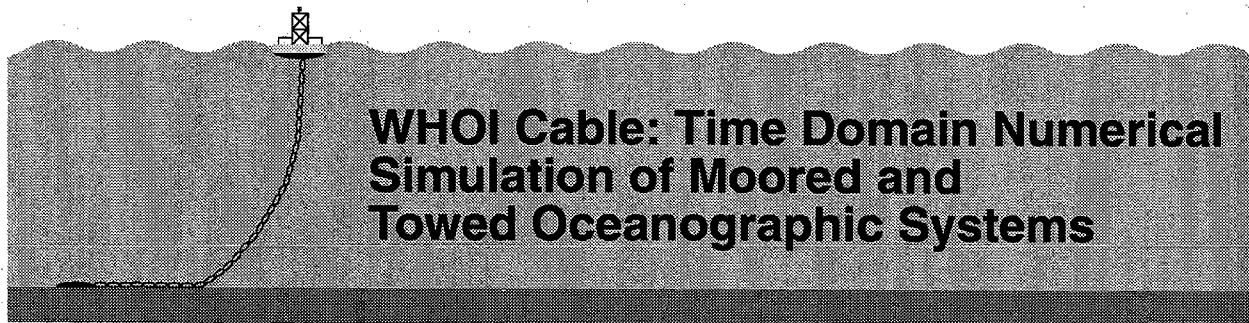
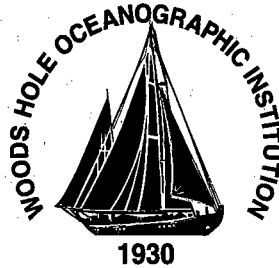


WHOI-97-15

Copy 2

Woods Hole Oceanographic Institution



WHOI Cable: Time Domain Numerical Simulation of Moored and Towed Oceanographic Systems

by

Jason I. Gobat, Mark A. Grosenbaugh, and Michael S. Triantafyllou

November, 1997

Technical Report

Funding was provided by the Office of Naval Research under
Contract Nos. N00014-92-J-1269 and N00014-95-1-0106

Copyright ©1997 by Woods Hole Oceanographic Institution. All rights reserved.

WHOI-97-15

WHOI Cable: Time Domain Numerical Simulation of Moored and Towed Oceanographic Systems

by

Jason I. Gobat, Mark A. Grosenbaugh, and Michael S. Triantafyllou

Woods Hole Oceanographic Institution
Woods Hole, MA 02543

November, 1997

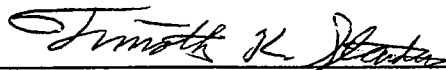
Technical Report

Funding was provided by the Office of Naval Research through grants N00014-92-J-1269
and N00014-95-1-0106 and an Office of Naval Research Graduate Fellowship.

Reproduction in whole or in part is permitted for any purpose of
the United States Government. This report should be cited as:
Woods Hole Oceanographic Institution Technical Report WHOI-97-15.

Copyright ©1997 by Woods Hole Oceanographic Institution. All rights reserved.

Approved for Distribution:



Timothy K. Stanton, Chairman
Department of Applied Ocean Physics
and Engineering



Contents

Foreword	9
About this Manual	9
Acknowledgements	9
Typographical Conventions	10
1 Introduction	11
1.1 Overview of problem types	11
1.2 <i>WHOI Cable</i> mathematical features	12
1.3 <i>WHOI Cable</i> implementation features	12
2 Mathematical and Numerical Theory	13
2.1 General numerical approach	13
2.2 Numerical details of static problems	15
2.2.1 Boundary conditions	15
2.2.2 Initialization	17
2.2.3 Coordinate integration	18
2.2.4 Bottom interaction	18
2.3 Numerical details of dynamic problems	18
2.3.1 Wave forcing	19
2.3.1.1 Wave followers	19
2.3.1.2 Morison's equation	19
2.3.1.3 Froude-Krylov forcing model	20

2.3.2	Coordinate integration	20
2.3.3	Bottom interaction	21
2.3.4	Dynamic pay-in and pay-out of cable	21
2.4	Equations of motion	22
2.4.1	Two-dimensional problems	22
2.4.1.1	Static equations	22
2.4.1.2	Dynamic equations	23
2.4.2	Three-dimensional problems	23
2.4.2.1	Static equations	23
2.4.2.2	Dynamic equations	24
2.5	Coordinate transformations	26
2.5.1	Two-dimensional	26
2.5.2	Three-dimensional	27
3	Structure of a <i>cable</i> Problem	29
3.1	Notation and coordinate systems	29
3.2	Basic language features	29
3.2.1	Expressions	30
3.2.1.1	Continuous functions	30
3.2.1.2	Discrete functions	31
3.2.2	Units	31
3.3	Components of an input file	31
3.3.1	Problem description	32
3.3.2	Analysis parameters	32
3.3.3	Environmental parameters	35
3.3.4	Cable, chain and rope materials	37
3.3.5	Connectors	38
3.3.6	Buoys	39
3.3.7	Anchors	40

3.3.8	System layout	40
3.3.9	The end statement	45
3.4	Tips and tricks	45
3.4.1	Static problems	45
3.4.2	Dynamic problems	46
4	The <i>cable</i> Application	47
4.1	Basic operation	47
4.2	Using the run-time solution controls	48
4.3	Using the C pre-processor	49
4.4	Summary of command line parameters	50
4.5	Interpreting the output from <i>cable</i>	53
5	Post-processing <i>cable</i> Results	55
5.1	Using <i>cable</i> results with Matlab	55
5.1.1	Format of the Matlab file	55
5.1.2	Example Matlab manipulations	56
5.1.3	<i>res2mat</i> command line parameters	57
5.2	The <i>animate</i> post-processing application	57
5.2.1	The main animation window	58
5.2.2	Coordinates and zooming	60
5.2.3	Animate command line parameters	60
5.3	ASCII output	63
5.3.1	<i>res2asc</i> command line parameters	63
6	<i>cable</i>'s Windows Interface	65
6.1	Introduction	65
6.2	Building an input file	65
6.3	Solving a problem	67
6.4	Viewing and converting results	69

6.5	Working with files	70
6.6	Command reference	70
6.7	Installing <i>WHOI Cable</i> for Windows	71
6.7.1	System requirements	71
6.7.2	Installation instructions	72
6.7.3	Printing from <i>animate</i> under Windows	72
6.7.4	Modifying the installation	73
6.7.4.1	File and pathnames	73
6.7.4.2	Templates	73
A	Subsurface Mooring Example	75
B	Shallow Water Surface Mooring	79
C	Deep Water S-tether Mooring	83
D	Horizontal Array Mooring	89
E	Towed Vehicle Example	95
	References	99

List of Figures

2.1	Local and global two-dimensional coordinate systems.	26
3.1	Geometric definitions for <i>cable</i>	30
4.1	<i>cable</i> 's graphical information and control dialog.	49
4.2	The binary file format for <i>cable</i> results files.	54
5.1	The main window of <i>animate</i>	58
5.2	A time plot of the forces at two marked nodes.	59
6.1	The relationships between the <i>WHOI Cable</i> component programs.	66
6.2	The main window of the <i>WHOI Cable</i> Windows interface.	66
6.3	The solution control dialog in the <i>WHOI Cable</i> Windows interface.	68
6.4	The results control dialog in the <i>WHOI Cable</i> Windows interface.	69
6.5	The setup dialog used to configure <i>WHOI Cable</i> pathnames.	73
A.1	Example result from <i>animate</i> for a subsurface mooring.	77
A.2	A plot of the time history of forces for the subsurface mooring example.	77
B.1	The static configuration of the surface mooring example problem.	81
B.2	The time history of total tension for the surface mooring example.	82
C.1	Static and dynamic results for an S-tether mooring.	87
D.1	Animation result for the horizontal mooring example problem.	93
D.2	Motion records for nodes on the horizontal mooring.	93

E.1	Steady-state configuration of the towing example.	97
E.2	Depth profile of the tow sled during a tow-yow maneuver.	98

List of Tables

5.1	The names that <i>res2mat</i> assigns to Matlab variables.	56
6.1	Complete command structure for the <i>WHOI Cable</i> for Windows encapsulator.	71

Foreword

About this Manual

This report documents **version 1.0** of *WHOI Cable*. While it is our intention to provide up-to-date, comprehensive, accurate documentaion, *WHOI Cable* is very much a work in progress and as such undergoes frequent change. If you find something that behaves differently than the way this document says it should behave then please let us know.

This report is presented largely as a user's guide for *WHOI Cable*. We generally provide technical algorithmic details only to give the user a loose understanding of how problems are solved or in places where the information is not published elsewhere. User should consult the references listed in the bibliography for additional details (particularly [4, 12]).

Acknowledgements

Though the current implementation of *WHOI Cable* is a relatively recent development, it does owe much to several pieces of work completed over the last few years. The time domain simulation algorithm is based on code originally developed by Christopher Howell for his Ph.D. thesis [4] and modified by Thanassis Tjavaras for his Ph.D. thesis [12].

The funding for the development of *WHOI Cable* has been provided by the Office of Naval Research through grant numbers N00014-92-J-1269 at WHOI and N00014-95-1-0106 (ONR Code 321, Ocean Engineering and Marine Systems Program) and an Office of Naval Research Graduate Fellowship.

WHOI Cable is copyright ©1997 by Woods Hole Oceanographic Institution. *WHOI Cable* is proprietary software; free redistribution of *WHOI Cable* software is not permitted.

Matlab is a trademark of The Mathworks, Inc. Pentium and Pentium Pro are trademarks of Intel Corporation. Windows 95 and Windows NT are trademarks of Microsoft Corporation.

Typographical Conventions

This report employs a number of typographical conventions to mark buttons, command names, menu options, screen interaction, etc.

Bold Font Used to mark **buttons**, and **menu options** in graphical environments.

Italics Font Used to indicate an application program name, e.g. *res2mat*.

Typewriter Font

Used to represent screen interaction at the shell prompt. Also used for example input files, and keywords that belong in input files.

Key Represents a key (or key combination) to press, as in press Return to continue.

Chapter 1

Introduction

1.1 Overview of problem types

The types of systems that we classify as oceanographic mooring systems include simple tethered buoys, towed and drifting systems, and complex strings of instrumentation suspended in deep water. From an engineering design perspective it is important that we can predict how these systems will respond to a variety of environmental factors, particularly waves and current. We might want to know just how much current it will take to pull a surface buoy under water or what the maximum tension will be in a mooring line during a large storm. The scientific purposes of a system might require that the motion of a particular instrument not exceed a certain level in typical operating conditions. The unifying problem behind analyzing these kinds of systems is one of nonlinear cable mechanics.

Typical oceanographic mooring systems consist of rope, wire, and chain connected together by shackles, instruments, and buoys and terminated at the ends with buoys, ships, sinker weights, or anchors. *WHOI Cable* is a collection of computer programs for cable mechanics designed specifically to solve this nonlinear problem for systems which can be defined in these terms and which fit into one of several basic categories.

For traditional single point moorings *WHOI Cable* (sometimes referred to by the name of the primary application program, *cable*) can solve subsurface and both taut and slack surface moorings. The system can consist of any combination of different cable/chain/rope segments in series with one another. Instruments, floats, and connectors between segments are treated as lumped masses (i.e., the rigid body dynamics of an in-line instrument are not modeled, but the mass, weight and drag effects of the instrument are considered). *cable* can also solve problems in which both ends of the mooring are anchored on the bottom and towing/drifter problems in which the subsurface end is unconstrained and the surface

end is free to move either under the influence of current (drifters) or with a prescribed (possibly time varying) speed (towing). Towing problems can also include the effects of time varying pay-in and pay-out rates. In all cases *cable* can produce solutions in either two or three dimensions and can solve either the static (steady-state) problem given forcing by current or the dynamic problem given forcing by both current and waves.

1.2 *WHOI Cable* mathematical features

WHOI Cable is built around a mathematical model that includes the effects of arbitrary geometric nonlinearities, material nonlinearities, material bending stiffness, and material torsion. Including geometric nonlinearities and bending stiffness means that *WHOI Cable* can accurately model systems in which cable segments go slack. The nonlinear, one-sided boundary condition at the seabed is modeled as an elastic foundation for systems with cable lying on the bottom. The numerical implementation includes an adaptive time stepping algorithm to speed the solution of problems with high nonlinearity.

1.3 *WHOI Cable* implementation features

WHOI Cable is a suite of applications, all of which are centered around the primary solver program, *cable*. *cable* is responsible for processing user input files and generating results for all of the various problem types. Input files are constructed using an intuitive, object based syntax. This high-level syntax allows for the use of symbolic expressions in assignment statements, the re-use of object descriptions that may be stored in central database files, and a largely free-form construction of input files. It also facilitates detailed error reporting.

Results can be post-processed either by converting them to Matlab format with *res2mat* or by viewing them with the graphical application *animate*. *animate* provides a simple environment for viewing system configurations in both two- and three-dimensions and for generating graphs of result variables. Spectra of time domain results are also available.

WHOI Cable for Windows includes an encapsulator application that allows for control of all of the component programs from within a familiar Windows style interface. This interface gives the user total push button control of the various options for solving problems and analyzing results.

Chapter 2

Mathematical and Numerical Theory

2.1 General numerical approach

For all combinations of boundary conditions, 2D or 3D and static or dynamic problems, the mathematical problem is posed as a system of coupled, nonlinear partial differential equations. *cable* solves these systems numerically by discretizing the continuous (exact) forms of these governing equations onto a grid of nodes on which it will calculate an approximate solution. As the grid becomes finer and finer the approximate solution will approach the exact solution. The cost of these finer discretizations which buy better solutions is an increase in computation time.

Both the static and the dynamic cable problems can be generalized as a system of N first-order nonlinear partial differential equations

$$\frac{\partial \vec{Y}}{\partial s} + \vec{F}(s, \vec{Y}) = 0, \quad (2.1)$$

where \vec{Y} is the vector of the N dependent variables, s is the position variable along the cable (the Lagrangian coordinate), and \vec{F} is a vector of functions that depends on the form of the governing equations. For example, in the 2D static problem (the simplest of all possible cases). equation 2.1 represents four equations in four unknowns: strain (from which we can always derive tension via a constitutive relationship), shear force, inclination angle, and curvature. This equation is discretized at the n nodal points using centered finite differences written on the half-grid points (which makes the differences second order accurate [12, 13]). At node k the discretized result is

$$\vec{Y}_k - \vec{Y}_{k-1} + \frac{s_k - s_{k-1}}{2} (\vec{F}_k + \vec{F}_{k-1}) = 0. \quad (2.2)$$

Equation 2.2 represents an $N \times n$ system of coupled, nonlinear equations which *cable* solves using a Newton-Raphson like relaxation technique [8].

Equation 2.2 can only be satisfied by an exact solution for \vec{Y}_k . Given an inexact first guess at this solution, \vec{Y}_k^0 , we need to develop an iterative scheme to calculate successively better approximations, \vec{Y}_k^i , through a series of update vectors, $\Delta\vec{Y}_k^i$, such that

$$\vec{Y}_k^{i+1} = \vec{Y}_k^i + \Delta\vec{Y}_k^i \quad (2.3)$$

where \vec{Y}_k^{i+1} brings us closer to satisfying the equality in equation 2.2. In quantitative terms we want to iteratively minimize the error function

$$\vec{e}_k^i(\vec{Y}_k^i, \vec{Y}_{k-1}^i) = \vec{Y}_k^i - \vec{Y}_{k-1}^i + \frac{s_k - s_{k-1}}{2} (\vec{F}_k^i + \vec{F}_{k-1}^i). \quad (2.4)$$

Neglecting for clarity the dependence on the previous nodal point ($k-1$), we can very loosely write

$$\frac{[\vec{e}_k^{i+1}(\vec{Y}_k^i + \Delta\vec{Y}_k^i) - \vec{e}_k^i(\vec{Y}_k^i)]}{\Delta\vec{Y}_k^i} \approx \frac{\partial \vec{e}_k}{\partial \vec{Y}_k}. \quad (2.5)$$

The derivatives on the right hand side of equation 2.5 can be calculated analytically from the known form of the discretized governing equations (equation 2.4). If we were to re-insert the dependence on \vec{Y}_{k-1} , we would note that these derivatives actually constitute an $N \times 2N$ Jacobian matrix at each k (the matrix is composed of the derivatives of the N equations with respect to the $2N$ variables represented by \vec{Y}_k and \vec{Y}_{k-1}). We can assemble the Jacobian matrices from each node into a single global Jacobian matrix (much like element stiffness matrices are assembled into global stiffness matrices in the finite element method), add boundary condition information and formulate a linear system that will find $\Delta\vec{Y}_k^i$ to drive the updated error, \vec{e}_k^{i+1} , to zero. If J^i is this global Jacobian matrix evaluated at \vec{Y}^i then we see from equation 2.5 that

$$J^i \Delta\vec{Y}^i = -\vec{e}^i. \quad (2.6)$$

Because only two nodes (k and $k-1$) are coupled by each individual Jacobian matrix the assembled global Jacobian matrix in equation 2.6 will be very sparse, with the only non-zero entries clustered near the main diagonal. *cable* takes advantage of this sparsity in solving equation 2.6 by using a sparse Gaussian Elimination algorithm, NSPIV, due to Sherman [10]. Sparse algorithms such as NSPIV exploit sparsity to reduce both memory requirements and computation time (normal Gaussian elimination is an $O(n^3)$ operation, sparse algorithms can be as efficient as $O(n)$).

The actual update to \vec{Y} is scaled by a relaxation factor ω

$$\vec{Y}^{i+1} = \vec{Y}^i - \omega \Delta\vec{Y}^i. \quad (2.7)$$

The purpose of this relaxation factor is to slow (under-relax) the update in cases where strong nonlinearities may mean that the update is not quite as robust as we would like. For highly nonlinear problems, where small changes in parameters can mean big changes in system configuration, the approximation of equation 2.5 becomes less valid and our update $\Delta\vec{Y}^i$, if fully applied ($\omega = 1$), may actually increase the total system error. A small relaxation factor increases the accuracy of the linearized Taylor series expansion that equation 2.5 represents. By slowing the process down ($\omega < 1$) the movement of the system from iteration to iteration towards equilibrium will be smoother because the steps between iterations will be smaller.

The iterative updates of \vec{Y} continue until the update vectors, $\Delta\vec{Y}$, become smaller than a user defined convergence value. Given the update vector

$$\Delta\vec{Y}_k = [\Delta Y_{1,k} \cdots \Delta Y_{N,k}] \quad (2.8)$$

at each node k , the convergence condition is

$$\frac{1}{N} \sum_{i=1}^N \left[\frac{1}{n\bar{Y}_i} \sum_{k=1}^n |\Delta Y_{i,k}| \right] < \text{tolerance.} \quad (2.9)$$

\bar{Y}_i in the above are a set of canonical values that express the typical order of magnitude of the variable represented by Y_i . A canonical value for strain, for example, is 0.01.

2.2 Numerical details of static problems

2.2.1 Boundary conditions

Static boundary conditions for the various problem types can typically be described by an anchor restraint at the first end and an applied static force at the opposite, free end. The simplest case is a user prescribed force vector applied at the free end (general problems, see section 3.3.1). With a force that is known a priori, *cable* can generate a solution with one set of iterations directly. The problem is similar for subsurface moorings because *cable* can directly calculate the buoyancy and drag forces on the completely submerged buoy at the free end of the system. Static (steady-state) towing problems can also be solved this way by fixing the position of the ship and applying weight and drag forces to the towed-body end of the system. The drag on the tow body and cable is generated both by the environmentally applied current and by an artificially superposed current that is equal in magnitude and opposite in direction to the steady-state tow speed.

In systems with a buoy on the free surface the problem is more complicated because we do not know the forces at the buoy end before solving the problem. Vertical and horizontal

forces applied by the buoy on the cable segment under the buoy are a function of the buoy draft and the known buoyancy and drag properties of the buoy. *cable* begins the solution with forces calculated from an initial guess of the draft (equal to the maximum available draft), H_g^0 . We then calculate the actual draft, H_s^0 , for these forces. The absolute error is

$$e_H^0 = H_s^0 - H_g^0 \quad (2.10)$$

If this error is positive then we can tell immediately that the buoy does not have sufficient buoyancy to come to the surface. If the error is negative then we know that we need less buoyancy and we proceed to make a series of guesses

$$H_g^{i+1} = \beta H_g^i \quad (2.11)$$

until we get a solution such that

$$e_H^i = H_s^i - H_g^i \quad (2.12)$$

is positive. With the actual solution now bracketed between H_g^0 and H_g^i , we proceed to use a regula falsi root finding technique [2] to home-in on a final solution. This root finding procedure forms a second, outer loop of iterations. At each new guessed draft we must go through a new series of iterations to solve the problem. This inner loop of iterations is the process of finding the equilibrium position for a given applied static force based on the current best guess at the draft. Note that β in equation 2.11 is an outer iteration “relaxation” factor that controls how fast we search for the minimum draft that brackets the real draft. It should always be smaller than unity (it defaults to 0.95). Making it too small can result in singularities because very small drafts equate to forces which may not be large enough to support the weight of the system.

