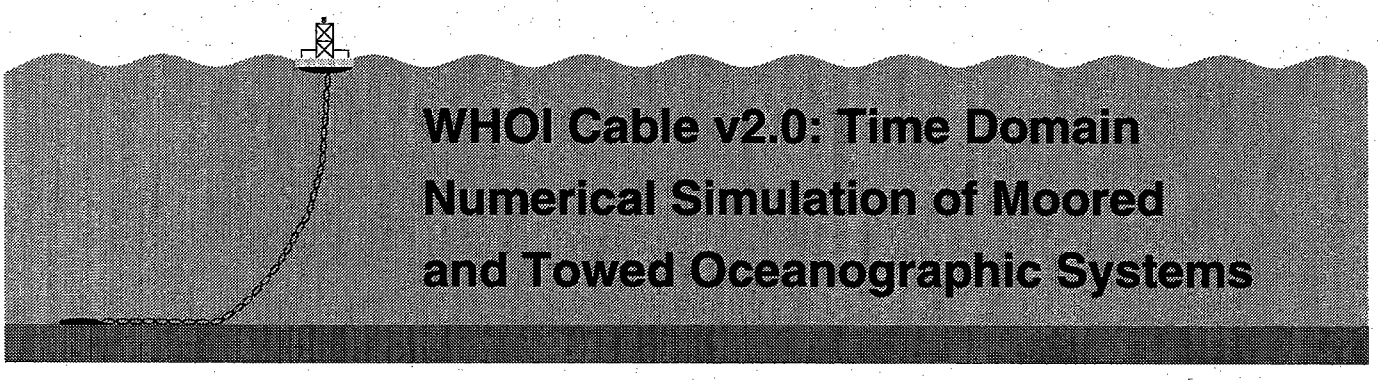
WHOI-2000-08

cd

Woods Hole Oceanographic Institution



WHOI Cable v2.0: Time Domain Numerical Simulation of Moored and Towed Oceanographic Systems

by

Jason I. Gobat
Mark A. Grosenbaugh

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

July 2000

Technical Report

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

WHOI-2000-08

WHOI Cable v2.0: Time Domain Numerical Simulation of Moored and Towed Oceanographic Systems

by

Jason I. Gobat
Mark A. Grosenbaugh

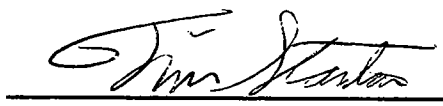
Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

July 2000

Technical Report

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

Approved for Distribution:



Timothy K. Stanton, Chair

Department of Applied Ocean Physics and Engineering

Contents

List of Figures

List of Tables

Foreword

About this Manual

This report documents **version 2.0** of WHOI Cable. While it is our intention to provide up-to-date, comprehensive, accurate documentation, WHOI Cable remains 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 an updated version of WHOI Technical Report 97-15. It is presented largely as a user's guide for WHOI Cable and as such contains a complete description of the WHOI Cable suite of programs as of version 2.0. It does not provide numerical or technical details about the program. Technical information about many aspects of the program is available in [2]. Other sources of information include the previous report [3], and the theses that laid the groundwork for the original implementation of WHOI Cable [4, 8].

Acknowledgements

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

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, such as waves, wind, 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.

WHOI Cable was developed with usability by the operational community in mind. The program can solve a wide range of problems all from within a single consistent interface. The modeled system can consist of any combination of different cable, chain, and rope segments, with instruments, floats, and connectors between segments. The geometry of the system can be multiply-connected (multi-leg moorings and geometries with segments dangling from other segments). The program can also solve towing and drifter problems.

In all cases WHOI Cable can produce solutions in either two or three dimensions and can solve either the static (steady-state) problem given forcing by current, wind, and ship

speed, or the dynamic problem given forcing by waves and time varying wind, current, ship speed, and cable pay-out rates.

1.2 WHOI Cable mathematical features

A detailed derivation of the governing equations for two- and three-dimensional, static, and dynamic problems can be found in [2, 8]. For all problem types the governing differential equations for WHOI Cable include bending stiffness, material nonlinearities, coordinate transformation based on Euler parameters, and a model for cable-bottom interaction. These features provide important new capabilities compared to previous software programs used for modeling oceanographic cable structures.

