3 Running MAP++

7.3.1 What are the units of the linearized stiffness matrix?

The linearized stiffness matrix is a 6×6 entry comprised of four 3×3 blocks:

$$\mathbf{K}_{6\times6} = \begin{bmatrix} \mathbf{A}_{3\times3} & \mathbf{B}_{3\times3} \\ \mathbf{C}_{3\times3} & \mathbf{D}_{3\times3} \end{bmatrix}$$

The units are:

- [N/m] for $\mathbf{A}$
- [N/rad] for $\mathbf{B}$
- [Nm/m] for $\mathbf{C}$

- [Nm/rad] for $\mathbf{D}$

Note that the reference position is fixed at the *global origin*.

7.3.2 The linearized stiffness values are not consistent

This is likely attributed to round-off errors. We recommend testing linearized stiffness matrix entries for sensitivity against different epsilon values, tolerances, and finite differencing methods. The matrix entries should converge towards a set of values for those different options.

CHAPTER 8

Help

You can forward inquiries to the following:

map[dot]plus[dot]plus[dot]help[at]gmail[dot]com

CHAPTER 9

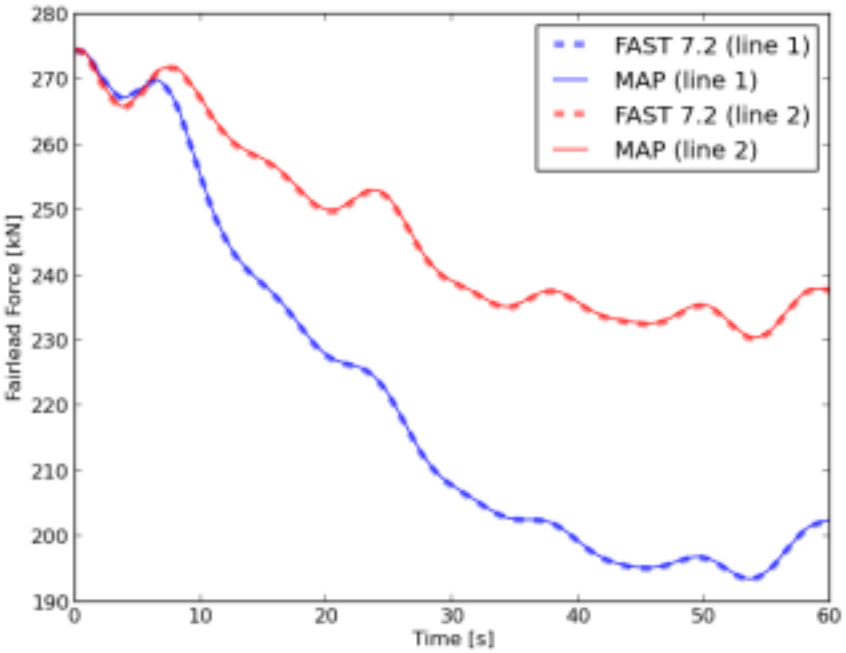
References

The Mooring Analysis Program is a library to model static loads and geometry of cables. MAP++ is designed to hook into other simulation codes through its API and can be customized to do a few things:

- Prototype a design
- Find the force-displacement relation for a given footprint
- Integrate into other dynamic simulation programs to produce a nonlinear restoring force time history

A quick–start guide is available *here*. We integrated MAP++ into other programs written in Python, C, C++, and Fortran. MAP++ follows the FAST Offshore Wind Turbine Framework *[2]* pattern.

More information on the theory behind MAP++ is described *[3]*. MAP++ is licensed under Apache version 2.


FAST 7.2 (line 1)
MAP (line 1)
FAST 7.2 (line 2)
MAP (line 2)
Fairlead Force [kN]
Time [s]
280
270
260
250
240
230
220
210
200
190
0
10
20
30
40
50
60

Bibliography

[1] H.M. Irvine. *Cable structures*. Volume 5. Dover Publication,s New York, NY, 1992.

[2] J.M. Jonkman. The new modularization framework for the fast wind turbine cae tool. In *51st AIAA Aerospace Sciences Meeting and 31st ASME Wind Energy Symposium, Grapevine, Texas*. 2013.

[3] M., Masciola, J., Jonkman, and A. Robertson. Implementation of a multisegmented, quasi-static cable model. In *The Twenty-third International Offshore and Polar Engineering Conference*. International Society of Offshore and Polar Engineers, 2013.

[4] A.H., Pevrot and A.M. Goulois. Analysis of cable structures. *Computers & Structures*, 10(5):805–813, 1979.

[5] M. H., Scott and G. L. Fenves. Krylov subspace accelerated newton algorithm: application to dynamic progressive collapse simulation of frames. *Journal of Structural Engineering*, 2009.