The idea of outer loop iterations is also used for problems with both ends anchored on the bottom (see example, Appendix D). In this case, we do not know a priori the reaction forces at the second anchor. Given the position of the second anchor, however, we can perform outer loop iterations by changing the applied force at the second end with each outer iteration, until that second end is brought to its actual known position. The adjusted applied force at each outer iteration is calculated from

$$\vec{F}^{k+1} = \vec{F}^k - \beta (\vec{X}^k - \vec{X}) \quad (2.13)$$

where \vec{F}^k is the applied force vector at outer iteration k , \vec{X}^k is the calculated position of the second anchor at outer iteration k and \vec{X} is the desired position of the second anchor. β is a “stiffness” factor defaults to 5.

The final type of problem that requires outer loop iterations is drifting systems. In principle, solving a drifter problem requires the same boundary conditions as a towing

problem. The situation is more complicated for drifters, however, because for a complex current profile we do not know the steady-state drift speed of the system. At steady-state, the sum of the integrated drag force in the horizontal directions must be zero,

$$\int_0^L \frac{1}{2} \rho C_{dx}(s) S(s) |U_c(s) - \bar{U}| (U_c(s) - \bar{U}) ds = 0, \quad (2.14)$$

where the integrand represents the drag force in a horizontal direction as a function of position due to a relative velocity that is the value of the current at that position minus the steady-state drift speed, \bar{U} . For $U_c(s)$ constant, $\bar{U} = U_c$ satisfies this constraint and we know that the system will drift with the current speed. For $U_c(s)$ not constant, however, it is clear that at some points on the system \bar{U} must be less than $U_c(s)$ and at other points it must be greater than $U_c(s)$. For drifting systems, the outer loop of iterations determines \bar{U} such that equation 2.14 is satisfied.

The initial value of \bar{U} is set to 105% of the maximum current speed (setting it to 100% of the current speed leads to numerical problems because there may be no resultant horizontal drag for cases of constant current). The outer loop iteration procedure then uses the same regula falsi root finding scheme as discussed above to find the actual drift speed which must be bracketed between the maximum current value and zero. The calculated speed at the end of each iteration is determined from the drag force that is required to balance the tension and shear forces at the top of the system. The convergence of the procedure is based on the absolute relative difference between this calculated speed and the guessed speed that was used to begin the iteration.

2.2.2 Initialization

With the system discretized according to user inputs, the first step in solving a problem is to calculate a zero order approximate solution based on an inextensible catenary with no bending stiffness. This solution provides the initialization for the iterative scheme outlined in section 2.1 For multi-segment problems the catenary solution is based on a single equivalent stiffness and weight. The equivalent unit weight is calculated by summing the total wet weight of all components in the system (cable segments, connectors, and attachments) and dividing by the total length of all cable segments. The equivalent stiffness is found by adding all cable segments together as simple linear springs in series and then dividing by total length.

2.2.3 Coordinate integration

Because the global coordinate variables x, y, z do not appear in any of the governing equations, they are simply integrated based on cable coordinates and cable orientation after each iteration. While the coordinates do not figure directly into the governing equations it is important that they be updated because they are used in evaluating the current at a node, determining if a node is lying on the bottom, and fixing the position of the top node to determine the draft of a buoy.

The first node is always located at the origin. In 2D the position of any subsequent node, k , is then calculated from

$$z_k = z_{k-1} + \Delta s_{k-1} \cos \phi_k (1 + \epsilon_k), \quad (2.15)$$

$$x_k = y_{k-1} + \Delta s_{k-1} \sin \phi_k (1 + \epsilon_k). \quad (2.16)$$

Δs_{k-1} is the spacing between nodes k and $k-1$. ϵ_k and ϕ_k are the strain and tilt angle at node k . A similar form applies in 3D, with the sin and cos terms replaced by appropriate functions of the four Euler parameters (see section 2.5).

2.2.4 Bottom interaction

For problems with cable segments that may be lying on the bottom, *cable* models the sea bed as an elastic mattress with a linear spring. If z is the vertical coordinate of a node along the cable and $z < 0$ (where 0 is the vertical position of the sea floor) then the vertical reaction force applied by the bottom at that point is

$$F_s = k |z|, F_s \leq w_0 \quad (2.17)$$

where w_0 is the wet weight per unit length of the cable at that node and k is a parameter describing the stiffness of the bottom. This representation is reasonably smooth and has the advantage that it enforces a limit on the force exerted by the bottom (it cannot exceed the weight of the cable). The smoothness is important to avoid the abrupt changes in configuration that can occur between iterations in systems with high nonlinearity. The bottom boundary condition, being one-sided, is necessarily very nonlinear.

2.3 Numerical details of dynamic problems

The solution of dynamic problems proceeds by applying the same iterative scheme at each time step. With the continuous form of the governing equations written as

$$\frac{\partial \vec{Y}}{\partial s} + \mathbf{M}(\vec{Y}) \frac{\partial \vec{Y}}{\partial t} + \vec{F}(s, \vec{Y}) = 0 \quad (2.18)$$

we can use backward finite differences in time for the time derivatives and write a discrete form of the governing equations that is of the same basic form as 2.2 because we know the solution at the previous time step. M in equation 2.18 is an $N \times N$ matrix that depends on the form of the governing equations. The initial guess for the solution at each new time step is simply the solution from the previous time step.

There are limits to the maximum allowable time step that can be used to propagate the solution in time without giving rise to numerical instabilities. *cable* does have an adaptive time stepping algorithm whereby if an instability arises the time step will be automatically reduced to try to get through that portion of the simulation. At each time step where the baseline time increment is not small enough to accurately propagate the solution, *cable* will reduce the increment by a factor of ten and take ten steps at the smaller increment. It will descend as low as five orders of magnitude from the baseline increment before giving up.

Adaptive time stepping is only of limited usefulness, however, without some care being taken in the choice of a baseline time increment. If the program is deciding that it needs a smaller time increment at every step then it would be faster to have set a smaller time step in the first place (rather than wasting computational resources at each time step deciding that a smaller increment is necessary).

2.3.1 Wave forcing

2.3.1.1 Wave followers

The dynamic excitation for wave following surface buoys is the simplest of the forcing models. For wave following buoys the model is forced by matching the vertical velocity at the free (buoy) end to the vertical velocity of the incident wave. In 2D

$$u_n = U_s \cos(\phi_n) \quad (2.19)$$

$$v_n = U_s \sin(\phi_n) \quad (2.20)$$

where u_n, v_n are the tangential and normal velocities at the topmost node, ϕ_n is the inclination from vertical at the topmost node, and U_s is the vertical surface velocity. There is no imposed horizontal component of motion with this representation. Prescribed motion with both horizontal and vertical components can be imposed using the `velocity` forcing method (see section 3.3.3).

2.3.1.2 Morison's equation

Morison's equation provides a convenient way to model the hydrodynamic, wave-induced loads on ocean structures by linearly superposing the solutions to a viscid and an invis-

cid problem [6, 1]. For *cable* problems it is most appropriate for modeling the forces on subsurface buoys.

In inviscid theory we can derive a force term that is due to the wave induced inertia of the fluid surrounding the buoy. This force can be written as

$$\vec{F}_I = \rho \nabla (1 + C_M) \frac{\partial \vec{U}_w}{\partial t} \quad (2.21)$$

where C_M is an added-mass coefficient ($= 0.5$ for a sphere) and \vec{U}_w is the velocity of the water particles under the waves. The viscous portion of the force is a drag term based on an experimentally derived drag coefficient, C_D , and the relative velocity of the buoy through the water, \vec{U}_R ,

$$\vec{F}_V = \frac{1}{2} \rho C_D S \vec{U}_R |\vec{U}_R|. \quad (2.22)$$

S is the projected area of the buoy.

2.3.1.3 Froude-Krylov forcing model

The forcing model that *cable* uses for surface buoys that are not wave followers is described in [3] and derived in detail in [7]. It combines a Froude-Krylov force (calculated by integrating the dynamic pressure of the wave field over the surface of the buoy) with the Haskind relations to calculate the wave exciting and damping forces. The Haskind relations calculate a wave damping coefficient that is proportional to the square of the wave exciting force.

2.3.2 Coordinate integration

In a dynamic problem the coordinate integration given in equations 2.15 and 2.16 must be modified slightly in towing problems to account for the possibility that the first node may have moved and/or that cable may be paying out at the top node. At time step i , the integration begins by fixing the position of the topmost node

$$z_n^i = z_n^{i-1} + (u_n^i \cos \phi_n^i - v_n^i \sin \phi_n^i) dt, \quad (2.23)$$

$$x_n^i = x_n^{i-1} + (u_n^i \sin \phi_n^i + v_n^i \cos \phi_n^i) dt, \quad (2.24)$$

where u_n^i, v_n^i are the tangential and normal velocities of the topmost node at time step i . Integrating from the top down then, the coordinates for subsequent nodes, k , are calculated from

$$z_k = z_{k+1} - \Delta s_k \cos \phi_k (1 + \epsilon_k), \quad (2.25)$$

$$x_k = y_{k+1} - \Delta s_k \sin \phi_k (1 + \epsilon_k). \quad (2.26)$$

For other types of problems in which the first node always remains fixed at the anchor (and thus at the origin), equations 2.15 and 2.16 still hold.

2.3.3 Bottom interaction

The bottom boundary condition for the dynamic problem is modeled slightly differently than for the static problem. Because impact forces can cause the sea floor to exert a reaction force greater than the weight of the cable, the dynamic vertical reaction force of the elastic foundation is modeled as a simple linear spring/dashpot combination,

$$F_s = k |z| \quad (2.27)$$

for nodes with $z < 0$. If the bottom is sufficiently stiff (k large) or z is sufficiently negative than the force per unit length can easily exceed w_0 . Bottom damping is simulated by adding a velocity proportional damping term,

$$F_d = b \dot{z}. \quad (2.28)$$

b is the dimensional damping coefficient as calculated from

$$b = 2\zeta (m + m_a) \omega_n, \quad (2.29)$$

where ζ is the damping ratio ([11, 9]) of the bottom, $m + m_a$ is the mass plus added mass per unit length of the cable segment, and ω_n is the natural frequency of the elastic foundation/cable segment system

$$\omega_n = \sqrt{\frac{k}{m + m_a}}. \quad (2.30)$$

2.3.4 Dynamic pay-in and pay-out of cable

Changes in the length of the top cable segment during the course of a dynamic simulation are handled through the addition or deletion of nodes. Given a pay-rate, $P(t)$, the current time step, Δt , and an initial node spacing at the top of the system, Δs , the amount of cable being added or removed from the system at the current time step, t^i , is

$$L_a = P(t^i) \Delta t. \quad (2.31)$$

If $|L_a| > \Delta s$ then the number of nodes to add or subtract, n_a , is simply the integer portion of the quotient

$$n_a = \text{integer} (|L_a| / \Delta s). \quad (2.32)$$

The initial node spacing at the top of the system is thus preserved for all added nodes. Any fractional remainder from the quotient of equation 2.32 is carried over into the amount of cable added at the next time step.

2.4 Equations of motion

Detailed derivations of the continuous forms of the equations of motion used in *cable* are provided in [12]. They are presented below for reference. Common to all problem types is the notation for current, U_c, V_c, W_c (corresponding to global z, x, and y, directions) and u_c, v_c, w_c (corresponding to local tangential, normal, and bi-normal directions), material properties EA (axial stiffness), EI (bending stiffness), GJ (torsional stiffness), w_0 (wet weight per length), m (mass per unit length), and C_{dn}, C_{dt} (normal and tangential drag coefficients). $T(\epsilon)$ is the tension as a function of strain.

2.4.1 Two-dimensional problems

Two-dimensional problems represent a significant simplification over three-dimensional problems because we only need to make one rotation to transform between local and global coordinate systems. This rotation is represented in the equations of motion by the inclination angle, ϕ .

2.4.1.1 Static equations

In the two-dimensional static problem we can drop time dependent terms. The problem reduces to a system in four equations and four unknowns: ϵ (strain), S_n (shear force), ϕ (angle of inclination), and Ω_3 (curvature). We must explicitly include curvature,

$$\Omega_3 = \frac{\partial \phi}{\partial s}, \quad (2.33)$$

in order to maintain the system as first-order. Thus, the four equations for two-dimensional statics are

$$\frac{\partial \epsilon}{\partial s} - \frac{S_n \Omega_3}{T'(\epsilon)} - \frac{w_0}{T'(\epsilon)} \cos \phi + \frac{1}{2} \rho_w d \pi C_{dt} (\cos \phi U_c + \sin \phi V_c) \cdot |\cos \phi U_c + \sin \phi V_c| \frac{\sqrt{1 + \epsilon}}{T'(\epsilon)} = 0 \quad (2.34)$$

$$\frac{\partial S_n}{\partial s} + T(\epsilon) \Omega_3 + w_0 \sin \phi - \frac{1}{2} \rho_w d C_{dn} (\sin \phi U_c - \cos \phi V_c) |\sin \phi U_c - \cos \phi V_c| \sqrt{1 + \epsilon} = 0 \quad (2.35)$$

$$\frac{\partial \Omega_3}{\partial s} + \frac{1}{EI} S_n (1 + \epsilon)^3 = 0 \quad (2.36)$$

$$\frac{\partial \phi}{\partial s} - \Omega_3 = 0 \quad (2.37)$$

2.4.1.2 Dynamic equations

For the dynamic problem we must keep time dependent terms. The velocity variables, u (tangential velocity) and v (normal velocity), enter the problem and we have a system in six equations and six unknowns:

$$\begin{aligned} \frac{\partial \varepsilon}{\partial s} - \frac{m}{T'(\varepsilon)} \frac{\partial u}{\partial t} + \frac{m}{T'(\varepsilon)} v \frac{\partial \phi}{\partial t} - \frac{S_n \Omega_3}{T'(\varepsilon)} - \frac{w_0}{T'(\varepsilon)} \cos \phi \\ - \frac{1}{2} \rho_w d \pi C_{dt} (u - \cos \phi U_c - \sin \phi V_c) \cdot |u - \cos \phi U_c - \sin \phi V_c| \frac{\sqrt{1 + \varepsilon}}{T'(\varepsilon)} = 0 \end{aligned} \quad (2.38)$$

$$\begin{aligned} \frac{\partial S_n}{\partial s} - (m + m_a) \frac{\partial v}{\partial t} - \left[m u + \left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\sin \phi V_c + \cos \phi U_c) \right] \frac{\partial \phi}{\partial t} + T(\varepsilon) \Omega_3 \\ + w_0 \sin \phi - \frac{1}{2} \rho_w d C_{dn} (v + \sin \phi U_c - \cos \phi V_c) |v + \sin \phi U_c - \cos \phi V_c| \sqrt{1 + \varepsilon} = 0 \end{aligned} \quad (2.39)$$

$$\frac{\partial \Omega_3}{\partial s} + \frac{1}{EI} S_n (1 + \varepsilon)^3 = 0 \quad (2.40)$$

$$\frac{\partial u}{\partial s} - \frac{\partial \varepsilon}{\partial t} - \Omega_3 v = 0 \quad (2.41)$$

$$\frac{\partial v}{\partial s} - (1 + \varepsilon) \frac{\partial \phi}{\partial t} + \Omega_3 u = 0 \quad (2.42)$$

$$\frac{\partial \phi}{\partial s} - \Omega_3 = 0 \quad (2.43)$$

2.4.2 Three-dimensional problems

For three-dimensional problems, the transformation from local to global coordinate systems is written in terms of four Euler parameters (this contrasts to 3D formulations written in terms of three Euler angles, see [5, 12] for details).

2.4.2.1 Static equations

The three-dimensional static equations are written in terms of ten variables: ε , S_n , S_b (shear force in the bi-normal direction), the Euler parameters $(\beta_0, \beta_1, \beta_2, \beta_3)$, and curvatures Ω_1 (torsion), Ω_2 (curvature about the normal axis), and Ω_3 . The ten equations for three-dimensional statics are

$$\frac{\partial \varepsilon}{\partial s} + \frac{S_b \Omega_2}{T'(\varepsilon)} - \frac{S_n \Omega_3}{T'(\varepsilon)} - \frac{w_0}{T'(\varepsilon)} (\beta_0^2 + \beta_1^2 - \beta_2^2 - \beta_3^2) + \frac{1}{2} \rho_w d \pi C_{dt} u_c |u_c| \frac{\sqrt{1 + \varepsilon}}{T'(\varepsilon)} = 0 \quad (2.44)$$

$$\frac{\partial S_n}{\partial s} + T(\varepsilon)\Omega_3 - S_b\Omega_1 - 2w_0(\beta_1\beta_2 - \beta_0\beta_3) + \frac{1}{2}\rho_w dC_{dn}v_c\sqrt{v_c^2 + w_c^2}\sqrt{1 + \varepsilon} = 0 \quad (2.45)$$

$$\frac{\partial S_b}{\partial s} + S_n\Omega_1 - T(\varepsilon)\Omega_2 - 2w_0(\beta_1\beta_3 + \beta_0\beta_2) + \frac{1}{2}\rho_w dC_{dn}w_c\sqrt{v_c^2 + w_c^2}\sqrt{1 + \varepsilon} = 0 \quad (2.46)$$

$$\frac{\partial \Omega_1}{\partial s} = 0 \quad (2.47)$$

$$\frac{\partial \Omega_2}{\partial s} + \left(\frac{GI_p}{EI} - 1\right)\Omega_1\Omega_3 - \frac{1}{EI}S_b(1 + \varepsilon)^3 = 0 \quad (2.48)$$

$$\frac{\partial \Omega_3}{\partial s} + \left(1 - \frac{GI_p}{EI}\right)\Omega_1\Omega_2 + \frac{1}{EI}S_n(1 + \varepsilon)^3 = 0 \quad (2.49)$$

$$\frac{\partial \beta_0}{\partial s} + \frac{1}{2}(\beta_1\Omega_1 + \beta_2\Omega_2 + \beta_3\Omega_3) = 0 \quad (2.50)$$

$$\frac{\partial \beta_1}{\partial s} - \frac{1}{2}(\beta_0\Omega_1 - \beta_3\Omega_2 + \beta_2\Omega_3) = 0 \quad (2.51)$$

$$\frac{\partial \beta_2}{\partial s} - \frac{1}{2}(\beta_3\Omega_1 + \beta_0\Omega_2 - \beta_1\Omega_3) = 0 \quad (2.52)$$

$$\frac{\partial \beta_3}{\partial s} + \frac{1}{2}(\beta_2\Omega_1 - \beta_1\Omega_2 - \beta_0\Omega_3) = 0 \quad (2.53)$$