Incorporating the effects of bending stiffness eliminates the singularity associated with slack tension [4]. Implicit codes without bending stiffness become unstable when tension goes to zero anywhere in the system. For oceanographic applications a very small bending stiffness is usually adequate to overcome this numerical instability. The incorporation of bending stiffness also allows for the seamless modeling of systems with much larger materials, including offshore production risers.

Another instability in many three-dimensional codes arises from the use of three Euler angles to transform between local and global coordinates. Given a specific sequence of rotations, the transformation matrix can become singular in this approach. In long time simulations of geometrically nonlinear systems it is difficult to initialize the transformations such that this singularity will not occur. By using four parameters and treating the transformation as a single rotation about a principal axis, Euler parameter based transformations avoid this problem [5, 8].

The model for the interaction of cable segments with the sea floor is based on a linear elastic foundation [9]. This approach is very general and with appropriate care in selecting parameters, is quite accurate [2]. It treats both bottom stiffness and damping and allows for an arbitrarily varying elevation of the bottom.

1.3 WHOI Cable numerical features

For both static and dynamic problems, the mathematical problem is posed as a system of coupled, nonlinear partial differential equations. The program solves this system numerically by discretizing the continuous (exact) forms of the governing equations using spatial finite differences centered on the half-grid points (which makes the approximation second order accurate [10]) and for dynamic problems, the generalized-α time integration algo-

rithm [1]. The generalized-α algorithm offers superior accuracy and stability compared to alternative algorithms such as the box method, trapezoidal rule, and backward differences [2]. Together with an adaptive time stepping algorithm, the generalized-α algorithm produces very stable dynamic solutions.

At each step of the problem the discretized system of nonlinear equations is solved by an iterative, implicit, adaptive relaxation technique [2, 6]. The initial guess for the solution in static problems is calculated using a shooting method. For the dynamic solution, the initial guess at each time step is the solution from the previous time step. At each iteration the equations are solved using a a sparse Gaussian elimination algorithm [7] for which the computational effort scales linearly with the number of nodes.

1.4 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. For a given model system, a single input file is used to define numerical settings, environmental parameters, system components, and system geometry. Input files are constructed using an intuitive, object based syntax. Definitions for commonly used components can also be contained in central database files. The input syntax allows for the use of symbolic expressions in most assignment statements and for variable expressions in many of the environmental and forcing function definitions. For example, current can be defined as a function of depth using the symbolic variable *H*.

The program does not impose any restrictions on the units used in defining a problem. The ability to use expressions in assignment statements allows for easy conversion from one set of units into the base units chosen for a given problem.

The main post-processing program is *animate*. Solutions for three-dimensional problems can be drawn and animated in perspective view, with controls for on-the-fly perspective rotation and scaling. The program can also generate animated plots of the various independent variables in the solution (velocities, forces, etc.). Plots are available both for the variable as a function of Lagrangian coordinate along the system, updated at each time step, and for the variable at a specific node as a function of time for the entire simulation. Spectra of the temporal plots can be generated with the click of a button. All of the plots and drawings provide controls for unlimited zooming. A mechanism for printing to postscript files is also provided.

For more detailed access to the results, WHOI Cable also includes post-processors to convert results to binary Matlab format or to ASCII text files. The Matlab file contains all of the results generated by the solver, with conveniently assigned variable names. A result

converted to a text file contains only those variables specifically requested by the user.

All of the component programs are written in the C programming language, making them easily portable across platforms. The primary interface under Windows is an encapsulator that provides a graphical interface to all of the component programs. The animation post-processor is written for the X Window environment commonly found on scientific workstations. Under Windows, the encapsulator communicates with the animation application via an X server running locally on the PC. This approach to the development of WHOI Cable allows a single code base to be used for both PC and workstation platforms. The addition of the encapsulator interface under Windows provides a more comfortable environment for PC users who may be unfamiliar with command-line based workstation applications.

Chapter 2

Structure of a *cable* Problem

2.1 Notation and coordinate systems

The basic coordinate system for *cable* is shown in figure 2.1. 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 page[1]. 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.

2.2 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 interpretaion 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.

[1]Note 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.