2.4.2.2 Dynamic equations

Three-dimensional dynamic problems restore the time-dependent terms and contain three velocity components, u , v , and w (w is the velocity in the bi-normal direction) giving 13 equations in all:

$$\begin{aligned} & \frac{\partial \varepsilon}{\partial s} - \frac{m}{T'(\varepsilon)} \frac{\partial u}{\partial t} + \frac{\Omega_2}{T'(\varepsilon)} (S_b - S_n) - \frac{w_0}{T'(\varepsilon)} (\beta_0^2 + \beta_1^2 - \beta_2^2 - \beta_3^2) \\ & - \frac{2m}{T'(\varepsilon)} \left[(v\beta_3 - w\beta_2) \frac{\partial \beta_0}{\partial t} - (w\beta_3 + v\beta_2) \frac{\partial \beta_1}{\partial t} + (w\beta_0 + v\beta_1) \frac{\partial \beta_2}{\partial t} + (w\beta_1 - v\beta_0) \frac{\partial \beta_3}{\partial t} \right] \\ & - \frac{1}{2}\rho_w d\pi C_{dt} (u - u_c) |u - u_c| \frac{\sqrt{1 + \varepsilon}}{T'(\varepsilon)} = 0 \end{aligned} \quad (2.54)$$

$$\frac{\partial S_n}{\partial s} - (m + m_a) \frac{\partial v}{\partial t}$$

$$\begin{aligned}
& -2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_3 U_c - \beta_0 V_c - \beta_1 W_c) + m (w\beta_1 - u\beta_3) \right] \frac{\partial \beta_0}{\partial t} \\
& + 2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_2 U_c - \beta_1 V_c + \beta_0 W_c) - m (u\beta_2 - w\beta_0) \right] \frac{\partial \beta_1}{\partial t} \\
& + 2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_1 U_c + \beta_2 V_c + \beta_3 W_c) + m (u\beta_1 + w\beta_3) \right] \frac{\partial \beta_2}{\partial t} \\
& - 2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_0 U_c + \beta_3 V_c - \beta_2 W_c) + m (u\beta_0 + w\beta_2) \right] \frac{\partial \beta_3}{\partial t} \\
& + T(\varepsilon)\Omega_3 - S_b\Omega_1 - 2w_0 (\beta_1\beta_2 - \beta_0\beta_3) \\
& - \frac{1}{2}\rho_w d C_{dn} (v - v_c) \sqrt{(v - v_c)^2 + (w - w_c)^2} \sqrt{1 + \varepsilon} = 0
\end{aligned} \tag{2.55}$$

$$\begin{aligned}
& \frac{\partial S_b}{\partial s} - (m + m_a) \frac{\partial w}{\partial t} \\
& + 2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_2 U_c - \beta_1 V_c + \beta_0 W_c) + m (v\beta_1 - u\beta_2) \right] \frac{\partial \beta_0}{\partial t} \\
& + 2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_3 U_c - \beta_0 V_c - \beta_1 W_c) - m (v\beta_0 + u\beta_3) \right] \frac{\partial \beta_1}{\partial t} \\
& + 2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_0 U_c + \beta_3 V_c - \beta_2 W_c) + m (u\beta_0 - v\beta_3) \right] \frac{\partial \beta_2}{\partial t} \\
& + 2 \left[\left(\rho_w \frac{\pi d^2}{4} + m_a \right) (\beta_1 U_c + \beta_2 V_c + \beta_3 W_c) + m (v\beta_2 + u\beta_1) \right] \frac{\partial \beta_3}{\partial t} \\
& + S_n\Omega_1 - T(\varepsilon)\Omega_2 - 2w_0 (\beta_1\beta_3 + \beta_0\beta_2) \\
& - \frac{1}{2}\rho_w d C_{dn} (w - w_c) \sqrt{(v - v_c)^2 + (w - w_c)^2} \sqrt{1 + \varepsilon} = 0
\end{aligned} \tag{2.56}$$

$$\frac{\partial \Omega_1}{\partial s} = 0 \tag{2.57}$$

$$\frac{\partial \Omega_2}{\partial s} + \left(\frac{GI_p}{EI} - 1 \right) \Omega_1 \Omega_3 - \frac{1}{EI} S_b (1 + \varepsilon)^3 = 0 \tag{2.58}$$

$$\frac{\partial \Omega_3}{\partial s} + \left(1 - \frac{GI_p}{EI} \right) \Omega_1 \Omega_2 + \frac{1}{EI} S_n (1 + \varepsilon)^3 = 0 \tag{2.59}$$

$$\frac{\partial u}{\partial s} - \frac{\partial \varepsilon}{\partial t} + \Omega_2 w - \Omega_3 v = 0 \tag{2.60}$$

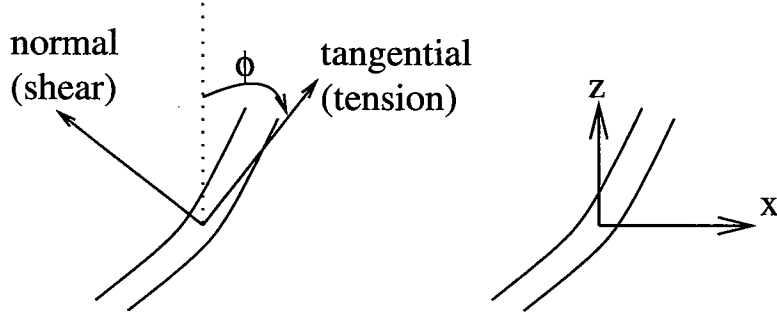


Figure 2.1: Local and global two-dimensional coordinate systems.

$$\frac{\partial v}{\partial s} + 2(1 + \varepsilon) \left(\beta_3 \frac{\partial \beta_0}{\partial t} - \beta_2 \frac{\partial \beta_1}{\partial t} + \beta_1 \frac{\partial \beta_2}{\partial t} - \beta_0 \frac{\partial \beta_3}{\partial t} \right) + \Omega_3 u - \Omega_1 w = 0 \quad (2.61)$$

$$\frac{\partial w}{\partial s} - 2(1 + \varepsilon) \left(\beta_2 \frac{\partial \beta_0}{\partial t} + \beta_3 \frac{\partial \beta_1}{\partial t} - \beta_0 \frac{\partial \beta_2}{\partial t} - \beta_1 \frac{\partial \beta_3}{\partial t} \right) + \Omega_1 v - \Omega_2 u = 0 \quad (2.62)$$

$$\frac{\partial \beta_0}{\partial s} + \frac{1}{2} (\beta_1 \Omega_1 + \beta_2 \Omega_2 + \beta_3 \Omega_3) = 0 \quad (2.63)$$

$$\frac{\partial \beta_1}{\partial s} - \frac{1}{2} (\beta_0 \Omega_1 - \beta_3 \Omega_2 + \beta_2 \Omega_3) = 0 \quad (2.64)$$

$$\frac{\partial \beta_2}{\partial s} - \frac{1}{2} (\beta_3 \Omega_1 + \beta_0 \Omega_2 - \beta_1 \Omega_3) = 0 \quad (2.65)$$

$$\frac{\partial \beta_3}{\partial s} + \frac{1}{2} (\beta_2 \Omega_1 - \beta_1 \Omega_2 - \beta_0 \Omega_3) = 0 \quad (2.66)$$

2.5 Coordinate transformations

2.5.1 Two-dimensional

The relationship between local and global coordinates in two dimensions is shown in figure 2.1. The transformation of a vector (f_t, f_n) from local coordinates to a vector (F_z, F_x) in global coordinates is expressed in terms of a transformation matrix that is only a function of ϕ , the local inclination angle

$$\begin{bmatrix} F_z \\ F_x \end{bmatrix} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} f_t \\ f_n \end{bmatrix}. \quad (2.67)$$

Using the principle that the inverse of a transformation matrix is the transpose of the transformation, the reverse transform, from global to local coordinates, is

$$\begin{bmatrix} f_t \\ f_n \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} F_z \\ F_x \end{bmatrix}. \quad (2.68)$$

2.5.2 Three-dimensional

In three dimensions the transformation matrix that takes a vector in local coordinates (f_t, f_n, f_b) to global coordinates (F_z, F_x, F_y) is expressed in terms of the four Euler parameters, $\beta_0, \beta_1, \beta_2, \beta_3$,

$$\begin{bmatrix} F_z \\ F_x \\ F_y \end{bmatrix} = \begin{bmatrix} \beta_0^2 + \beta_1^2 - \beta_2^2 - \beta_3^2 & 2(\beta_1\beta_2 - \beta_0\beta_3) & 2(\beta_1\beta_3 + \beta_0\beta_2) \\ 2(\beta_1\beta_2 + \beta_0\beta_3) & \beta_0^2 - \beta_1^2 + \beta_2^2 - \beta_3^2 & 2(\beta_2\beta_3 - \beta_0\beta_1) \\ 2(\beta_1\beta_3 - \beta_0\beta_2) & 2(\beta_2\beta_3 + \beta_0\beta_1) & \beta_0^2 - \beta_1^2 - \beta_2^2 + \beta_3^2 \end{bmatrix} \begin{bmatrix} f_t \\ f_n \\ f_b \end{bmatrix}. \quad (2.69)$$

The reverse transformation is

$$\begin{bmatrix} f_t \\ f_n \\ f_b \end{bmatrix} = \begin{bmatrix} \beta_0^2 + \beta_1^2 - \beta_2^2 - \beta_3^2 & 2(\beta_1\beta_2 + \beta_0\beta_3) & 2(\beta_1\beta_3 - \beta_0\beta_2) \\ 2(\beta_1\beta_2 - \beta_0\beta_3) & \beta_0^2 - \beta_1^2 + \beta_2^2 - \beta_3^2 & 2(\beta_2\beta_3 + \beta_0\beta_1) \\ 2(\beta_1\beta_3 + \beta_0\beta_2) & 2(\beta_2\beta_3 - \beta_0\beta_1) & \beta_0^2 - \beta_1^2 - \beta_2^2 + \beta_3^2 \end{bmatrix} \begin{bmatrix} F_z \\ F_x \\ F_y \end{bmatrix}. \quad (2.70)$$

Chapter 3

Structure of a *cable* Problem

3.1 Notation and coordinate systems

The basic coordinate system for *cable* is shown in figure 3.1. Note that the origin of the coordinate system is always located at the anchor and that the global z direction is positive upwards, global x is positive to the right, and global y is positive into the page¹. Current can be defined as a function of depth and can flow in both the x and y directions. Currents with vertical components (along the z axis) are not allowed. Depending on the problem type under consideration the depth may or may not be required in the problem definition.

3.2 Basic language features

The input language for *cable* is meant to be as flexible and as forgiving as possible in terms of the detailed structure of an input file. The file is broken into sections, with each section containing definition statements for a particular aspect of the problem. In general, sections can be specified in any order, as can definitions within a section. Multiple sections of the same type can be included in a single input file.

White space (blank lines, spaces, tabs) does not affect the interpretation of the problem and can be used arbitrarily to suit individual tastes. Comments are denoted as in the C programming language; anything between `/*` and `*/` will be ignored as a comment no matter where it appears in the file.

¹Note that *cable* uses a rotated internal coordinate system for calculations and results storage in which x is up, y is right, and z is into the page. Both user and internal coordinate systems have their origin at the anchor.

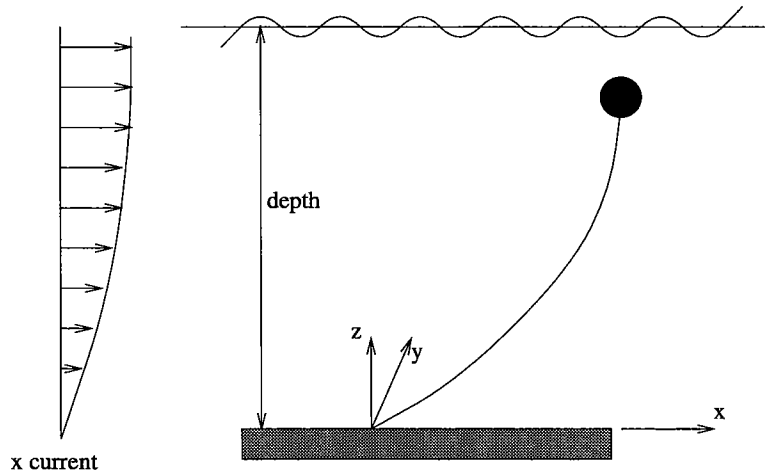


Figure 3.1: Geometric definitions for *cable*.

Object names (i.e., the names you assign to specific buoy or material definitions) cannot be keywords. They must begin with an alphabetic character and should contain only alphabetic characters, numbers, and underscores. If a name has spaces in it then it must be contained in double quotes. The case of keywords (either upper, lower, or mixed) does not matter. The capitalization of object names, however, is relevant.

3.2.1 Expressions

3.2.1.1 Continuous functions

As a convenience, wherever a floating point numeric value is required for a parameter you can specify an arbitrary mathematical expression, including the operators $+$, $-$, $*$, $/$, $\%$ (modulo) and the standard mathematical library functions *sin*, *cos*, *tan*, *sqrt*, *hypot*, *pow*, *exp*, *log*, *log10*, *floor*, *ceil*, *fabs* and *fmod*. Note that arguments to the trigonometric functions should be given in terms of radians just as if you were calling them from a C program using the standard math library. Other than this difference, these functions should be used and should behave as they are described in the manual pages for your local mathematics library. Expressions can also contain the ternary conditional operator as in the C programming language: “if a then b else c” is symbolized in a *cable* input file as $a ? b : c$ where a, b, and c are all valid expressions. The logical operators to use in constructing a are the same as those in C ($==$, $\&\&$ (and), $||$ (or), $<=$, $<$, $>$, $>=$, $!=$ (not equal)). Current specifications are a special case of the use of expressions; the current can be defined as a function of depth using expressions containing the variable H. These expressions will be dynamically evaluated throughout the course of the simulation. Expressions for tow speed

and apy-out rate may contain the variable `t` to denote them as a function of time.

3.2.1.2 Discrete functions

Because some current functions are easier to express in a discretized (as opposed to continuous) form, the *cable* syntax also includes a mechanism for specifying a discrete representation of a current. The basic specification consists of a series of pairs of the form (H, U) where U is the value of the function at depth H. In evaluating the function, *cable* will linearly interpolate between adjacent pairs for positions that fall between two pairs. The following illustrates this idea for the case of a current defined piecewise linear

```
x-current = (0, 0.4) (100, 0.4) (500, 0.2) (1000, 0.0)
```

From the surface (depth = 0.0) to a depth of 100, the current is constant at 0.4. Over the interval from 100 to 500, the current decreases linearly from 0.4 to 0.2. From a depth of 500 to 1000 the current decreases linearly from 0.2 to 0.0. Points below 1000 would be extrapolated based on the last two pairs (in our example, extrapolation would result in negative values for current at depths greater than 1000). Note that the pairs must be given in order of increasing depth coordinate. You can express a periodic discrete function simply by defining one period and then entering a + symbol at the end of the expression.

3.2.2 Units

There are no set units for the dimensional quantities that you specify in defining a problem for *cable*. The important thing is to remain consistent in the units that you use; numerical results will then be consistent with the input dimensions. Some examples of consistent units would be lengths in meters, weights in Newtons, elastic moduli in Pascals (N m^{-2}); moments of inertia would be in m^{-4} . Convenient English units are often pounds, feet, and psf (pounds per square foot) or kips (kilopounds), feet, and ksf.

3.3 Components of an input file

A *cable* input file consists of a series of definition statements contained within eight distinct sections. There are three sections which define the basic numerical and environment set-up for the model (**problem description**, **environment**, **analysis parameters**), four sections for defining the system components (**materials**, **connectors**, **buoys**, **anchors**) and a single section to define how the system gets put together (**layout**). The appendices to this manual provide several complete example input files for a variety of problem types. Detailed

definitions for all elements of the input syntax are provided below. Section 3.4 offers some suggestions about how the various input elements can be manipulated to get fast, robust, and accurate solutions for models that may initially prove difficult to converge.

3.3.1 Problem description

The `problem description` section contains the most basic description of the system: a descriptive title string and the definition of the problem type. It must be the first section within the input file and cannot be repeated.

`title = string`

A character string containing the problem title to be used in the display of results.

`type = problem type`

`problem type` can currently be one of `general`, `surface`, `subsurface`, `horizontal`, `towing`, `drifter`. For `general` problems, the second terminal end in the `layout` section must have a static force `x-force=`, `y-force=`, etc. defined. In `surface` problems, the second terminal end must have a completely defined buoy attached. `cable` will perform outer loop iterations to solve for the static draft of the buoy. In `subsurface` problems the second terminal end must also be a completely defined buoy but `cable` can calculate the forces on that buoy without outer loop iterations. In `horizontal` problems, the second terminal end must be an anchor and must have a position associated with it `x=`, `y=`, `z=`. `cable` will perform outer loop iterations to calculate the appropriate reaction force at the second anchor that brings the anchor to the required position. `towing` problems must have buoys defined at both terminals. The buoy description at the first terminal defines the tow-body and must be complete. The buoy at the second terminal does not need to be completely defined. Tow speeds (`x-speed=`, etc) must be specified for the second terminal of a `towing` problem. `drifter` problems are terminated with buoys at both ends; both buoy definitions must be complete.

3.3.2 Analysis parameters

The `analysis parameters` section contains definitions that control the numerical algorithms which are used in the static and dynamic solutions of `cable` problems.

duration = constant expression

The total length of the dynamic simulation. Must be given if a dynamic solution is going to be performed.

time-step = constant expression

The time step of the dynamic simulation. Decreasing the time step is sometimes a good way to get around a singularity that may be occurring. Must be given if a dynamic solution is going to be computed. *cable* does have an adaptive time stepping algorithm that allows it to dynamically decrease the time step if it encounters a singularity or a time step which exceeds the dynamic iteration limit. The time step will be reduced by successive factors of ten up to a maximum of 5 times. If after the fifth nested reduction a singularity is encountered the program will halt.

tolerance = constant expression

The global convergence tolerance of the relaxation iterations to be used if specific tolerances are not given for the separate phases of the problem. The tolerance dictates the minimum acceptable relative error between iterations for a solution phase to be considered converged.

dynamic-tolerance = constant expression

The convergence tolerance for dynamic iterations. In some problems, it may be desirable to have different tolerances for static and dynamic solutions. If not given it will default to the value given by **tolerance=**. At least one or the other must be given.

static-tolerance = constant expression

The convergence tolerance for static iterations. In some problems, it may be desirable to have different tolerances for static and dynamic solutions. If not given then it will default to the value given by **tolerance=**. At least one or the other must be given.

static-outer-tolerance = constant expression

The convergence tolerance for the outer loop of static iterations. This will control the relative error in the iterations used to determine surface draft or placement of the second anchor for example. If not given then it will default to the value given by **tolerance=** then to **static-tolerance**. At least one of these must be given.

relaxation = constant expression

The global relaxation factor to use in the iterative update scheme. Typi-

cally it can be most safely and reliably set to 1.0, but there are exceptions, particularly in static solutions of systems with complex geometry or cable lying on the bottom, where it may need to be quite small (0.01 – 0.1) to get a solution to converge.

dynamic-relaxation = constant expression

The relaxation factor to be used in the dynamic solution. In some problems, it may be necessary to have different factors for static and dynamic solutions (see **relaxation=** above). If not given then it will default to the value given by **relaxation=**. At least one or the other must be given.

static-relaxation = constant expression

The relaxation factor to be used in the static solution. In some problems, it may be necessary to have different factors for static and dynamic solutions (see **relaxation=** above). If not given then it will default to the value given by **relaxation=**. At least one or the other must be given.

static-outer-relaxation = constant expression

The “relaxation” or “stiffness” factor to be used in static outer iterations. This is the factor β in equations 2.11 and 2.13. For **surface** problems it defaults to 0.95. For **horizontal** problems it defaults to 5.0.

max-iterations = integer

The global maximum number of iterations in all convergence loops. It provides a single default for the other iteration controls.

static-iterations = integer

The maximum permitted number of relaxation iterations in the static solution. This number may need to be quite high for problems with small **static-relaxation** factors. It will default to **max-iterations** if not given explicitly. At least one or the other must be given.

dynamic-iterations = integer

The maximum permitted number of relaxation iterations at each time step. Generally, static solutions can take more iterations than the dynamic solution at a single time step so this number can be set lower. It will default to **max-iterations** if not given explicitly. At least one or the other must be given.

static-outer-iterations = integer

The maximum permitted number of iterations to take in resolving the anchor or buoy position in the static solution. Because the algorithms for finding

the position of the second anchor in a horizontal mooring problem or the surface buoy in a surface mooring problem are quite conservative, they can often take many hundreds of iterations to converge. This parameter gives you the capability to allow for large numbers of iterations in these outer convergence loops, but not in the general relaxation iteration. It will default to `max-iterations` or `static-iterations` in that order of preference.

`ramp-time = constant expression`

The time period over which the excitation amplitudes will be linearly ramped up to their full values. A non-zero ramp time is often used to minimize numerical transients. If not specified or if given as 0.0 then the excitation amplitudes will simply be at their full value right from the start of the simulation.

`current-steps = integer`

The number of steps to take in bringing the current up to its full value in the static solution. For some problems with high currents it can help convergence if the current is brought up to speed slowly.

3.3.3 Environmental parameters

The `environment` section is used to define the external parameters in which the simulation is run. Density, gravity, depth, waves, current, and bottom parameters are all defined here

`gravity = constant expression`

The acceleration of gravity expressed in appropriate units. Must always be specified.