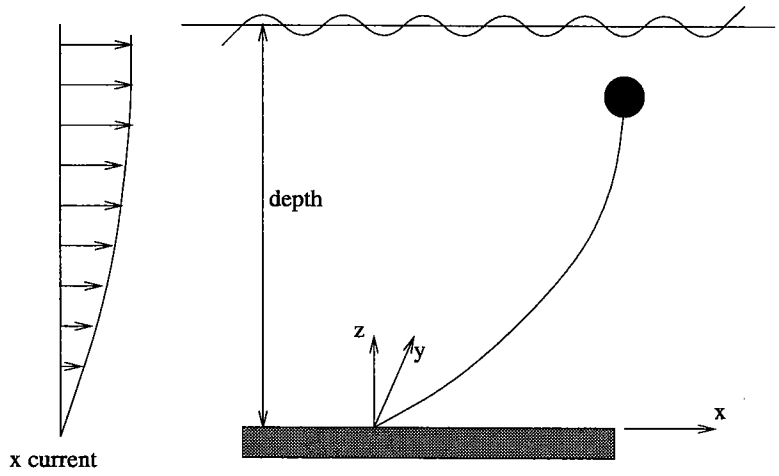


Figure 2.1: Geometric definitions for *cable*.

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.

2.2.1 Expressions

2.2.1.1 Constant expressions

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. 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)). The symbolic constant `pi` can be used in any expression as the value of π (to 20 places).

2.2.1.2 Variable expressions

For parameter specifications that require a functional dependence on a variable, these same constructs can be used with symbolic variables. For example a current definition could be written using the symbolic variable `H` to specify depth. Speed, thrust, and pay-rates at terminal definitions, and wind and current modulation functions in the environment definition can be specified as functions of time, `t`. The bottom elevation in the environment definition can be written as a function of both `x` and `y`. Constitutive functions for nonlinear materials can be written as a function of strain `e`.

For example, a ship speed that varies sinusoidally between -5 and +5 knots with a period of 5 minutes could be written as

```
x-speed = 0.514*5.0*sin(2.0*pi/300*t)
```

A bilinear stress-strain relationship could be written as

```
T = e < 0.01 ? 50000*e : 25000*e
```

2.2.1.3 Discrete functions

Because some functions (particularly currents) 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 function. The basic specification consists of a series of pairs of the form `(x, f(x))` where `f(x)` is the value of the function at independent variable `x`. 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 independent coordinate. You can express a periodic discrete function simply by defining one period and then entering a + symbol at the end of the expression.

2.2.2 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.

2.3 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`). Detailed definitions for all elements of the input syntax are provided below. Chapter 6 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.

2.3.1 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. If the title contains spaces it should be enclosed in double quotation marks.

`type = problem type`

`problem type` can currently be one of `general`, `surface`, `subsurface`, `towing`, `drifter`, `positioned`, `tensioned`, `riser`.

`general` — `general` problems are the simplest problem for *cable* to solve. The second terminal can be either an anchor or buoy (an anchor if the end is fixed in dynamics and a buoy if motions or forces will be applied to the end). The static solution is

based on forces applied at the second terminal which must be specified directly with `x-force=, y-force=`, etc. defined. Thus, while this type is simple for *cable*, it is not very useful for real problems because the static forces are typically not known a priori.

`surface` This is the type to use for single point surface mooring 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. Oftentimes `surface` mooring static solutions are difficult to obtain; automatic dynamic relaxation often works well in such cases (see section 6.2). With some care `surface` problems can also be used to define multileg moorings.

`subsurface` This type is for subsurface single point moorings. The second terminal end must also be a completely defined buoy but *cable* can calculate the forces on that buoy without outer loop iterations. Thus, these problems are typically much easier to solve statically than `surface` problems. However, automatic dynamic relaxation will work for these problems if difficulties do arise.

`towing` `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` `drifter` problems are terminated with buoys at both ends; both buoy definitions must be complete. *cable* will use outer loop iterations to calculate the steady state drift speed of the system.

`positioned` These problems were called `horizontal` in earlier versions of WHOI Cable. In `positioned` problems, the second terminal end must be positioned away from the first terminal with `x=, y=, z=` definitions. *cable* will perform outer loop iterations to calculate the appropriate reaction force at the second terminal that brings the terminal to the required po-

sition. The terminal can contain either an anchor or a buoy, depending on the whether motions are to be applied in the dynamic problem. This is often the best choice of problem type for multileg moorings (particularly if there is no surface expression).

`riser` — `riser` problems are nearly identical to `positioned` problems except for the treatment of material above the free surface. In `positioned` problems buoyant material above the surface is made to float on the surface. In `riser` problems, the weight of material above the surface is taken to be the air weight of the material.

`tensioned` — This problem type is useful for systems with a specified pretension applied. Both `tension=` and `z=` (vertical position) of the second terminal must be specified. *cable* will use outer loop iterations to calculate the angle at the top of the mooring that brings the top node to the specified vertical position given the applied tension.

`deployment` — This type is used to simulate anchor last deployment of single point moorings. The first terminal should have a fully defined anchor and a speed specification. The second terminal should have a fully defined buoy specified. The static solution will be calculated by assuming that the surface buoy is being towed (at the speed given in the first terminal definition) with all of the mooring paid out and only the anchor left on deck. This is essentially a surface mooring solution where the tow speed can be represented as a current flowing in the opposite direction and the anchor is on the surface rather than the bottom. At the beginning of the dynamic solution the anchor is released from surface and the whole system begins to fall towards the bottom. Once the anchor hits the bottom it is fixed in place. Static solutions for these problems can be difficult to obtain. Brute force with small static relaxation factor and near unity outer relaxation typically works best if problems are encountered.

2.3.2 Analysis parameters

The `analysis parameters` section contains definitions that control the numerical algorithms which are used in the solution of *cable* problems. The solution of a *cable* problem can be broken down into two main phases: static and dynamic solutions. The dynamic solution is actually quite straightforward from a user perspective. The static solution can be more complicated.

Within the static solution process there is an initial guess solution, static iterations, and, in many problems, static outer iterations. The first step in a static solution is always an initial guess based on either a catenary or a shooting solution. Shooting solutions require iterations and can fail to converge, but are quite accurate when they can be obtained. Catenary solutions are nearly foolproof, but they are seldom a good guess and thus leave much work for the following stages. After the initial guess is calculated the program uses an iterative relaxation solver to calculate a static solution given a complete set of prescribed boundary conditions (forces at buoys, anchor positions, etc.) Static outer iterations are needed when the some of the boundary conditions are not known a priori. For example, the forces on a buoy are dependent on the draft of the buoy, but the draft cannot be calculated until all the forces are known. Thus, the boundary conditions must be solved for in this outer loop of iterations. Once a final static solution is calculated, that solution is used as the initial condition for the dynamic solution. The dynamic solution uses a single set of iterations to find the instantaneous equilibrium solution at each time-step, with the solution from the previous time-step as the initial guess.

2.3.2.1 Global options

`relaxation = constant expression`

The primary nonlinear solution technique within *cable* is known as relaxation. Using an initial guess at a solution vector, the algorithm calculates an update to the solution that will bring the system closer to equilibrium. The updated solution is then fed back into the algorithm and the process repeats until some suitably close approximation of equilibrium is achieved. The relaxation factor is used to speed or slow this iterative process. In highly nonlinear problems the update to the solution may not be very accurate and thus should not be fully applied. The relaxation factor allows this partial application of the update. *cable* does have an adaptive relaxation algorithm that will adjust the relaxation factor in troublesome portions of the solution (see section 2.3.2.4 below).

`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.

`max-iterations = integer`

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

2.3.2.2 Static solution options

`static-initial-guess = string`

Determines the algorithm used to form the initial guess for the static solution algorithm. Must be one of `shooting` or `catenary`. Defaults to `catenary` if it is not explicitly given. `shooting` is often a better choice but shooting solutions are not available for all problem types.

`static-solution = string`

Determines the algorithm used to calculate the final static solution to be used as the initial condition in the dynamic solver. Must be one of `relaxation`, `shooting`, or `catenary`. When dynamic solutions are desired it should almost always be `relaxation` and this is the default. The other options are provided primarily for debugging purposes. Also, because shooting solutions are often very fast and reasonably accurate, they can be useful in preliminary design studies or when dynamic solutions are not desired.

`static-relaxation = constant expression`

The relaxation factor to be used in the static solution. In many problems, it may be necessary to have different factors for static and dynamic solutions. Dynamic solutions almost always proceed best with a relaxation factor of 1.0 (full update applied at each iteration), but many static problems (particularly those with complex geometry or cable lying on the bottom) are only stable with a relaxation factor of 0.1 or smaller. If not given the static relaxation factor will default to the value given by `relaxation=`. 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. For many static problems a tolerance of 0.01 - 0.0001 is adequate. In some rare cases a static tolerance that is too large leads to poor resolution of boundary conditions and static outer iterations can fail to converge.