`rho = constant expression`

The density of the fluid medium. Must always be specified.

`depth = constant expression`

The depth of the water. Required for all problems except `towing` and `drifter`.

`input-type = string`

Specifies the nature of the dynamic inputs, either `regular` (harmonic) or `random`. `regular` type inputs treat the excitation as harmonic functions with the given amplitude, period and phase. `random` inputs build a random profile using the given amplitude as the significant amplitude and the given period as the peak period. Input phase information is ignored in `random`

type inputs because the phase of each component is assigned randomly.

forcing-method = string

This must be specified for dynamic problems. The only currently acceptable values are: **wave-follower**, **morison**, **froude-krylov**, and **velocity**. **morison** forcing is really only appropriate for subsurface moorings where free surface effects are negligible. **froude-krylov** forcing is appropriate for surface buoys which are not wave followers. Note that buoys must be defined as **axisymmetric** type with a diameter of 0.0 at the lower end of the buoy if **froude-krylov** forcing is used. **velocity** forcing simply imposes a specified motion on the topmost node of the system.

x-wave = (constant expression, constant expression, constant expression)

The amplitude, period, and relative phase of the surface wave travelling in the global x direction.

y-wave = (constant expression, constant expression, constant expression)

The amplitude, period, and relative phase of the surface wave travelling in the global y direction.

x-input = (constant expression, constant expression, constant expression)

The amplitude, period, and relative phase of the dynamic input in the global x direction. Only useful when **forcing-method** is **velocity**.

y-input = (constant expression, constant expression, constant expression)

The amplitude, period, and relative phase of the dynamic input in the global y direction. Only useful when **forcing-method** is **velocity**.

z-input = (constant expression, constant expression, constant expression)

The amplitude, period, and relative phase of the dynamic input in the global z direction. Only useful when **forcing-method** is **velocity**.

x-current = variable expression

The current in the global z-direction, possibly as a function of depth (using either a discrete expression or a continuous expression with the symbolic variable H).

y-current = variable expression

The current in the global y-direction, possibly as a function of depth (using either a discrete expression or a continuous expression with the symbolic variable H).

bottom-stiffness = constant expression

The spring stiffness of the bottom per unit length.

bottom-damping = constant expression

The damping ratio of the bottom. The dashpot coefficient for the bottom is calculated from this ratio using a natural frequency based on bottom stiffness and cable mass. Setting this parameter too high can sometimes lead to instabilities in the dynamic solution of a problem.

3.3.4 Cable, chain and rope materials

The `materials` section defines the cable materials that make-up the system. Each material definition consists of a unique name followed by a series of material property definitions, such as

wire	EA = 4.4e6	EI = 500	GJ = 25
	m = 0.160	am = 0.05	wet = 1.15
	d = 0.0063	Cdt = 0.01	Cdn = 1.5

Remember that white space and ordering does not matter so these properties could be arranged in many other ways.

EA = constant expression

The axial stiffness of the material. Must be non-zero. For ropes and cables this value should typically be slightly less than the straight product of E (elastic modulus) times A (cross-sectional area). It is best determined from the slope of an experimentally derived load-elongation curve.

EI = constant expression

The bending stiffness of the material. Must be non-zero. For chains it should be very small. For ropes and cables the value should probably be something less than the theoretical value for a solid rod given by

$$E \frac{\pi R^4}{4}. \quad (3.1)$$

GJ = constant expression

The torsional stiffness of the material. Must be non-zero, but is ignored in 2D problems and could thus be arbitrary in those cases. For ropes and cables the value should probably be something less than the theoretical value for a

solid rod given by

$$G \frac{\pi R^4}{2}. \quad (3.2)$$

m = constant expression

The mass per unit length of the material. Must be non-zero.

am = constant expression

The transverse added mass per unit length of the material. If the added mass for a material is not specified (or is specified to be zero) then the added mass will be calculated as $am = A\rho$ where A is the cross-sectional area of the material based on the specified material diameter and ρ is the density of the fluid medium defined in the **environment** section.

wet = constant expression

The wet weight (weight in the fluid characterized by the density defined in the **environment** section) per unit length. If the wet weight for a material is not specified (or is specified to be zero) then it will be calculated as $wet = mg - A\rho g$ where A is the cross-sectional area of the material based on the specified material diameter, m is the specified material mass per unit length, and g and ρ are the gravitational acceleration and density of the fluid medium defined in the **environment** section.

d = constant expression

The diameter of the material. This value is used in drag calculations for projected area and to calculate wet weight and added mass when these values are not explicitly given. For chains, this diameter is typically taken as the outside width of a single link.

Cdn = constant expression

The drag coefficient in the normal (transverse) direction. Typically between 1.5 and 2.0 for standard circular oceanographic cables.

Cdt = constant expression

The drag coefficient in the tangential (longitudinal) direction. Typical values for oceanographic cables range between 0.01 and 0.1.

3.3.5 Connectors

The **connectors** section is used to define the shackles, float, and instruments that are placed between cable segments. A connector is defined by a unique name followed by a series of

property definitions.

wet = constant expression

The wet weight of the connector.

m = constant expression

The mass of the connector.

Cdn = constant expression

The drag coefficient of the connector. The drag is currently assumed to be equal in both the normal and tangential directions.

d = constant expression

The characteristic diameter used to calculate a drag area.

All connectors are currently moment releasing – that is they cannot transmit a moment between the segments which they are placed between. For shackle and pin-type connectors this is a reasonable assumption.

3.3.6 Buoys

The buoys section defines the buoy or ship that is used at the top of the mooring. If part of the solution involves calculating the static forces at the top of the mooring due to buoyancy and drag, or if morison or froude-krylov forcing was specified in the `environment` section then buoy definitions must be complete. Buoys are also used to represent the towed vehicle end of a towing problem or the sinker weight in a drifter problem. In this case they should be defined as an equivalent sphere with the diameter and an explicitly specified buoyancy (rather than an automatically calculated buoyancy based on the diameter of the sphere) manipulated to simulate the proper drag area and wet weight (see example, Appendix E).

type = string

The basic buoy type. Currently recognized values are `sphere`, `cylinder`, `capsule`, `axisymmetric`.

d = constant expression

The diameter of the buoy for buoy shapes with pre-defined geometry (`cylinder`, `sphere`, `capsule`).

h = constant expression

The total height of a `cylinder` buoy or the total length of a `capsule` buoy.

diameters = (x₁, d₁) ... (x_n, d_n)

The description of the geometry of an axisymmetric buoy from the bottom up. Each pair of numbers represents a level and a diameter. The buoy geometry is defined as an axisymmetric body of revolution formed by the lines connecting these points.

buoyancy = constant expression

If given this will be used as the total available buoyancy of the buoy. If not given it will be computed based on buoy type and specified geometry.

m = constant expression

The mass of the buoy.

Cdn = constant expression

The drag coefficient of the buoy.

3.3.7 Anchors

The anchors section defines the anchors that are used at one or both ends of the system. None of the parameters are currently used in any of the analyses but you do need to at least create a valid anchor name to be used in the terminal definitions of the layout section.

3.3.8 System layout

The geometry of the model system is built from the bottom up as a series of segments with optional connectors between segments and terminal points at the ends. Terminal points can consist either of buoys or anchors. The layout section for a single point mooring with just one shot of material looks like the following

Layout

```
terminal = { anchor = clump }
segment = {
    length    = 200
    material  = wire
    nodes     = (100, 1.0)
}
terminal = { buoy = snubber }
```

If there was a shot of nylon above the wire connected by a shackle then we simply add a connector = statement and a second segment = statement

Layout

```
terminal = {
  anchor = clump
}
segment = {
  length    = 200
  material   = wire
  nodes     = (100, 1.0)
}
connector = shackle
segment = {
  length    = 50
  material   = nylon
  nodes     = (100, 1.0)
}
terminal = {
  buoy = snubber
}
```

In both of these examples all of the named objects `clump`, `wire`, `shackle`, `snubber` need to be defined somewhere in the input file.

If we wanted to define a problem with both ends anchored to the bottom then we simply specify a different terminal at the second end of the system

Layout

```
terminal = {
  anchor = clump
}
segment = {
  length    = 20
  material   = wire
  nodes     = (40, 1.0)
}
connector = glass_sphere
segment = {
  length    = 100
  material   = nylon
  nodes     = (50, 1.0)
}
```

```

connector = glass_sphere
segment = {
    length      = 20
    material    = wire
    nodes      = (40, 1.0)
}
terminal = {
    anchor = clump
    z = 0.0
    x = 100.0
}

```

This would define a three segment system with both ends anchored; the second anchor is located 100 units to the right of the first anchor.

```
terminal = { terminal definition }
```

A terminal definition must come both at the beginning and end of the list of segments and connectors and it cannot occur anywhere else. It can consist of statements of the form `anchor=`, `buoy=`, `x=`, `y=`, `z=`, `x-force=`, `y-force=`, `z-force=`, `x-speed=`, `y-speed=`, `z-speed=`, and `pay-rate=`. It must contain at least an anchor or buoy definition.

`z = constant expression`

The z location of an anchor in the global coordinate space.

`x = constant expression`

The x location of an anchor in the global coordinate space.

`y = constant expression`

The y location of an anchor in the global coordinate space.

`z-force = constant expression`

Optionally user provided static force on a buoy. If specified it is important that it be large enough to support the weight of the mooring. If it is not large enough the solution will either be upside down or the problem will not be solvable. User specified static forces are only used for **general** problems. In all other cases, *cable* automatically calculates end-point static forcing based on currents and drag properties and weights and buoyancies.

`x-force = constant expression`

Optionally user provided static force on a buoy.

y-force = constant expression

Optionally user provided static force on a buoy.

x-speed = variable expression

Optionally user specified speed of the terminal in the global x direction. The expression can be a function of t (time) or a discrete step-wise expression. This is most commonly used to specify a tow speed or the speed of a surface drifter. As functions of time, the **x-speed** and **y-speed** can be used to specify complex motions of a surface tow ship (for example, two sinusoidal functions out of phase with one another could be used to specify circular or elliptical tow patterns). In order to get a valid static (steady-state) solution, some components of the speed should evaluate to non-zero values at time $t = 0.0$.

y-speed = variable expression

Optionally user specified speed of the terminal in the global y direction.

z-speed = variable expression

Optionally user specified speed of the terminal in the global z direction.

pay-rate = variable expression

The pay-out (or pay-in) rate of material off of the terminal. This is typically most useful for towing problems. The expression can be a function of time or a discrete expression. Positive rates indicate material being added to the system; negative rates indicate material being taken out of the system. Rates should be specified in units of length per time. See section 2.3.4 for additional details. Remember that for problems with positive pay-rates the total number of nodes in the problem will be greater than the total number defined by the sum of all nodes over all segments defined in the layout section.

anchor = string

Specifies the anchor type to use at this termination.

buoy = string

For traditional single point moorings this defines the name of the buoy to be used at the top terminal of the system. For drifter and towing problems, a **buoy = statement** is used in the first **terminal** definition to define the mass at the subsurface free end of the system (usually a depressor weight or a vehicle).

release-time = constant expression

The time point during the simulation at which the the buoy or anchor should be released from the system. This can be used to simulate anchor release for mooring retrieval problems and cable breaking for towing problems.

segment = { segment definition }

A segment definition consists of three required statements: **length =** , **material =** , and **nodes =** . The statement **attachments =** is optional.

length = constant expression

The length of the segment.

material = string

The material type to be used for this segment.

nodes = (integer, constant expression) (integer, constant expression) ...

The number and distribution of nodes to be used in discretizing the segment. In general a discretization will consist of a series of pairs of the form (number of nodes, fraction of length) where the total number of nodes for the segment is derived from the number of nodes listed in each pair and the length fractions of all pairs must add to 1.0. The construct allows for increasing node density over portions of a segment where high spatial gradients are expected (oftentimes the endpoints of a segment). For instance a specification of the form **nodes = (100, 0.1) (100, 0.8) (100, 0.1)** will place 100 nodes in both the first and last 10% of the segment and 100 nodes in the middle 80% of the segment.

attachments = string : (n₁¹, n₂¹, ...), string : (n₁², n₂², ...), ...

Specifies an optional list of attached objects on this segment. Attachments add mass, weight, and drag at a node. Each attachment consists of an object defined in the **connectors** section and a list of local node numbers (i.e., node numbers referenced to the segment that is being defined) at which that type of object should be placed. Multiple types of attachments can be defined as shown. Any given node can only have one type of object attached, however.

connector = string

Specifies the optional connector that can be placed between segments. If no connector is specified between segments then the joined ends of the two segments simply overlap and the results for the two nodes located at that point will always be identical. Omitting a connector between segments is one way to model a connection that does not transmit moments.

3.3.9 The end statement

The final statement in any input file must be an **end** statement.

3.4 Tips and tricks

The following sections separate the common causes of convergence failures, singularities, and instabilities into problems arising in the static and dynamic solutions. For each solution type, the common problems are bulleted and a brief description of possible ways to get around that problem is given.

3.4.1 Static problems

- Bad initial guess

For any problem in which the initial guess based on the catenary solution for a homogeneous, inextensible material with drag forcing only at the two ends is a very poor guess, consider using a small **static-relaxation** (maybe 0.1 – 0.2).

- Strong currents cause solution instabilities

If the current is sufficiently large to render the initial catenary guess a very poor solution, or if there is current in both horizontal directions and the static solution will have a complex non-planar shape, consider using **current-steps** to ramp the current slowly to its full value. 5 or 10 steps usually does the trick. At each step the initial guess is the solution from the previous step; a greater number of steps means a smoother transition from the catenary solution with no current to the actual solution with the full current.

- Cable on the bottom causes singularity or convergence failure

Problems with cable lying on the bottom will almost certainly require that **static-relaxation** be set to something on the order of 0.1 or possibly even smaller. Remember to allow for lots of **static-iterations** with small static relaxation factors.

- Surface buoy draft is difficult to converge

If the outer-iteration loop to find the draft of a surface buoy seems to be oscillating but never converging or if it goes down to very small guessed drafts which cause instabilities or singularities, use a larger **static-outer-relaxation** (but keep it less than 1.0, consider going to something like 0.98 or 0.99). You may need to raise the **static-outer-iterations** limit as well.

- Outer-iterations convergence is slow

You can usually speed up outer-iterations by raising the static-outer-tolerance to something on the order of 0.01. This tolerance translates directly to a percentage error in the guessed draft or position of the second anchor and so usually means an answer that is good to a couple of centimeters. A second option is to raise `static-tolerance` to speed the inner-iterations at every outer-iteration.

3.4.2 Dynamic problems

- Time step is always adapting

If *cable* is constantly adapting the time step downward (but always makes progress at the smaller time step) then it is best to simply set the base step (`time-step=`) to something smaller.

- The adaptation limit is exceeded

Exceeding the adaptation limit can be a sign of an unstable problem. Sometimes you will find that setting a base `time-step` that is 10% of the original base results in no adaptive reductions and reliable results. In other cases try lowering `dynamic-relaxation` to give *cable* more ability to work through a problem spot at the larger time steps. Going too low can dramatically slow the solution down, however; 0.5 is a reasonable lower limit. If *cable* is adapting because it is hitting the iteration limit, raise `dynamic-iterations` (particularly if you lowered `dynamic-relaxation`).

- There is a DC drift in solution variables

If the time histories of the result variables seem to have a large DC drift component, consider adding nodes to the problem (throughout the system, not just at spots of high gradient) and using `ramp-time` to slowly bring the excitation level to its full value.

- Cable impacting the bottom causes singularities

A damping ratio that is too high can cause dramatic convergence problems. If the system has cable which is being lifted and lowered from the bottom and the problem is not converging well, use a smaller `bottom-damping` value.

- 2D solution is fine, but 3D solution is difficult

If a problem solves with the 2D algorithm, but runs into singularities or exceeds the adaptation limit with the 3D algorithm, use a smaller `dynamic-relaxation` and a smaller base value for `time-step` when using the 3D algorithm.

Chapter 4

The *cable* Application

4.1 Basic operation

The basic way to solve a static problem with *cable* is simply to type

```
% cable -in foo.in -out foo.res -static
```

on the command line, where `foo.in` is the name of a *cable* input file and the output file will be named `foo.res`¹. For a dynamic problem, a typical command line might look like

```
% cable -in foo.in -out foo.res -nodes 50 100 -sample 0.1 -snap_dt 1.0
```

Like the static problem, the input and output files are required parameters on the command line. The contents of the results file are determined by the remaining parameters; it will contain information at nodes 50 and 100 at every 0.1 seconds and information at all nodes (a “snapshot”) at every 1.0 seconds. Exactly what information gets saved at those time points (and the information written for a static result) is controlled by additional parameters. By default, as many variables as are applicable will be output for a given problem – you can change this behavior by turning unnecessary variables off. In static solutions, available information includes motion (position), forces, moments and Euler parameters; in dynamic solutions, velocity is added to the list of available information.

For example, a static problem solved with the command

```
% cable -in foo.in -out foo.res -static +motion +moment +euler
```

¹There are no enforced naming conventions for input or output files (i.e., there is no requirement that input files have the extension `.in` or that output files have the extension `.res`).

will contain only force (tension and shear forces) information because all other applicable variables have been turned off (with the `+motion`, `+moment`, and `+euler` switches). If we remove the static solution switch and add sampling information

```
% cable -in foo.in -out foo.res -snap_dt 1.0 +motion +moment +euler
```

then `cable` will follow the static solution with a dynamic solution and the results file will contain both force and velocity information, but only in snapshot form at 1.0 second intervals. If we also wanted a detailed time history of the position of node 100 then the above command line would become

```
% cable -in foo.in -out foo.res -snap_dt 1.0 -sample 0.1 -nodes 100
+moment +euler
```

4.2 Using the run-time solution controls

By default, `cable` provides run-time feedback in the form of ASCII text output to the terminal. This information consists of the current iteration number, time step, error tolerance and any diagnostic messages. This information can be logged by redirecting the stdout output stream to a file.

An alternative to this form of feedback is the graphical information and control dialog pictured in figure 4.1. This control can be enabled by specifying `-X` on the `cable` command-line. If this dialog is enabled then the textual output to the stdout stream of the terminal will be suppressed and all diagnostic information is sent to the appropriate fields of the control dialog. Note that with the information and control dialog enabled, `cable` does not automatically exit after the solution is complete. The status message will change to indicate a complete problem, and the dialog will remain on the screen until the **quit** button is pressed.

The information and control dialog also allows certain aspects of the analysis parameters to be adjusted during the solution of the problem. The relaxation factor, tolerance and iteration limit for all three iteration loops (static, static outer, and dynamic) can be adjusted. A typical need for such an adjustment might be to allow for more iterations if a problem is observed to be converging but not fast enough that it will reach the desired tolerance by the iteration limit pre-set within the input file. Relaxation factors can also sometimes be adjusted advantageously to speed-up convergence or to stabilize an iteration.

To adjust a parameter, the problem must be paused. With the problem paused, changes can be made within the grid of nine adjustable parameters. In order for these changes to

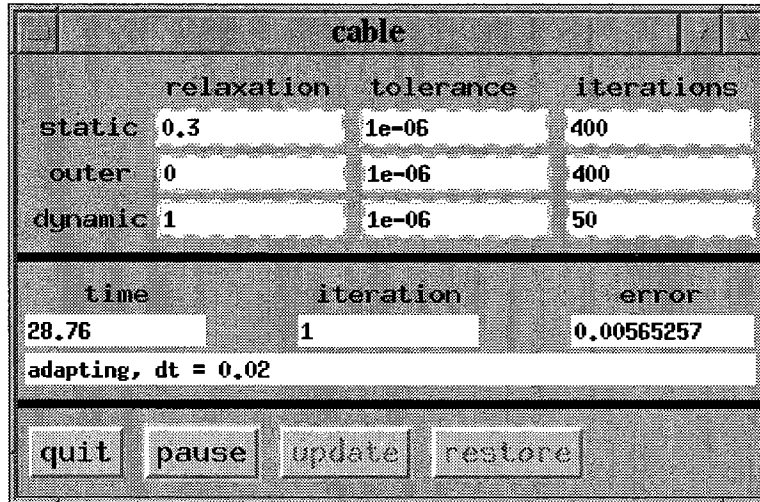


Figure 4.1: *cable*'s graphical information and control dialog.

take affect, the **update** button must be pushed and the problem unpaused. The problem can be paused and parameters adjusted any number of times. The **restore** button will reset all fields to their original values.