`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.

`static-outer-relaxation = constant expression`

The "relaxation" or "stiffness" factor to be used in static outer iterations. This is not actually a relaxation factor in the same sense as described above. It is really a scaling factor that has different uses depending on what type of problem is being solved. In `surface` problems it is the factor by which the trial draft of the buoy is multiplied at each outer iteration when the algorithm is trying to bracket the solution between the maximum draft and a minimum draft at which a solution can be obtained. It defaults to 0.95 in these problems; smaller values can speed outer loop convergence, but can also lead to solution instabilities. For `positioned` problems it is the factor by which distance errors are multiplied to generate an update to the applied force vector at a positioned boundary. It defaults to 5.0 in these problems; larger values can lead to faster convergence, but can also can cause solution instabilities.

`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 (in a `surface` problem) or position of the second terminal (in a `positioned` problem) 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. A value of 0.01 is typically sufficient for an accurate solution.

`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 `positioned` 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.

`shooting-iterations = integer`

This is the maximum number of allowed iterations in shooting solution initial guesses. If not given it will default to the maximum number of allowed static iterations. Shooting solutions can require outer iterations similar to those for regular static solutions. The limit determined by `static-outer-iterations` is used for those iterations.

2.3.2.3 Dynamic solution options

`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 occuring. 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 maxium of 5 times. If after the fifth nested reduction a singularity is encountered the program will halt. Typical base time steps range from 0.01 to 0.1 seconds for mooring problems to 0.5 to 1.0 seconds for towing problems.

`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. This tolerance should generally be low ($< 10^{-6}$) to prevent errors from building up as the solution progresses in time.

`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. The best choice for this parameter is almost always 1.0.

`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. A value of 20 is appropriate for most problems.

`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. It is generally advisable to ramp the excitation over one or two excitation or wave periods.

2.3.2.4 Advanced options

`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. Use with caution though, as other problems may converge most quickly at high current.

`relax-adapt-up = constant expression`

This is the amount by which the current relaxation factor will be multiplied when the iterative solution is progressing well. It will not grow beyond the base value. To turn off adaptive relaxation set this value and `relax-adapt-down` to 1.0. `relax-adapt-up` defaults to 1.02.

`relax-adapt-down = constant expression`

This is the amount by which the current relaxation factor will be divided when the iterative solution is not progressing well. The relaxation factor can continue to shrink until it becomes so small that the solution stalls (see `relax-stall-limit` below). When the solution starts to make progress again `relax-adapt-up` will be applied and the relaxation factor will slowly increase back to its baseline value or to an equilibrium value at which the solution can make the best overall progress. To turn off adaptive relaxation

set this value and `relax-adapt-up` to 1.0. `relax-adapt-down` defaults to 1.1.

`relax-stall-limit = integer`

This is the maximum number of iterations allowed with a non-progressing solution before the relaxation factor will revert back to its baseline value.

`mesh-smoothing-length = constant expression`

This is the length over which the curvature will be averaged to compute the the mesh distribution weighting function.

`mesh-amplification = constant expression`

The adaptive mesh distribution algorithm tries both to increase node density in areas of high curvature and to keep nodes from being placed too far apart. In that algorithm, this is the weight given to curvature effects (relative to unity for spacing effects). A value of 20 appears to reasonable for chain catenary moorings.

`dynamic-alpha-k = constant expression`

This is the weighting factor for stiffness and force terms in the generalized-α time integration algorithm.

`dynamic-alpha-m = constant expression`

This is the weighting factor for mass terms in the generalized-α time integration algorithm.

`dynamic-gamma = constant expression`

This is the generalized trapezoidal rule integration weighting factor in the generalized-α time integration algorithm. The time integration will be second order accurate so long as

$$\gamma + \alpha_m - \alpha_k = \tfrac{1}{2}. \tag{2.1}$$

`dynamic-lambda = constant expression`

This is the value of $\lambda_{1,2}^{\infty}$ in the generalized-α time integration algorithm. If specified, values for α_k, α_m, and γ will be calculated automatically from

$$\alpha_k = \frac{\lambda^{\infty}}{\lambda^{\infty} - 1}, \qquad \alpha_m = \frac{3\lambda^{\infty} + 1}{2\lambda^{\infty} - 2}, \tag{2.2}$$

and the requirement that the integration be second order accurate (equation 2.1). Valid values are $-1 \leq \lambda_{1,2}^{\infty} < 1$. The box method (the closest thing to the algorithm in previous versions of *cable*) is defined by $\lambda_{1,2}^{\infty} = -1$.