4.3 Using the C pre-processor

Every input file that is run through *cable* is pre-processed by the C preprocessor. This allows for the use of macro definitions and include files within an input file. This is a particularly powerful feature for users doing parametric design studies of a given system or for users who have built up large databases of cable, buoy, and connector properties.

For the parametric design study, consider the case where we have written an input file for a mooring system and we want to generate time series of the results at node 100 for a variety of system inputs – say for sinusoidal vertical excitation at 3.0 m amplitude and 4.0, 6.0, 8.0, 10.0, and 12.0 second period. In our input file then we might have an environmental description line (see chapter 3) that looks like

```
z-input = (3.0, PERIOD, 0.0)
```

When we run *cable* we just need to supply a C pre-processor macro to replace the PERIOD variable with the actual period that we want to run

```
% cable -in foo.in -out foo.res -nodes 100 -sample 0.1 -DPERIOD=4.0
```

Running the full series of excitation periods is very simple with shell constructs such as csh's `foreach` command

```
% foreach T (4, 6, 8, 10, 12)
? cable -in foo.in -out foo_${T}.res -nodes 100 -sample 0.1 -DPERIOD=${T}
? end
```

This will run, one right after another, the model for each of the five periods and store the results in files named `foo_4.res`, `foo_6.res`, etc.

To create a material database, simply create a file with just a `materials` section (the same concepts apply to other objects as well: anchors, buoys, connectors) and definition information for all of the material types that you regularly use. If you called that file `ropes.db` then all you need to do to use the database information in any input file is to include the line

```
# include "ropes.db"
```

in your input file (it must come after the `problem description` section, but other than that it can be anywhere in the input file). Because sections can be repeated in an input file you can include as many such databases as necessary and also have "local" sections defined right in the file.

4.4 Summary of command line parameters

The following are all of the available command line switches and parameter controls for *cable*.

`-in filename`

the name of the *cable* input file.

`-out filename`

the name to use in creating the output results file.

`-load filename`

if given, indicates that the static solution should be read from the results file given by `filename` (as opposed to the static solution being generated during the current run). `filename` must contain a complete (all variables present) static solution for the exact problem geometry defined by the current input file. This option is most useful for problems which require time consuming static solutions (surface problems, bottom interaction problems) and for

which numerous different dynamic inputs are being investigated. The output filename and the load filename cannot be the same.

- twoD** boolean option to use the two-dimensional algorithm for static and dynamic solutions of this problem. This is currently the default. To get the 3D solver specify **+twoD**.
- static** stops the solution process after the static solution is calculated, i.e., no dynamic solution will be generated.
- nodes n1 n2 n3 ...** specifies a list of node numbers at which temporal results will be written to the output file. Remember that for problems with positive pay-rates the total number of nodes in the problem will be greater than the total number defined by the sum of all nodes over all segments defined in the **layout** section of the input file.
- first** boolean option that adds the first node to the list of output nodes at which temporal results will be written to the output file.
- last** boolean option that adds the last node to the list of output nodes at which temporal results will be written to the output file. This can be useful for problems for which it may be difficult to manually determine what the total number of nodes will be in the problem (such as problems with positive pay-rates).
- connectors** boolean option that adds the top node of every segment with a connector defined to the list of output nodes at which temporal results will be written to the output file. This is typically the node below the connector. If you want output at the node above the connector you must explicitly specify that node number using the **-nodes** option.
- sample dt** specifies the time increment for writing temporal results to the output file. If no value is given the sample rate will be set to the time step of the dynamic analysis; this can result in the output file becoming much larger than is necessary.
- snap_dt dt** specifies the time increment at which spatial distributions of the output variables will be written to the results file. These distributions are snapshots of the output variables at all of the nodes in the problem and are generally most useful for animations. If no value is given then no snapshots will be recorded in the output.

- motion** boolean option to include motion (x, y, z coordinate information) results in the output file. The default is for this option to be on; to turn it off use **+motion**.
- vel** boolean option to include velocity (u, v, w in local coordinates) results in the output file. The default is for this option to be on; to turn it off use **+vel**.
- force** boolean option to include force (tension and shear forces) results in the output file. The default is for this option to be on; to turn it off use **+force**.
- moment** boolean option to include moment (torsion and normal and bi-normal bending moments) results in the output file. The default is for this option to be on; to turn it off use **+moment**.
- euler** boolean option to include Euler angle (2D problems) or Euler parameters (3D problems) results in the output file. The default is for this option to be on; to turn it off use **+euler**. Euler information must be included in the results file if you want to do any rotations into global coordinates during post-processing (see chapter 5).
- version** display the current version number of *cable* and exit.
- help** display a brief help message which lists all of the available command line options.
- debug** will generate a *cable* file, that if all is working well, should look exactly like the original input file. The generated file represents what the application thinks it was given.
- cpp filename** substitute *filename* for the pre-processor to run on the input file.
- nocpp** do not run the input file through the pre-processor.
- Idirectory** add *directory* to the standard search path for include files in the pre-processor.
- Uname** undefine the macro *name* in the pre-processor.
- Dname=value** define *name* to be the macro *value* in the pre-processor.

4.5 Interpreting the output from *cable*

The output files that come out of *cable* are written in a custom binary format outlined in figure 4.5. The basic layout of the file is a static solution with the user specified result variables at every node followed by an optional dynamic results section with node-time histories interleaved with full system snapshots, again only for the user specified result variables. Variables from the dynamic solution (except for absolute position: x , y , z) are always stored as dynamic quantities. That is they represent the dynamic deviation from the static value given by the static solution. Because the static solution is always present in an output file the total quantity of any variable can always be reconstructed. Remember that non-linearities in the equations of motion or boundary conditions can mean that the static solution may not always represent the true DC value of the dynamic solution.

Note that the format was designed to be a complete and compact single file container for the output from *cable*. Readability and ease of interpretation were not the primary design goals. Auxiliary tools do exist which can either interpret this format directly or convert this format into more user friendly form (see chapter 5).

'c' 'a' 'b' 'r' 'e' 's' six character unique magic number

problem type indicator 8 bits used to indicate the way that the results should be interpreted. First bit is always 1 as a redundant validity check. Second bit indicates whether or not the problem is depth referenced (and thus if a depth is stored in the output). Third bit indicates a horizontal problem, i.e., that both ends are anchored. Fourth bit indicates a towing problem, i.e., that the first end is not fixed on the bottom and that the top end is a ship. Fifth bit indicates that the solution is from a 2D algorithm. The last three bits are currently unused.

dynamic byte used as Boolean indicator for the presence of the dynamic solution in the file.

n the number of nodes in the problem as a 4-byte integer

length the number of characters in the title string as a 4-byte integer

title string the problem title string stored as an array of characters

{depth} water depth for this system as an 8-byte double. This is an optional entry - its presence is dependent on the depth reference bit of the problem type indicator byte.

output map a length 10 array of bytes, each byte being a Boolean flag indicating the presence of a single variable type in the file. The ordering is motion, velocity, force, moment, euler. The last five bytes are currently unused. Example: [1 0 1 0 1] indicates that only motion, force, and Euler parameters are contained in this file.

s(1) {x(1) y(1) z(1)} {T(1) S_n(1) S_b(1)} {M_t(1) M_n(1) M_b(1)} {B₀(1) B₁(1) B₂(1) B₃(1)}

⋮

s(n) {x(n) y(n) z(n)} {T(n) S_n(n) S_b(n)} {M_t(n) M_n(n) M_b(n)} {B₀(n) B₁(n) B₂(n) B₃(n)}

array of 8-byte doubles containing the static solution. The only variable guaranteed to be present is s, the Lagrangian coordinate of the node. The curly braces indicate variable groupings which may or may not be present depending on the information in the output map. This is the end of the file if the dynamic solution is not present.

duration the total time length of the simulation as an 8-byte double

dt the time step of the simulation as an 8-byte double

sample dt the sampling time step for the node-time histories as an 8-byte double

snapshot dt the time increment between system snapshots as an 8-byte double. If it is zero, then no snapshots are present

n_o a 4-byte integer giving the total number of output nodes for which node-time histories are stored. If the number is zero then no node-time histories are present.

output nodes a length n_o array of 4-byte integers giving the node numbers for which node-time histories are stored

{x(1) y(1) z(1)} {u(1) v(1) w(1)} {T(1) S_n(1) S_b(1)} {M_t(1) M_n(1) M_b(1)} {B₀(1) B₁(1) B₂(1) B₃(1)}

⋮

{x(n_o) y(n_o) z(n_o)} {u(n_o) v(n_o) w(n_o)} {T(n_o) S_n(n_o) S_b(n_o)} {M_t(n_o) M_n(n_o) M_b(n_o)} {B₀(n_o) B₁(n_o) B₂(n_o) B₃(n_o)}

{x(1) y(1) z(1)} {u(1) v(1) w(1)} {T(1) S_n(1) S_b(1)} {M_t(1) M_n(1) M_b(1)} {B₀(1) B₁(1) B₂(1) B₃(1)}

⋮

{x(n) y(n) z(n)} {u(n) v(n) w(n)} {T(n) S_n(n) S_b(n)} {M_t(n) M_n(n) M_b(n)} {B₀(n) B₁(n) B₂(n) B₃(n)}

Dumps of type stamped result dumps. A dump of node-time histories will start with a single 't' byte; a snapshot dump will start with a single 's' byte. There is no time stamping of either dump - they should simply be written at the appropriate time increment. The time stamp, if needed during post-processing, can be backed out from the position of a given dump in the output and the known increment between dumps.

Figure 4.2: The binary file format for *cable* results files.

Chapter 5

Post-processing *cable* Results

5.1 Using *cable* results with Matlab

The binary results files that *cable* produces can easily be converted into Matlab format using the *res2mat* application. *res2mat* reads the available results information in the *cable* output file and writes a Matlab (.mat) file containing symbolically named variables for all of the results. The results can be written to Matlab format either in local (tangential, normal, bi-normal) or global (x, y, z) coordinate system. *res2mat* can only do the transformation to global coordinates if the Euler information was written into the results file.

5.1.1 Format of the Matlab file

res2mat will convert all of the appropriate information in the *cable* results file into the Matlab file according to a few simple rules. Static information is written to variables with no subscript (\mathbf{x} , T , Mn , etc.). Node-time histories are written to variables with names subscripted by t (\mathbf{x}_t , T_t , Mn_t , etc.). Snapshots are given names subscripted with s (\mathbf{x}_s , T_s , Mn_s , etc.). The basic variable names that are used depend on whether or not the results are written to Matlab format in local or global coordinates. The range of names is detailed in table 5.1. Also included in the Matlab file are variables with the sample rate (dt), snapshot rate ($snapp_dt$), Lagrangian coordinate of each node (s), a list of output node numbers ($nodes$) and a time vector appropriate for the node-time histories (t). The water depth is stored in $depth$ if it is available within the results file.

information	2-D Results		3-D Results	
	local names	global names	local names	global names
position	x, z	x, z	x, y, z	x, y, z
velocity	u, v	U, W	u, v, w	U, V, W
force	T, Sn	Fx, Fz	T, Sn, Sb	Fx, Fy, Fz
moment	Mb	My	Mt, Mn, Mb	Mx, My, Mz
Euler	phi	phi	B0, B1, B2, B3	B0, B1, B2, B3

Table 5.1: The names that *res2mat* assigns to Matlab variables.

5.1.2 Example Matlab manipulations

The node-time history result for each variable is an $n_t \times n_o$ matrix, where n_t is the number of samples and n_o is the number of output nodes. Thus, each column of the variable contains the full time series of that variable for one node, so

```
>> plot(t, T.t(:, 3));
```

plots the tension at the third output node as a function of time.

The snapshot results for each variable are stored in an $n \times n_s$ matrix, where n is the number of nodes in the system and n_s is the total number of snapshots that were written. The tenth snapshot (at time $t = (10 - 1)\text{snap_dt}$) can be plotted simply as

```
>> plot(s, T.s(:, 10));
```

The geometric configuration of the system at every snapshot can be plotted as a “spaghetti” plot of lines on a single graph with a command like

```
>> plot(x_s, z_s)
```

If we wanted to plot the dynamic component of the horizontal position of one of our output nodes we would need to do the following

```
>> plot(t, x.t(:, 3) - x(nodes(3)));
```

The coordinate positions are always stored in absolute form so we need to explicitly subtract off the static position. Because the static variables contain information at every node (they are simply an $n \times 1$ vector) we need to use the `nodes` vector to figure out what the actual node number of the third output node was.

5.1.3 *res2mat* command line parameters

res2mat accepts the following command line switches to control its behavior.

-in results file

the name of the file containing the *cable* results.

-out matlab file

the name to use in creating the Matlab output file. The suggested extension is *.mat* simply because this is what Matlab will look for. *res2mat* does not enforce any naming convention.

-twoD

specify that the results file came from the 2D solution algorithm. The option is not strictly necessary for 2D results but it will result in a smaller Matlab file because the all zero 3D information from the results file will not be written to the Matlab file. It is required if transformation to global coordinates is requested because it affects the interpretation of the Euler information used in the transformation. This is currently the default. If the solution is from the 3D algorithm specify *+twoD*. This option is provided for backward compatibility with older format result files which do not have this information stored in the problem type indicator bit. Any specification of this option will override the information stored in that bit.

-global

Boolean option to write results to the Matlab file in global coordinates rather than the default tangential, normal, bi-normal local coordinate system that they are stored in within the results file. The transformation cannot be performed if the results file does not contain the Euler information.

5.2 The *animate* post-processing application

On X11 based workstations, a second post-processing option exists in the form of the *animate* application. *animate* reads *cable* results files directly and can produce animations showing system spatial configuration in conjunction with the spatial distribution of force, moment and velocity quantities along the cable and/or the temporal distribution of these quantities at the specifically requested output nodes. Spectra of the time series quantities can also be plotted.

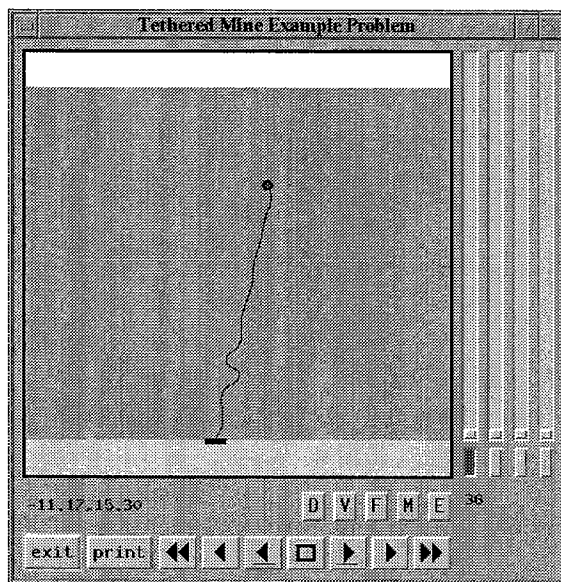


Figure 5.1: The main window of *animate*.

5.2.1 The main animation window

On start-up the *animate* main window (figure 5.1) pops up with the static configuration of the system drawn in the viewing area. Across the bottom of the window are controls for creating plots and controlling the time rate of the animation. Along the right side are four toggle button/slider pairs which control the placement of marker nodes. The marker nodes are used to indicate which of the output nodes you want to view the results for. The toggle button under each slider activates one of the markers; you can then use the slider to move the marker between output nodes by clicking and dragging on the slider thumb with the middle mouse button¹. Each marker is identified with a unique color – this is the color with which the time series or spectral results for that node will be drawn.

Which variables get plotted is controlled by the five buttons **D** (displacements), **V** (velocities), **F** (forces), **M** (moments), **E** (Euler information). If any of these buttons is engaged, a plot of the spatial distributions of those variables at the current time step will be generated. These plots show the value of a given variable as a function of Lagrangian coordinate; this is the coordinate which measures distance along the system from the first node. The first node always has Lagrangian coordinate 0 and the last node always has Lagrangian coordinate L , where L is the total length of the system. If any of the marker nodes are activated, then temporal distributions of those variables at the marked nodes

¹Most X-server software can be configured such that for two button mice, clicking both buttons at the same time emulates the middle button of a three button mouse

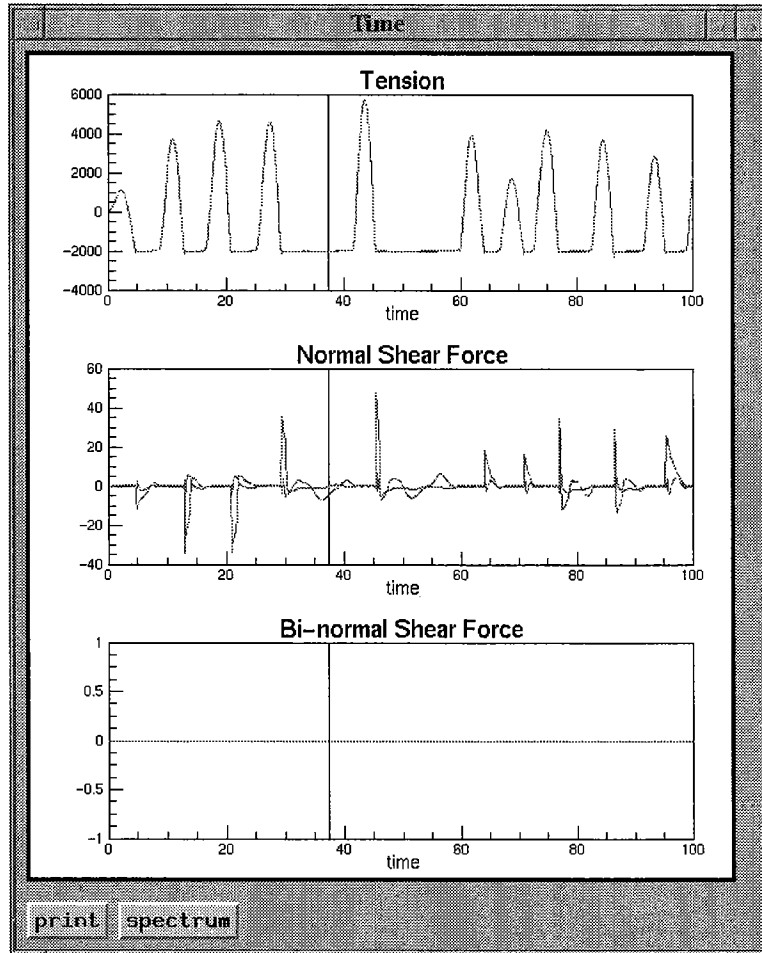


Figure 5.2: A time plot of the forces at two marked nodes.

will also be generated. A temporal plot of the forces for the animation shown in figure 5.1 is shown in figure 5.2. There are two curves on each graph because we currently have two marked nodes. The vertical black bars on each graph indicate the current time point. Spatial distribution plots are updated at the same rate as the main animation. Plots can be popped down simply by disengaging the appropriate button.

Spectra for time series results can be generated by clicking on the **spectrum** button at the bottom of the plot window. A graph window with the frequency domain analog of the time domain results plotted in that window will be automatically generated. The result for each graph in a window is based only on the results currently viewed on that graph. For zoomed graphs then, the spectra are computed using only that portion of the time series which is currently showing. Spectrum plots can be dismissed by clicking on the **dismiss**

button at the bottom of the window. The length of the FFTs used in computing the spectra is determined by fitting four windows over the data, with the data length padded to the nearest power of 2. Thus, a 500 point time series will have a spectrum computed using four 128 point windows. The spectra are plotted semi-log so the values on the y axis represent $\log_{10}(S)$.

The rate of the animation is controlled by the tape player-type buttons on the bottom-left of the main window (figure 5.1). From left to right they are: slow down (increase the time delay between frames), play the animation in reverse, back up one frame, pause the animation, go forward one frame, play the animation forward, and speed up (decrease the time delay between frames). The animation can be sped up or slowed down while it is playing. It must be paused before you can use the single frame forward and backward controls.

5.2.2 Coordinates and zooming

The coordinate pairs above the **exit** button give the x, y location of the cursor when it is moved around the main viewing area. Note that in 3D perspective view the reported coordinates are meaningless. The current time is always displayed on the right side of the window just under the marker node toggle buttons.

Zooming in the main animation window is accomplished simply by clicking with the left mouse button and dragging out a window which you want to zoom in on. Scrollbars will appear on the bottom and right side of the viewing area so that you can scroll around over the whole viewing area. The full view can be restored by clicking the right mouse button.

With the mouse in a graph window, you can click with the right mouse button to have the ordinate and abscissa value of that point reported at the bottom of the plot window. Zooming on plots is achieved by clicking the left mouse button and dragging out a rectangle that encompasses the area that you want to zoom in on. The full scale of a graph can be restored by clicking on the middle mouse button within the graph area or by pressing with the focus on the graph area and the mouse point on the appropriate graph (you may need to use to get the focus on the graph area).

5.2.3 Animate command line parameters

animate accepts the following command line options to control both how results are presented and some of the basic appearance parameters.

- in filename** the results file to interpret.

- global** boolean option to draw plotted results in global coordinates rather than the default tangential, normal, bi-normal local coordinate system that they are stored in within the results file. The transformation cannot be performed if the results file does not contain the Euler information.

- totals** boolean option to plot quantities as static + dynamic result. In normal operation only the dynamic portion is plotted for node-time histories.

- twoD** boolean option to treat the Euler information as the angle ϕ rather than the four Euler parameters – if results were written from the 2D solution algorithm then this option must be specified if rotations to global coordinates are to be performed correctly. This is currently the default. If the solution is from the 3D algorithm specify **+twoD**. This option is provided for backward compatibility with older format result files which do not have this information stored in the problem type indicator bit. Any specification of this option will override the information stored in that bit.

- realtime** specifies that the initial frame rate of the animation should match the snapshot increment. The default for this parameter is off.

- delay n** specify that the initial frame rate should be based on a delay of n microseconds. If no **-delay** option is specified and **-realtime** is off then the initial delay is 60000 microseconds.

- thick** specifies that all line drawing should be done with a thick line.

- box** specifies that a reference box (in 3D) or surface and bottom (2D) should be drawn as visual aids in interpreting the problem. The default is on; to turn off box drawing use **+box**.

- drifter** specifies that the problem should be interpreted as a drifter or towing problem. A free surface, but no bottom, will be drawn if reference box drawing is enabled. Any depth specification will be ignored; the free surface will be drawn at the static position of the last node in the system. This option is automatically activated if the appropriate bit in the problem type indicator byte of the results file indicates that this is a towing problem.

- ship** boolean flag to enable tow ship drawing for drifter problems. This will place a ship graphic at the last node in the problem. This option is automatically

activated if the appropriate bit in the problem type indicator byte of the results file indicates that this is a towing problem.

- `-color` boolean flag to indicate that contrasting colors should be used to draw the backgrounds in the main animation window. The default is for this flag to be on and sky to be white, water to be cyan and bottom to be brown (or wheat). For problems where you know that you do not want print-outs in color (or shades of grey when colors get translated by b/w laser printers) you can turn colors off using `+colors`.
- `-threeD` specifies that the animation should be done in 3D perspective view. The default is off even if the solution is from the 3D solution algorithm. When this parameter is on, an auxiliary control window with sliders for rotations and scaling will pop-up along with the main window.
- `-control` specifies that the 3D rotation and scaling controls should be activated for problems drawn in 3D perspective view (`-threeD`). The default is on.
- `-depth H` activates the drawing of a free surface at a depth H . This parameter overrides the information that may have been stored in the results file. If no depth is specified and the results file does not contain a depth reference then no free surface will be drawn. This option is provided for backward compatibility with older format result files which do not have this information stored in the problem type indicator information.
- `-magnify M` multiply the dynamic displacements from the static configuration by a factor of M . This is often the only way to tell that system components are moving during the animation of a system with large length scales and small excitation.
- `-anchors n1 n2 ...`
draw anchor symbols at the nodes given by $n1$, $n2$, etc. The anchor symbol is currently a black rectangle. If no buoys or anchors are specified then anchors will be drawn at nodes appropriate to the information stored in the problem type indicator of the results file.
- `-buoys n1 n2 ...`
draw buoy symbols at the nodes given by $n1$, $n2$, etc. The buoy symbol is currently a filled black sphere slightly larger than the circles used for node markers. If no buoys or anchors are specified then anchors will be drawn at nodes appropriate to the information stored in the problem type indicator of the results file.

- lbs** boolean flag to provide the very common conversion of force units from Newtons to pounds. When this flag is activated all of the force quantities (tension, shear, global horizontal and vertical forces) will be scaled by $\frac{1}{4.4482216}$. The default is off.
- yz** draw the 2D Y-Z plane of a 3D problem.
- xrot x** the initial x-axis rotation for 3D perspective view. The default is -20° .
- yrot y** the initial y-axis rotation for 3D perspective view. The default is 40° .
- zrot z** the initial z-axis rotation for 3D perspective view. The default is 0° .
- zscale s** the initial z-axis scaling (“eye distance”) for 3D perspective view. The default is 0.4.

5.3 ASCII output

The post-processing application *res2asc* can be used to convert the static parts of the *cable* binary output file format into tabular ASCII data. No capability currently exists within *res2asc* to tabulate dynamic results (either node time histories or snapshots).

5.3.1 *res2asc* command line parameters

res2asc accepts the following command line switches to control its behavior.

- in results file**
the name of the file containing the *cable* results.
- out ASCII output file**
the name to use in creating the ASCII output file.
- twoD** specify that the results file came from the 2D solution algorithm. Unlike *res2mat* this option must be on if you want to refer to variables by their 2D names (i.e., phi rather than B0). This is currently the default. If you want to refer to variables by their 3D names specify **+twoD**.
- variables v1 v2 v3 ...**
list of variable names that you want output into the table. Acceptable names are *s*, *x*, *y*, *z*, *T*, *Sn*, *Sb*, *Mt*, *Mn*, *Mb*, *B0*, *B1*, *B2*, *B3*, *phi*.

Chapter 6

cable's Windows Interface

6.1 Introduction

The Windows version of *WHOI Cable* includes an encapsulator application for all of the component programs discussed thus far. The encapsulator combines an editor for building problem description files with facilities for executing *cable*, *animate*, and *res2mat* to solve the problem and post-process the results, all from within a single Windows 95 or Windows NT based application. Figure 6.1 illustrates this interaction between the various component programs.

The main editor window, with one of the example problems loaded, is shown in figure 6.2. Across the top of the window is the main menu bar, with the usual menu entries for file and edit control and some special entries for solving problems and viewing results. There is also a toolbar below the main menubar which contains shortcut buttons for the items on the **Insert** menu and for invoking the various component programs.

6.2 Building an input file

There are several ways to go about constructing an input file for a new model. Starting with a blank editor (either at start-up or by selecting **New** from the **File** menu) you can write the file from scratch, open an existing problem and modify it to match the new problem, or you can build the file up from the template blocks that the encapsulator provides.

Templates are available either from the **Insert** menu or from the toolbar. When selected, a template is placed at the current insertion (cursor) point of the editor. Once placed, the entries in the template must be edited to match the problem that you are describing. Templates are available for the basic sectional layout of a problem (complete problem descrip-

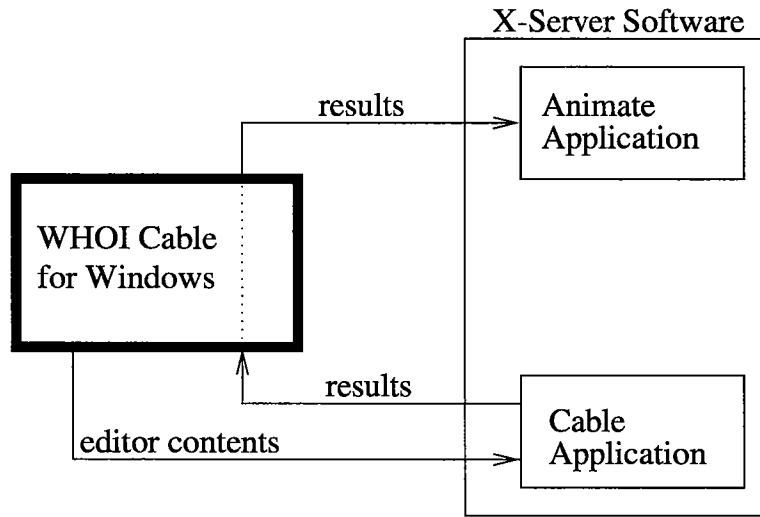


Figure 6.1: The relationships between the *WHOI Cable* component programs.

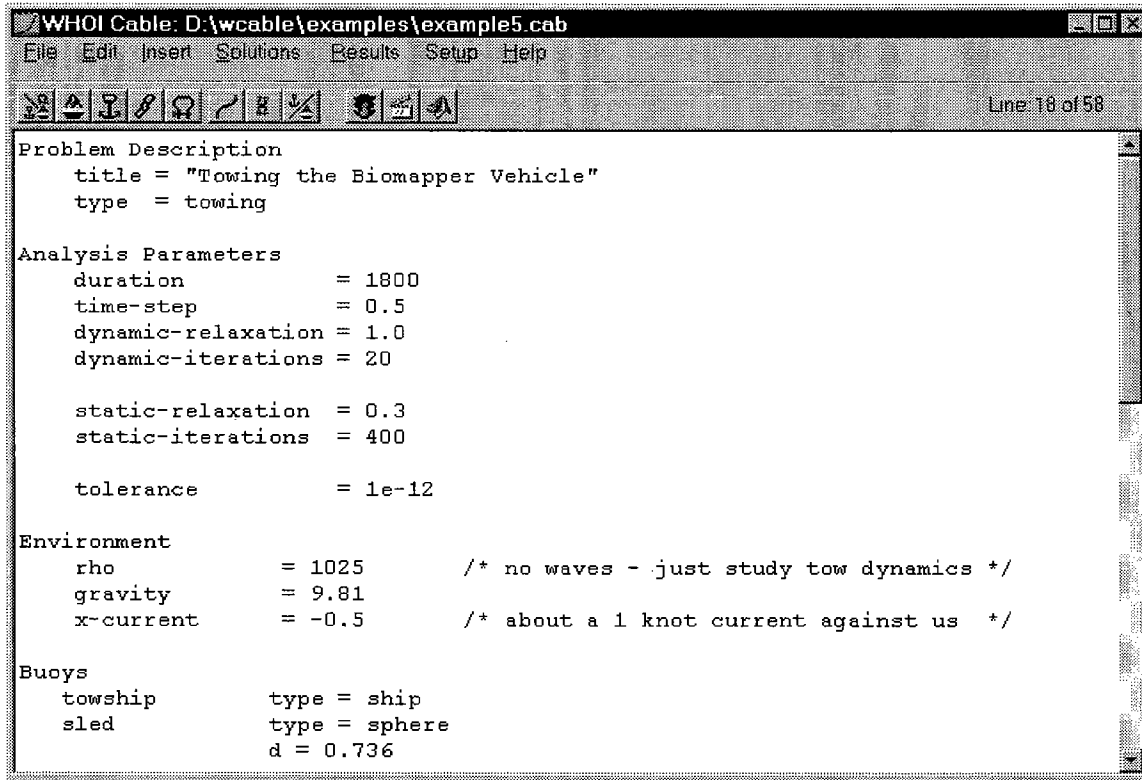


Figure 6.2: The main window of the *WHOI Cable* Windows interface.

tion and analysis parameters sections and section headers for buoys, anchors, connectors, materials, and layout), object definitions (buoys, anchors, materials, connectors), and the components of a layout (segments, connectors, terminals). Once inserted, templates can be deleted, edited, and moved just like any other text in the editor.

Each assignment in a template that requires a value is marked with `xxx` for values that require real numbers, `nnn` for values that require integer numbers, or `x_name_x` for cases where you must specify or assign a symbolic name (remember that names with spaces in them must be enclosed in double quotation marks). In the `analysis parameters` and `environment` sections that are placed by the sections template, several typical values have already been set with actual numbers. These values should be acceptable for most cases, but you should feel free to change them to better suit your exact problem.

You can get help on a keyword in a template by highlighting the word or words in the main editor and selecting **Keyword** from the **Help** menu (or by pressing `F1`). This will display the *WHOI Cable* syntax and keyword help dialog with a paragraph or two describing the highlighted keyword. If there are multiple ways in which the same keyword can be used, press the **Next** button on the help dialog to move through them.

6.3 Solving a problem

In the command-line version of *cable*, described in chapter 4, the details of the solution are controlled by command-line switches. In the Windows interface those switches are replaced by the checkboxes and text fields of the solution control dialog pictured in figure 6.3. You can view this dialog by selecting **Controls** from the **Solutions** menu.

The controls in the upper left frame determine the basic solution type – 2D or 3D, static or dynamic. If you have a results file that already contains a valid static solution for the current problem, then you can specify that *cable* should use that solution rather than generating a new static solution to use as the initialization for the dynamic solution. The bottom left frame provides control over which variables are included in the output file.

The controls in the **Dynamic Results** frame are only enabled when a dynamic solution is requested and define the sampling rates and nodes for the dynamic output. The list of output nodes is constructed by typing node numbers in the box at the top of the list and pressing **Add** (or pressing `return`). Nodes can be deleted by highlighting them within the list and clicking **Remove**. **Clear** deletes all entries in a list. The first node, last node, and nodes associated with a connector between segments can be automatically included in the output list (without explicitly specifying their node numbers) by clicking the check boxes below the list of output nodes. At least one form of dynamic output control must be specified for any dynamic solution of a problem. If a time series time step is given then

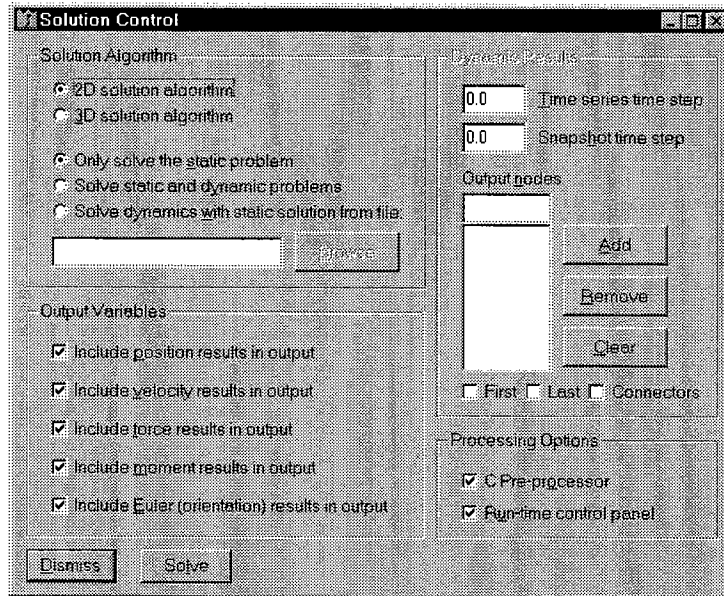


Figure 6.3: The solution control dialog available by selecting **Controls** from the **Solutions** menu.

a list of output nodes must also be specified. If no time series time step is given then a snapshot time step must be specified.

The **Solve** button on the dialog is a shortcut to the **Solve** selection on the **Solutions** menu. See sections 4.1 and 4.4 for additional information on what the various options control.

Once a problem description is constructed, and the appropriate control options have been selected, you can solve the problem simply by selecting **Solve** from the **Solutions** menu (also by pressing the **Solve** button on the control dialog, using the keyboard shortcut **ctrl-L**, or using the menu shortcut **alt-S** followed by **alt-S**, see section 6.6 for a complete list of the different ways to accomplish most tasks). This saves the current editor to a temporary file and invokes *cable* on that file. *cable*'s output is directed to a second temporary file. When the solution is complete, the temporary input file is deleted and program control returns to the main editor. See section 6.5 for details on how and when you should save files and how temporary files are used within the encapsulator.

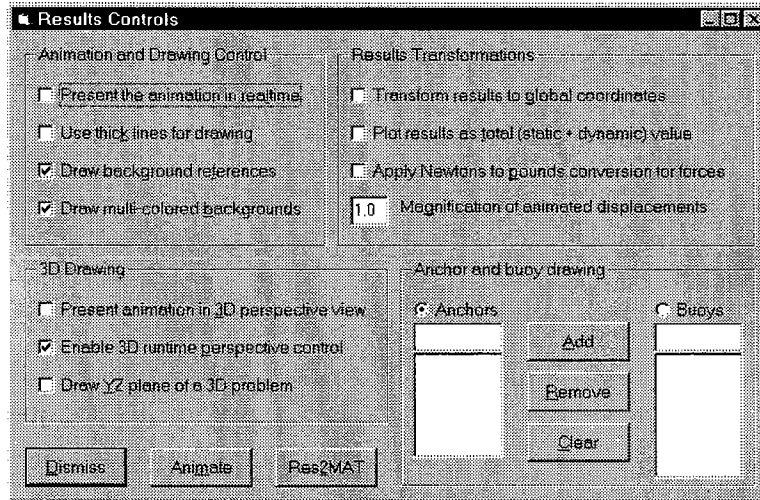


Figure 6.4: The results control dialog available by selecting **Controls** from the **Results** menu.

6.4 Viewing and converting results

Post-processing the results of a solved problem is as easy as selecting **Animate** from the **Results** menu. This will invoke the *animate* program with the current output file as input. Details of using *animate* are provided in section 5.2. The dialog to control how *animate* is invoked is shown in figure 6.4 and is raised by selecting **Controls** from the **Results** menu.

The controls in the upper left frame of figure 6.4 dictate the drawing and animation parameters that *animate* will use to present the results. The **3D Drawing** options control whether a 3D problem gets drawn in 3D perspective view or is simply projected onto the x-z plane. The controls in the **Results Transformations** frame govern how the result variables should be plotted – local or global coordinate system, total (static + dynamic) or just dynamic values, and whether force units should be converted from Newtons to pounds (this only makes sense if the original force units are in Newtons of course). *animate* will draw additional anchor and buoy symbols at the nodes indicated in the lists at the bottom right of the dialog. The **Add**, **Remove**, and **Clear** buttons apply to both anchor and buoy lists, but only to one list at a time. The active list is controlled by the **Anchors** and **Buoys** toggle buttons over the lists. To add a node number to a list simply type the number into the box at the top of the list and click **Add** (or press `return`). Nodes can be deleted by highlighting them within the list and clicking **Remove**. **Clear** deletes all entries in a list.

Two of the switches on this control (transformations for global coordinates and total

results) are also used when invoking *res2mat*. Conversion from *cable* results format to Matlab *.mat* format is done by selecting **Matlab conversion** from the **Results** menu. A file selection dialog will appear asking you to specify the name of the Matlab file to create.

6.5 Working with files

When working with the encapsulator it is important to keep track of two files. The first is the current input file that is contained in the main editor. The contents of the editor are saved to a temporary file during each solve procedure, but this temporary file should never be used as your own record of the problem (it is deleted as soon as the solution is completed). To assign a name to it and save it, select **Save As** from the **File** menu. If you have already assigned a name and simply want to save (the current name is shown in the titlebar of the main window, as in figure 6.2), then select **Save** from the **File** menu.

The second file type is the current output file. When a problem is solved the output is directed to a temporary file. That file then defines the current result. If you want to assign a name to the file and save it then select **Save Result As** from the **File** menu. If you have already defined a current output name then you can simply select **Save Result**. If you do not explicitly save an output file before you execute another solve process then that output file will be deleted and the result of the latest solve will become the current output. Note that you do not need to assign a name to an output file to view or convert the results; you only need to assign a name if you want to preserve results before doing additional solutions or exiting the program.

You can define an existing file as the current result (i.e., without doing a solve) by selecting **Load Result** from the **File** menu. Any post-processing selections will then refer to this already existing file. Note that this name becomes the current output name for the **Save Result** action so that any subsequent solves and solution saves will overwrite that pre-existing result.

6.6 Command reference

Between the main menu, keyboard shortcuts, and the toolbar, there are generally several ways to accomplish any one task from within *WHOI Cable's* Windows interface. Table 6.1 details this complete command structure.