Useful values for the generalized-α algorithm are 0.5 to -0.8. It defaults to -0.5.

`dynamic-rho = constant expression`
Same as `dynamic-lambda`.

2.3.3 Environmental parameters

The `environment` section is used to define the external conditions under 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`, `velocity`, and `force`.

`wave-follower`
With wave following surface buoys the heave motion of the buoy is governed by the instantaneous vertical displacement of the wave field defined by `x-wave=` and `y-wave=` or `wave-file=`. The buoy is unconstrained in the horizontal directions and is free to respond to time-varying forcing by current and wind.

Horizontal motions are not forced by wave inputs.

`velocity` For `velocity` forcing the motion at the top of the system is completely described by `x-input=`, `y-input=`, and `z-input=` or `velocity-file=`.

`force` Like `velocity` but the inputs are taken to be forces rather than motions.

`morison` `morison` forcing calculates drag and inertial forces on subsurface bodies based on wave particle velocity and acceleration. This is the most physically realistic choice for subsurface moorings.

`x-wave = (constant expression, constant expression, constant expression)`
The amplitude, period, and relative phase of the surface wave traveling in the global x direction. Used with `wave-follower` and `morison` forcing.

`y-wave = (constant expression, constant expression, constant expression)`
The amplitude, period, and relative phase of the surface wave traveling in the global y direction. Used with `wave-follower` and `morison` forcing.

`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` or `force`.

`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` or `force`.

`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` or `force`.

`x-current = variable expression`
The current in the global x-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`).

`x-current-modulation = variable expression`

The time varying modulation of the current in the x-direction, either as a discrete expression or a continuous expression with the symbolic variable `t`. Defaults to 1.0 (no modulation).

`y-current-modulation = variable expression`

The time varying modulation of the current in the y-direction, either as a discrete expression or a continuous expression with the symbolic variable `t`. Defaults to 1.0 (no modulation).

`x-wind = variable expression`

Wind, possibly as a function of time, in the x-direction.

`y-wind = variable expression`

Wind, possibly as a function of time, in the y-direction.

`bottom-elevation = variable expression`

The elevation of the bottom relative to the origin (the anchor in most problems). The elevation can be specified as a continuous function of both `x` and `y`. It should always be defined such that the origin is at elevation zero. Discrete expression syntax can only be used to describe elevations as a function of `x` only.

`bottom-friction = constant expression`

The coefficient of static friction between mooring line and sea floor.

`bottom-stiffness = constant expression`

The spring stiffness of the bottom per unit length. For oceanographic applications values from 100 to 1000 (in MKS units) are typical.

`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. A value greater than 1.0 is seldom necessary.

`wave-file = string`

The name of the data file containing a spectral description of the input wave field in the x-direction. The file should be ASCII data with three columns per line: radial frequency, spectral power, and phase.

`velocity-file = string`

The name of the data file containing time series of input velocity. The file should be ASCII data with four columns per line: time, x velocity, y velocity, and z velocity. This is typically most useful for model validation and calibration when trying to match experiment results given an observed time series of input motions.

`force-file = string`

Same as `velocity-file` when `forcing-method=force`.

2.3.4 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.

`m = constant expression`

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

`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. For neutrally buoyant materials specify some very small number.

`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 and 0.5 to 0.6 for chain.

`Cdt = constant expression`

The drag coefficient in the tangential (longitudinal) direction. Typical values for oceanographic cables range between 0.003 (for smooth cables) and 0.05 (for some faired cables). A typical value for chain is 0.01.

`EA = constant expression`

The axial stiffness of the material. Must be non-zero. For ropes and cables this value should typically be 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 the theoretical value for a solid rod given by

$$E\frac{\pi R^4}{4}. \tag{2.3}$$

`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

$$G\frac{\pi R^4}{2}. \tag{2.4}$$

`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.