Menu	Item	Menu key	Keyboard shortcut	Toolbar icon	Other
File	New	Alt-f, Alt-n			
	Open	Alt-f, Alt-o	Ctrl-o		
	Save	Alt-f, Alt-s	Ctrl-s		
	Save As	Alt-f, Alt-a	Ctrl-a		
	Load Result	Alt-f, Alt-l	Ctrl-d		
	Save Result	Alt-f, Alt-r			
	Save Result As	Alt-f, Alt-v			
	Print Setup	Alt-f, Alt-u			
	Print	Alt-f, Alt-p	Ctrl-p		
	Exit	Alt-f, Alt-x			
Edit	Undo	Alt-e, Alt-u	Ctrl-u		
	Cut	Alt-e, Alt-t	Ctrl-x		
	Copy	Alt-e, Alt-o	Ctrl-c		
	Paste	Alt-e, Alt-p	Ctrl-v		
	Find	Alt-e, Alt-f			
	Find Next	Alt-e, Alt-n	F3		
Insert	Section Template	Alt-i, Alt-t		chain+buoy+anchor+shackle	
	Buoy	Alt-i, Alt-b		buoy	
	Anchor	Alt-i, Alt-a		anchor	
	Material	Alt-i, Alt-m		chain	
	Connector	Alt-i, Alt-c		shackle	
	Layout segment	Alt-i, Alt-s		cable shot	
	Layout connector	Alt-i, Alt-o		shackle+shackle	
	Layout terminal	Alt-i, Alt-l		buoy+anchor	
Solutions	Controls	Alt-s, Alt-c	Ctrl-l		
	Solve	Alt-s, Alt-s	Ctrl-r	go light	button on control box
Results	Animate	Alt-r, Alt-a	Ctrl-n	movie	button on results box
	Matlab conversion	Alt-r, Alt-m	Ctrl-m	matlab	button on results box
Setup	Controls	Alt-r, Alt-c	Ctrl-t		
	Files	Alt-u, Alt-f			
Help	Fonts	Alt-u, Alt-n			
	Keyword	Alt-h, Alt-k	F1		
	About	Alt-h, Alt-a			

Table 6.1: Complete command structure for the *WHOI Cable* for Windows encapsulator.

6.7 Installing *WHOI Cable* for Windows

6.7.1 System requirements

WHOI Cable for Windows is only available for 32-bit Windows (95 or NT), Intel-based platforms. Most static problems can be solved in a reasonable amount of time on any Pentium or even a fast 486 processor. Dynamic problems and some static problems (notably those that use small static relaxation factors or require significant numbers of outer iterations) are best solved on faster Pentium or Pentium Pro architectures.

As illustrated in figure 6.1, PC X-server software must be installed if you want to take full advantage of all of the component programs. Without an X-server, *cable's* graphical control and information dialog (figure 4.1) and *animate's* post-processing capabilities are not available. Numerous companies market inexpensive PC X-server software¹.

¹There is a free, albeit somewhat limited, server available at <http://www/microimages.com/freestuff>. There is a reasonably complete list of commercial vendors, along with a review of four of them at <http://www.sun.com/sunworldonline/swol-11-1995/swol-11-pcx.html>

6.7.2 Installation instructions

Because the main Windows interface to *WHOI Cable* is simply an encapsulator for the rest of the component programs, it requires that standard versions of the *cable* component programs, compiled for 32-bit Windows, be installed on your computer. These programs and the supporting DLLs are provided as part of the standard distribution of *WHOI Cable* for Windows. They, along with the actual encapsulator application, examples, and support files are installed using the provided setup utility. If you received the distribution on disks, insert disk 1 into your floppy drive, select **Run** from the Windows **Start** menu and enter `a:\setup.exe` when prompted for the application name. If you received the distribution in a single ZIP file, copy the ZIP file to a temporary directory, unpack it using a utility such as Info-Zip's *unzip*², and execute `setup.exe`. After installation you can safely remove the ZIP file and its contents.

Your PC X-server software should generally be running before you run *WHOI Cable* (at the very least it needs to be running before you try to solve a problem or view any results). In order for the encapsulator to interact with the X-server, you should also insure that your `DISPLAY` environment variable is set. You can do this using the **System** icon followed by the **Environment** tab available under the Control Panel of Windows NT or by adding the line `set DISPLAY=foo:0.0` (where `foo` is the name of your machine) to your `autoexec.bat` file under Windows 95.

6.7.3 Printing from *animate* under Windows

Printing directly to a printer from *animate* running under Windows requires that you specify a valid print device (`lpt1`, `lpt2`, `com1`, etc.) and that this device name be mapped to a Postscript printer. If you use a network printer rather than a printer connected directly to your computer then you must map the network printer name to one of the standard MS-DOS printer device names. To do this under Windows NT you use the `net use` command from a command prompt. For example,

```
net use lpt1 \\server\ps_printer /persistent:yes
```

maps the `lpt1` device to a printer named `ps_printer` that is connected to the computer named `server`.

If you do not have a Postscript printer available to you then we recommend that you use the print to file option in *animate* and install the freely available³ ghostscript package for printing and viewing Postscript files on non-Postscript devices.

²freely available from `ftp://ftp.cdrom.com/pub/simtelnet/msdos/00_start/unz531x.exe`

³from `ftp.cdrom.com` in `pub/simtelnet/win95/print` for example

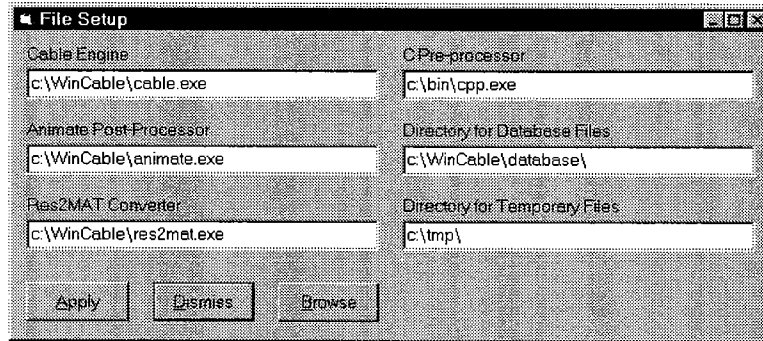


Figure 6.5: Configuring the pathnames for the *WHOI Cable* component programs. The File Setup dialog is raised by selecting **Files** from the **Setup** menu

6.7.4 Modifying the installation

6.7.4.1 File and pathnames

After installation, you can change the paths to any of the component programs or specify replacement programs for those components by selecting **Files** under the **Setup** menu. The dialog shown in figure 6.5 allows you to specify locations for the *cable*, *res2mat*, *animate* and *cpp* programs. You can also set the directory for database template (*.ctm) and object (*.db) files (this is the directory that will be used as the C pre-processor search directory). The directory for temporary files created during the solution of a problem can also be set. Directory names should end with a trailing \ as shown in figure 6.5.

6.7.4.2 Templates

The template blocks inserted by the selections on the **Insert** menu are contained in a series of files located in the *WHOI Cable* database directory. These files can be customized with any text editor as long as their filenames are not changed.

Appendix A

Subsurface Mooring Example

The following is an example *cable* input file for a simple, single segment system. The problem is a buoyant sphere attached to 20 m of rope in 25 m of water. We want to explore the motions of the sphere and the tensions in the rope as a regular wave with 0.8 m amplitude and 8.0 second period passes over. We will use Morison's equation to calculate the wave-induced forcing on the sphere. Because we are expecting lots of curvature as the rope buckles, we have used a relatively high node density (120 nodes over 20 m of rope). The current specification illustrates the use of a continuous expression to model an exponentially decaying current profile.

Problem Description

```
title = "Tethered Mine Example Problem"  
type = subsurface
```

Analysis Parameters

```
duration      = 60.0  
time-step     = 0.1  
relaxation    = 1.0  
max-iterations = 20  
tolerance     = 1e-6
```

Environment

```
forcing-method = morison  
input-type     = regular  
x-wave         = (0.8, 8.0, 0.0)  
x-current      = 0.2*exp(-0.055*H)  
rho            = 1027  
gravity        = 9.81  
depth          = 25
```

Buoys

```
mine      type = sphere
          d = 1.5      m = 1611
          Cdn = 0.5    Cdt = 0.5
```

Anchors

```
clump
```

Materials

```
rope      EA = 5.0e4    EI = 0.4      GJ = 0.04
          m = 0.09     am = 0.08    wet = 0.087
          d = 0.01     Cdt = 0.01   Cdn = 1.5
```

Layout

```
terminal = {
  anchor = clump
}
segment = {
  length = 20.0
  material = rope
  nodes = (120, 1.0)
}
terminal = {
  buoy = mine
}
```

End

Example results from this model are shown in figures A.1 and A.2. The command-lines that were used in generating these results were

```
% cable -in example1.in -out example1.res -nodes 120 -sample 0.1 -snap_dt
2.0
% animate -in example1.res -thick
```

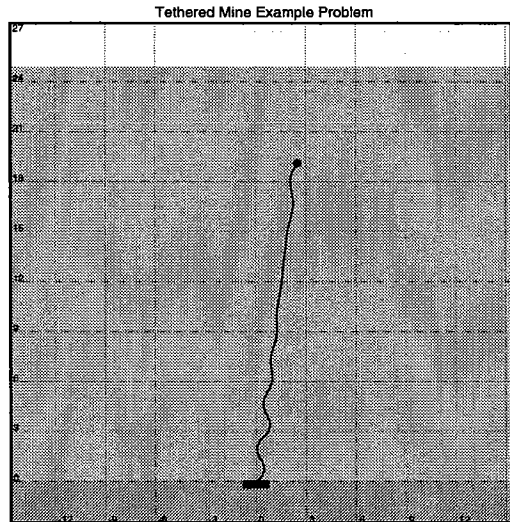


Figure A.1: The configuration of the system at $t = 26$ seconds in the simulation. This output was created using the **Print** button on the main *animate* window.

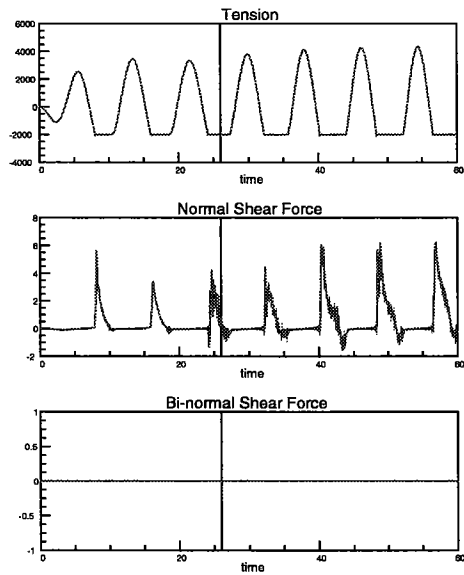


Figure A.2: The time history of forces for node 120 (the only node recorded in the results file) as printed by *animate*.

Appendix B

Shallow Water Surface Mooring

This example illustrates a typical shallow water surface mooring. It consists of an anchor, 6 feet of chain, 490 ft of wire rope, 6 feet of chain and a simple cylindrical surface buoy. We are careful to include `bottom-stiffness` and `bottom-damping` terms in the `environment` section to accomodate the chain and rope that will be on the bottom.

Problem Description

```
title = "Shallow Water Surface Mooring"
type = surface
```

Analysis Parameters

```
duration           = 120.0
time-step          = 0.01
dynamic-relaxation = 1.0
static-relaxation  = 0.01
static-outer-iterations = 25
static-outer-relaxation = 0.8
static-iterations  = 800
dynamic-iterations = 50
tolerance           = 0.01
ramp-time          = 16.0      /* minimize transients */
```

Environment

```
rho                = 1025
gravity             = 9.81
x-current           = 0.5*0.514
depth              = 100
input-type          = random      /* use random input spectrum */
forcing-method      = wave-follower
x-wave             = (48*0.0254, 8.0, 0.0) /* 8 ft sig height, 8 sec period */
```

bottom-stiffness = 100.0
bottom-damping = 1.0

Buoys

float type = cylinder
 m = 23
 h = 8*0.0254
 d = 36*0.0254
 Cdn = 1.0

Anchors

clump

Connectors

shackle d = 0.375*0.0254
 wet = 4.0
 Cdn = 0.0
 m = 0.5

Materials

chain	EA = 5.5e7	EI = 0.01	GJ = 0.1
	m = 4.4	am = 0.584	wet = 38.1
	d = 0.0265	Cdt = 0.01	Cdn = 1.0
wire	EA = 4.7e6	EI = 20	GJ = 5
	m = 0.320	am = 0.08	wet = 2.64
	d = 0.01	Cdt = 0.01	Cdn = 1.5

Layout

```
terminal = {  
  anchor = clump  
}  
segment = {  
  length = 72*0.0254  
  material = chain  
  nodes = (25, 1.0)  
}  
connector = shackle  
segment = {  
  length = 150  
  material = wire  
  nodes = (150, 1.0)
```

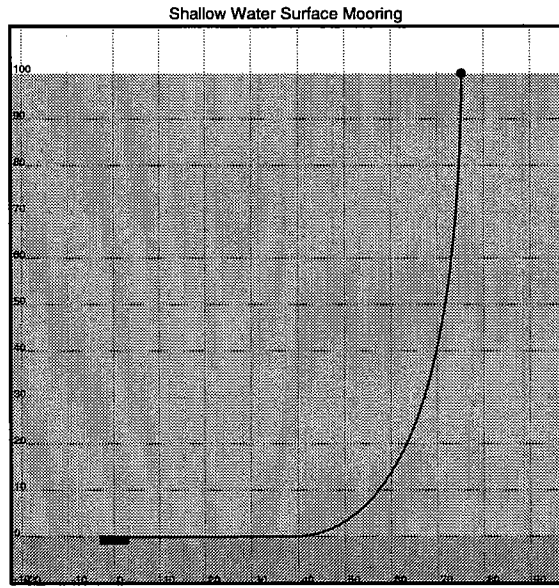


Figure B.1: The static configuration of the system as printed by *animate*.

```

}
connector = shackle
segment = {
    length = 72*0.0254
    material = chain
    nodes = (25, 1.0)
}
terminal = {
    buoy = float
}

End

```

Example results from this model are shown in figures B.1 and B.2. The command-lines that were used in generating these results were

```

% cable -in example2.in -out example2.res -nodes 1 200 -sample 0.1
% animate -in example2.res -thick
% res2mat -in example2.res -out example2.mat

```

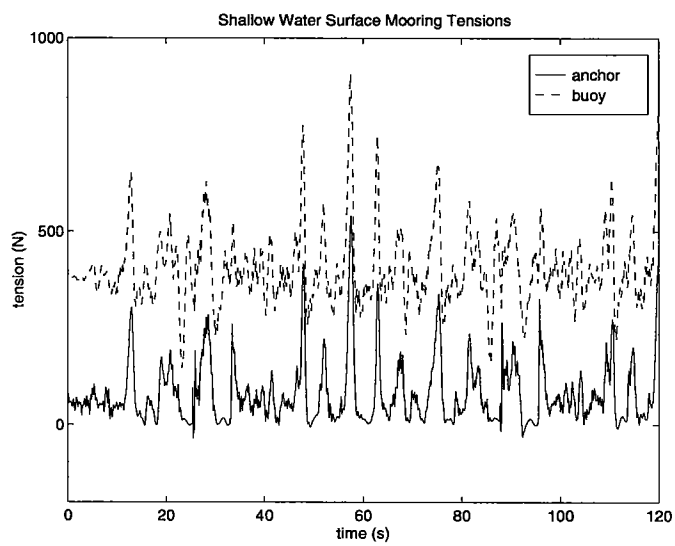


Figure B.2: The time history of total tension at the buoy and the anchor. This plot was generated from within matlab, `plot(t, T_t(:,1) + T(1), t, T_t(:,2) + T(200), '--')`.

Appendix C

Deep Water S-tether Mooring

This is an example of a complex oceanographic mooring in deep water. The current specification exemplifies the use of a discrete expression to model a linearly varying current profile.

Problem Description

```
title = "AOSN Labrador Sea Mooring"  
type = surface
```

Analysis Parameters

```
static-outer-iterations = 1000  
static-iterations      = 5000  
static-relaxation      = 0.1  
static-tolerance       = 0.001  
static-outer-tolerance = 0.01  
  
duration               = 400.0  
time-step              = 0.1  
dynamic-tolerance      = 1e-6  
dynamic-relaxation     = 1.0  
dynamic-iterations     = 20  
ramp-time              = 50.0
```

Environment

```
rho                   = 1025  
gravity              = 9.81  
x-current            = (0.0, 0.1) (100, 0.1) (1000, 0.05) (3500, 0.05)  
depth                = 3500  
input-type           = random  
forcing-method       = wave-follower
```

```

x-wave          = (2.3, 10.5, 0.0) /* sea state 6 */
bottom-stiffness = 0.0
bottom-damping  = 0.0

```

Buoys

```

float          type = sphere
               m    = 65
               d    = 1.22
               Cdn  = 1.0

```

Anchors

```

clump         color = red

```

Connectors

```

shackle       d = 0.375*0.0254
               wet = 8.0
               Cdn = 0.1
               m  = 1.0
ssf           d = 1.628
               m  = 931.3
               wet = -13350
               Cdn = 2.068

```

Materials

wire	EA = 4.4e6 m = 0.160 d = 0.0063	EI = 500 am = 0.05 Cdt = 0.01	GJ = 25 wet = 1.15 Cdn = 1.5
array	EA = 4.4e6 m = 1.484 d = 0.028	EI = 500 am = 0.631 Cdt = 0.01	GJ = 25 wet = 8.514 Cdn = 2.0
chain	EA = 5.5e7 m = 4.4 d = 0.0265	EI = 0.01 am = 0.584 Cdt = 0.01	GJ = 0.1 wet = 38.1 Cdn = 1.0
pole	EA = 5.5e7 m = 4.4 d = 0.0265	EI = 0.01 am = 0.584 Cdt = 0.01	GJ = 0.1 wet = 38.1 Cdn = 1.0
b_tether	EA = 1.0e7 m = 4.160 d = 0.05	EI = 800 am = 2.08 Cdt = 0.01	GJ = 40 wet = -4.45 Cdn = 1.5

w_tether	EA = 1.0e7	EI = 800	GJ = 40
	m = 4.160	am = 2.08	wet = 2.67
	d = 0.05	Cdt = 0.01	Cdn = 1.5
snubber	EA = 3.13e4	EI = 1000	GJ = 500
	m = 14.07	am = 9.74	wet = 42.46
	d = 0.11	Cdt = 0.01	Cdn = 1.5

Layout

```

terminal = {
  anchor = clump
}
segment = {
  length = 2900
  material = wire
  nodes = (200, 1.0)
}
connector = shackle
segment = {                                /* this is the pinger cage/spare pole */
  length = 4
  material = pole
  nodes = (5, 1.0)
}
connector = shackle
segment = {
  length = 100
  material = wire
  nodes = (20, 1.0)
}
connector = shackle
segment = {                                /* this is the docking station plus some */
  length = 6.5                               /* stuff fudged above and below it */
  material = pole
  nodes = (5, 1.0)
}
connector = shackle
segment = {                                /* this is the EM cable - what is it? */
  length = 400
  material = array
  nodes = (50, 1.0)
}
connector = shackle
segment = {                                /* 3 m of potted chain below SSF */

```



```

    length = 3
    material = chain
    nodes = (5, 1.0)
}
connector = ssf
segment = {
    length = 3
    material = chain
    nodes = (5, 1.0)
}
connector = shackle
segment = {
    length = 75
    material = b_tether
    nodes = (75, 1.0)
}
connector = shackle
segment = {
    length = 75
    material = w_tether
    nodes = (75, 1.0)
}
connector = shackle
segment = {
    length = 15
    material = snubber
    nodes = (30, 1.0)
}
terminal = {
    buoy = float
}

```

End

Example results from this model are shown in figures C.1. The command-lines that were used in generating these results were

```

% cable -in example3.in -out example3.res -nodes 230 470 -sample 0.1
% res2mat -in example3.res -out example3.mat -global

```

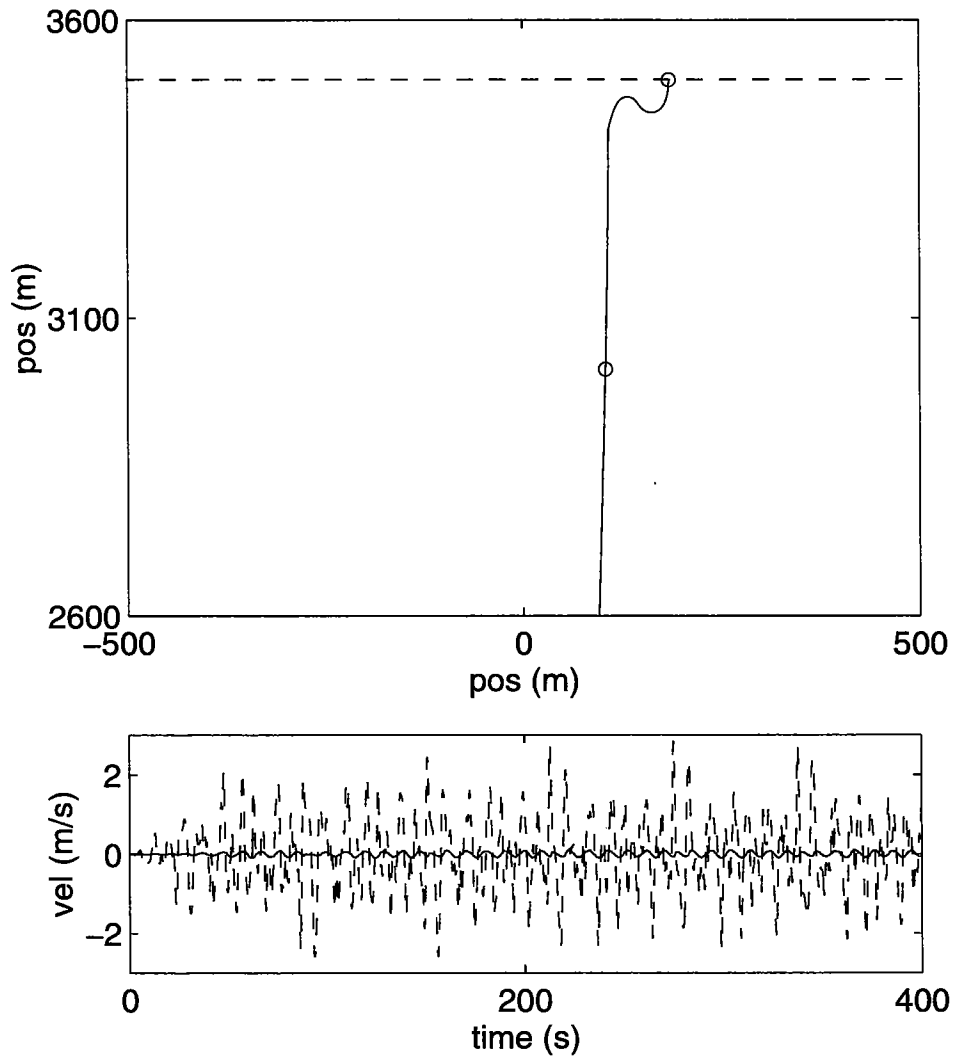


Figure C.1: The static configuration of the system and a comparison of motion at the surface buoy (dashed line) and docking station (solid line) to illustrate the effectiveness of the S-tether section at reducing vertical motions of the subsurface components. This plot was created within Matlab.

Appendix D

Horizontal Array Mooring

This is an example of a system with both ends anchored on the bottom. To define the geometry of the second anchor we simply need to give an anchor name in the last terminal definition along with the geometric location of the second anchor. This problem illustrates the use of both connectors and attachments in the geometry description. The temperature pods are located along the segments through the use of `attachment` statements and local (referenced to individual segments) node numbers.

Problem Description

```
title = "Horizontal Array"
type = horizontal
```

Analysis Parameters

```
duration          = 50.
time-step         = 0.1
static-relaxation = 1.0
static-iterations = 2000
static-outer-iterations = 1000
dynamic-iterations = 20
dynamic-relaxation = 1.0
tolerance         = 1e-6
```

Environment

```
forcing-method = morison
input-type    = random          /* random waves with 1 m ampl */
x-wave       = (1., 8.0, 0.0) /* and 8 second period */
rho          = 1025
gravity      = 9.81
x-current    = 0.5              /* about 1 knot of uniform current */
```

depth = 100

Anchors

clump

Connectors

sphere_48in d = 1.2307
wet = $-1590*4.448 + 4.45*4.448*.8667$
Cdn = 0.7944
m = $312.9 + 2*4.45/2.205$

tp_pod m = 4.91
wet = $5.3*4.448$
Cdn = 1.0
d = 0.30

termination_B m = $4.45/2.205$
wet = $4.45*4.448*.8667$
Cdn = 0.5
d = 0.168

acm_3d m = $20.65 + 2.31*2 + 6.88 + 3.18*2$
wet = $94.34 - 39.60*2$
Cdn = 1.0
d = $0.585 + .041*2$

Materials

wire_10mm	EA = 4.7e6	EI = 20	GJ = 5
	m = 0.320	am = 0.08	wet = 2.64
	d = 0.01	Cdt = 0.01	Cdn = 1.5
chain	EA = 5.5e7	EI = 0.01	GJ = 0.1
	m = 4.4	am = 0.584	wet = 38.1
	d = 0.0265	Cdt = 0.01	Cdn = 1.0
acoustic_release	EA = 1.0e8	EI = 1000	GJ = 100
	m = 44/2.5	am = 18/2.5	wet = 187/2.5
	d = 0.218	Cdt = 0.18	Cdn = 1.2

Layout

```
terminal = {  
  anchor = clump  
}  
segment = {  
  length = 5  
  material = chain  
  nodes = (5, 1.0),  
}
```

```

connector = termination_B
segment = {
    length = 108.1
    material = wire_10mm
    nodes = (50, 1.0)
}
connector = sphere_48in
segment = {
    length = 22.
    material = wire_10mm
    attachments = tp_pod : (11)
    nodes = (22, 1.0)
}
connector = acm_3d
segment = {
    length = 22.
    material = wire_10mm
    attachments = tp_pod : (11)
    nodes = (22, 1.0)
}
connector = acm_3d
segment = {
    length = 22.
    material = wire_10mm
    attachments = tp_pod : (11)
    nodes = (22, 1.0)
}
connector = acm_3d
segment = {
    length = 34.
    material = wire_10mm
    attachments = tp_pod : (11 22)
    nodes = (34, 1.0)
}
connector = sphere_48in
segment = {
    length = 103.2
    material = wire_10mm
    nodes = (50, 1.0)
}
connector = termination_B
segment = {
    length = 2.5
    material = chain

```

```

        nodes = (5, 1.0)
    }
connector = termination_B
segment = {
    length = 2.5
    material = acoustic_release
    nodes = (5, 1.0)
}
connector = termination_B
segment = {
    length = 5.0
    material = chain
    nodes = (5, 1.0)
}
terminal = {
    anchor = clump
    x = 260.0      z = 0.0
}

```

End

Example results from this model are shown in figures D.1 and D.2. The command-lines that were used in generating these results were

```

% cable -in example4.in -out example4.res -nodes 55 100 155 -sample 0.1 -snap.dt
0.5
% animate -in example4.res -thick -buoys 55 155

```

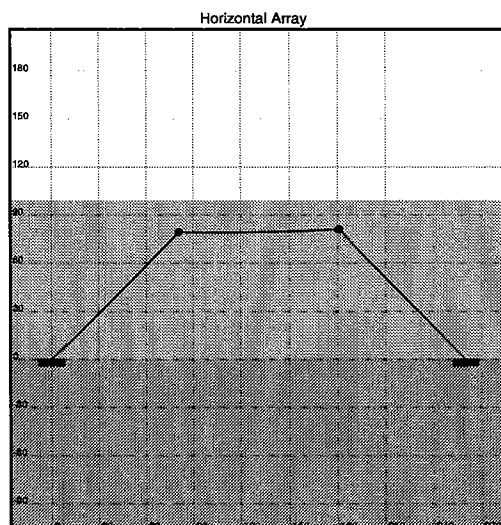


Figure D.1: The configuration of the system at $t = 36$ seconds in the simulation. This output was created using the **Print** button on the main *animate* window.

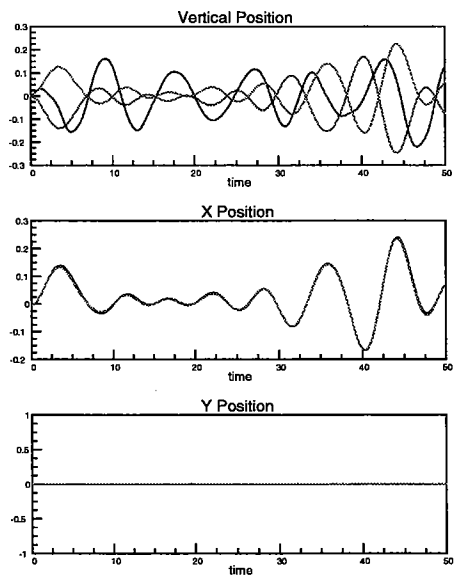


Figure D.2: The displacement history of the two corner buoys and a node along the middle of the horizontal array section as printed by *animate*.

Appendix E

Towed Vehicle Example

This example illustrates a typical towing problem. The system consists of the biomapper towed vehicle, 1000 m of tow cable and a tow ship. The model simulates a tow-yow type profile by varying both the speed of the tow ship and the pay-rate of the cable in a sinusoidal fashion. The forcing on the vehicle is a combination of both the tow-speed effects and the current which we are towing against.

Problem Description

```
title = "Towing the Biomapper Vehicle"  
type = towing
```

Analysis Parameters

```
duration          = 1800  
time-step         = 0.5  
dynamic-relaxation = 1.0  
dynamic-iterations = 20
```

```
static-relaxation = 0.3  
static-iterations = 400
```

```
tolerance         = 1e-12
```

Environment

```
rho              = 1025      /* no waves - just study tow dynamics */  
gravity          = 9.81  
x-current        = -0.5 /* about a 1 knot current against us */
```

Buoys

```
towship         type = ship  
sled            type = sphere
```

```

d = 0.736
m = 1135
buoyancy = 4448
Cdn = 0.77

```

Materials

```

cable  d = 0.0173
       m = 1.12
       wet = 8.89
       am = 0.24
       Cdn = 1.5
       Cdt = 0.01
       EA = 1.2717e7
       EI = 237.88
       GJ = 10.0

```

Layout

```

terminal = {
  buoy = sled
}
segment = {
  length = 1000
  nodes = (400, 1.0)
  material = cable
}
terminal = {
  buoy = towship
  pay-rate = -0.5*sin(0.0105*t)
  x-speed = 0.514*(5.0 + 3.0*sin(0.0105*t))
           /* come to steady state of 5 knots, then tow-yow for */
           /* 30 minutes, using both ship and winch to get yo-yo */
}

```

End

The steady state configuration of the system is shown in figure E.1. The command-lines that were used in generating these results were

```

% cable -in example5.in -out example5.res -nodes 1 -sample 0.5 -snap.dt 5.0
% animate -in example5.res -thick
% res2mat -in example5.res -out example5.mat -global

```

Figure E.2 shows the speed of the tow ship and the depth of the towed-body as a function of time. The plot of body depth also includes traces for a run in which the winch was not

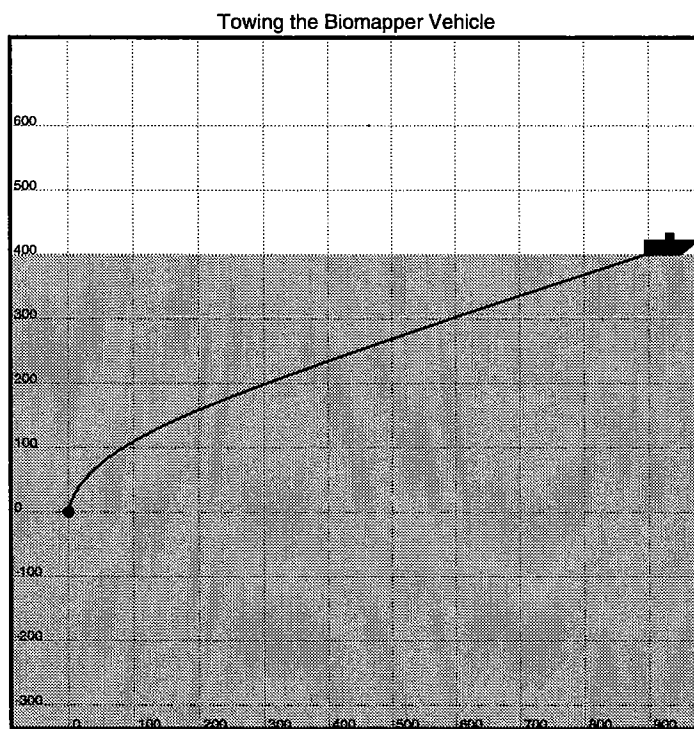


Figure E.1: The static configuration of the system as printed by *animate*.

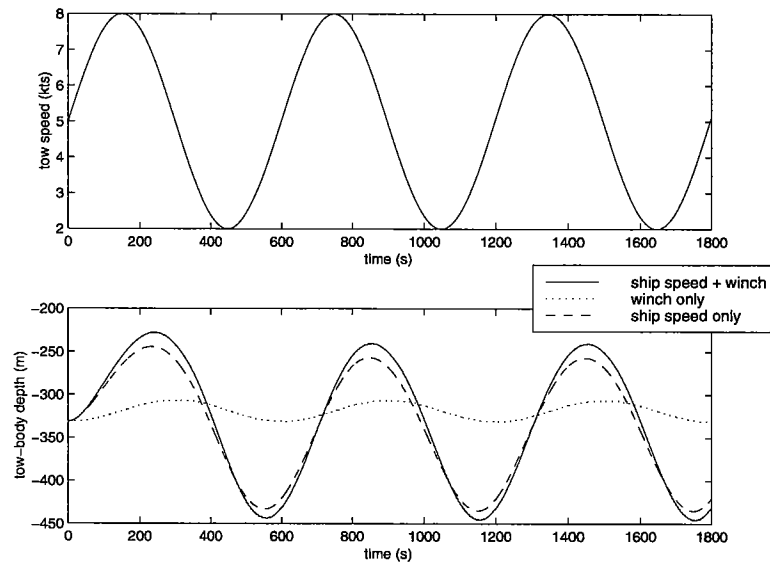


Figure E.2: The speed of the tow ship and the resulting depth profile of the tow sled. This plot was generated using *matlab*.

actively paying-in and paying-out and a run in which the speed does not vary but the winch is paying-in and out.

References

- [1] Odd M. Faltinsen. *Sea Loads on Ships and Offshore Structures*. Cambridge University Press, Cambridge, England, 1990.
- [2] Curtis F. Gerald and Patrick O. Wheatley. *Applied Numerical Analysis*. Addison-Wesley Publishing, Reading, MA, fourth edition edition, 1989.
- [3] Jason I. Gobat. Reducing mechanical and flow-induced noise in the Surface Suspended Acoustic Receiver. Master's thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution Joint Program, Woods Hole, MA, June 1997.
- [4] Christopher T. Howell. *Investigation of the dynamics of low-tension cables*. PhD thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution Joint Program, Woods Hole, MA, June 1992.
- [5] John L. Junkins and James D. Turner. *Optimal spacecraft rotational maneuvers*. Elsevier Science Publishers, Amsterdam, 1986.
- [6] James Lighthill. Fundamentals concerning wave loading on offshore structures. *Journal of Fluid Mechanics*, 173:667–681, 1986.
- [7] J.N. Newman. The motions of a spar buoy in regular waves. Technical Report 1499, David Taylor Model Basin, May 1963.
- [8] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The art of scientific computing*. Cambridge University Press, Cambridge, England, second edition edition, 1989.
- [9] Singiresu S. Rao. *Mechanical vibrations*. Addison-Wesley Publishing, Reading, MA, third edition edition, 1995.
- [10] Andrew H. Sherman. Algorithms for sparse Gaussian elimination with partial pivoting. *ACM Transactions on Mathematical Software*, 4:330–338, 1978.

- [11] William T. Thomson. *Theory of vibrations with applications*. Prentice Hall, Englewood Cliffs, NJ, third edition edition, 1988.
- [12] Athanassios A. Tjavaras. *Dynamics of highly extensible cables*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1996.
- [13] Burton Wendroff. On centered finite difference equations for hyperbolic systems. *Journal of the Society of Industrial and Applied Mathematics*, 8:549–555, 1960.

DOCUMENT LIBRARY

Distribution List for Technical Report Exchange – September 1997

University of California, San Diego
SIO Library 0175C
9500 Gilman Drive
La Jolla, CA 92093-0175

Hancock Library of Biology & Oceanography
Alan Hancock Laboratory
University of Southern California
University Park
Los Angeles, CA 90089-0371

Gifts & Exchanges
Library
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, NS, B2Y 4A2, CANADA

NOAA/EDIS Miami Library Center
4301 Rickenbacker Causeway
Miami, FL 33149

Research Library
U.S. Army Corps of Engineers
Waterways Experiment Station
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Institute of Geophysics
University of Hawaii
Library Room 252
2525 Correa Road
Honolulu, HI 96822

Marine Resources Information Center
Building E38-320
MIT
Cambridge, MA 02139

Library
Lamont-Doherty Geological Observatory
Columbia University
Palisades, NY 10964

Library
Serials Department
Oregon State University
Corvallis, OR 97331

Pell Marine Science Library
University of Rhode Island
Narragansett Bay Campus
Narragansett, RI 02882

Working Collection
Texas A&M University
Dept. of Oceanography
College Station, TX 77843

Fisheries-Oceanography Library
151 Oceanography Teaching Bldg.
University of Washington
Seattle, WA 98195

Library
R.S.M.A.S.
University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

Maury Oceanographic Library
Naval Oceanographic Office
Building 1003 South
1002 Balch Blvd.
Stennis Space Center, MS, 39522-5001

Library
Institute of Ocean Sciences
P.O. Box 6000
Sidney, B.C. V8L 4B2
CANADA

National Oceanographic Library
Southampton Oceanography Centre
European Way
Southampton SO14 3ZH
UK

The Librarian
CSIRO Marine Laboratories
G.P.O. Box 1538
Hobart, Tasmania
AUSTRALIA 7001

Library
Proudman Oceanographic Laboratory
Bidston Observatory
Birkenhead
Merseyside L43 7 RA
UNITED KINGDOM

IFREMER
Centre de Brest
Service Documentation - Publications
BP 70 29280 PLOUZANE
FRANCE

REPORT DOCUMENTATION PAGE	1. REPORT NO. WHOI-97-15	2.	3. Recipient's Accession No.						
4. Title and Subtitle WHOI Cable: Time Domain Numerical Simulation of Moored and Towed Oceanographic Systems			5. Report Date November 1997						
7. Author(s) Jason I. Gobat, Mark A. Grosenbaugh, and Michael S. Triantafyllou			6.						
9. Performing Organization Name and Address Woods Hole Oceanographic Institution Woods Hole, Massachusetts 02543			8. Performing Organization Rept. No. WHOI-97-15						
12. Sponsoring Organization Name and Address Office of Naval Research			10. Project/Task/Work Unit No.						
			11. Contract(C) or Grant(G) No. (C) N00014-92-J-1269 (G) N00014-95-1-0106						
15. Supplementary Notes This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-97-15.			13. Type of Report & Period Covered Technical Report						
			14.						
16. Abstract (Limit: 200 words) This report presents a numerical framework for analyzing the statics and dynamics of cable structures commonly encountered in oceanographic engineering practice. The numerical program, WHOI Cable, features a nonlinear solver that includes the effects of geometric and material nonlinearities, bending stiffness for seamless modeling of slack cables, and a model for the interaction of cable segments with the seafloor. The program solves both surface and subsurface single-point mooring problems, systems with both ends anchored on the bottom, and towing and drifter problems. Forcing includes waves, current, ship speed, and pay-out of cable. The programs that make-up WHOI Cable run under Unix, DOS, and Windows. There is a familiar Windows-style interface available for Windows 95 and Windows NT platforms. In the report, the mathematical and numerical framework for WHOI Cable is described, followed by detailed instructions for formulating problem input files and running the codes. Examples are included in an appendix to highlight the range of problems that WHOI Cable can solve.									
17. Document Analysis <table border="0" style="width: 100%;"> <tr> <td style="width: 20px;">a. Descriptors</td> <td>mooring dynamics cable dynamics towed systems</td> </tr> <tr> <td>b. Identifiers/Open-Ended Terms</td> <td></td> </tr> <tr> <td>c. COSATI Field/Group</td> <td></td> </tr> </table>				a. Descriptors	mooring dynamics cable dynamics towed systems	b. Identifiers/Open-Ended Terms		c. COSATI Field/Group	
a. Descriptors	mooring dynamics cable dynamics towed systems								
b. Identifiers/Open-Ended Terms									
c. COSATI Field/Group									
18. Availability Statement Copyright ©1997 by Woods Hole Oceanographic Institution. All Rights reserved.		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 103						
		20. Security Class (This Page)	22. Price						