`amt = constant expression`

The tangential added mass per unit length of the material. If not given it will be taken to be 0.0. Generally only applies for chain and some instruments that are modeled as material segments.

`amn = constant expression`

Same as `am`.

`Cat = constant expression`

The tangential added mass coefficient. If `amt` is not given and `Cat` is specified the tangential added mass will be calculated as $A\rho C_{a_t}$.

`Can = constant expression`

The normal added mass coefficient. If `amn` is not given and `Can` is specified the normal added mass will be calculated as $A\rho C_{a_n}$.

`Cmt = constant expression`

The tangential virtual mass coefficient. If `amt` and `Cat` are not given and `Cmt` is specified the tangential added mass will be calculated as $A\rho(C_{m_t} - 1)$.

`Cmn = constant expression`

The normal virtual mass coefficient. If `amn` and `Can` are not given and `Cmt` is specified the normal added mass will be calculated as $A\rho(C_{m_n} - 1)$.

`bt = constant expression`

Structural damping constant of the material for tangential motions. Negligible in most applications.

`bn = constant expression`

Structural damping constant of the material for transverse motions. Negligible in most applications.

`type = string`

Specifies the type of constitutive relationship to use for this material. Valid values are `linear` and `nonlinear`. If the material is linear then `EA` will be used for all tension-strain calculations. If `nonlinear` is specified `T`, `Te`, and `Tee` must be given. Default is `linear`.

`T = variable expression`

For nonlinear materials this function defines the load-elongation curve as a function of strain using the symbolic variable `e`. If this function is given both `Te` and `Tee` must be given as well. In such cases these functions will be used rather than `EA` to calculate the material's tension response.

`Te = variable expression`

For nonlinear materials the first derivative of the load-elongation curve as a function of strain, `e`. For linear materials the slope of the load-elongation

curve is EA.

`Tee = variable expression`
: For nonlinear materials the second derivative of the load-elongation curve as a function of strain, `e`. For linear materials the second derivative of the load-elongation curve is zero.

`comment = string`
: Defines a string that can be used to identify the material definition more fully than its symbolic name.

2.3.5 Connectors

The `connectors` section is used to define the shackles, floats, 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.

`d = constant expression`
: The characteristic diameter used to calculate a drag area.

`am = constant expression`
: The added mass of the connector. If it is not specified it will be calculated based on the specified characteristic diameter.

`Cdt = constant expression`
: The tangential drag coefficient of the connector. In global coordinates this is the drag coefficient used for vertical motions.

`Cdn = constant expression`
: The normal drag coefficient of the connector. In global coordinates this is the drag coefficient used for horizontal motions.

`comment = string`
: Defines a string that can be used to identify the connector definition more fully than its symbolic name.

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.

2.3.6 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` 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.

`type = string`
: The basic buoy type. Currently recognized values are `sphere`, `cylinder`, `capsule`, `axisymmetric`, `ship`, `platform`.

`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 = (`x_1`, d1) ... (`x_n`, dn)`
: 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 fully submerged displacement of the buoy. If not given it will be computed based on buoy type and specified geometry. This is not the same as the available buoyancy or wet weight.

`m = constant expression`
: The mass of the buoy.

`am = constant expression`

The added mass of the buoy. Currently only used when `morison` forcing is active or for the motions of tow bodies. If it is not specified it will be calculated based on buoy diameter.

`Cdt = constant expression`

The tangential drag coefficient of the buoy. Also used for vertical forces in global coordinates.

`Cdn = constant expression`

The normal drag coefficient of the buoy. Also used for horizontal forces in global coordinates.

`Cdw = constant expression`

The drag coefficient of the buoy in wind.

`Sw = constant expression`

The surface area of the buoy exposed to wind drag. If not given it will be calculated from the buoy draft and shape, but this will only be accurate for a small class of systems.

`comment = string`

Defines a string that can be used to identify the buoy definition more fully than its symbolic name.

2.3.7 Anchors

The `anchors` section defines the anchors that are used at one or both ends of the system. The parameters are currently only used in `deployment` problems. For other problems you must create a valid anchor name to be used in the `terminal` definitions of the `layout` section for any problems that require it.

`m = constant expression`

The mass of the anchor.

`wet = constant expression`

The wet weight of the anchor.

`d = constant expression`

The characteristic diameter of the anchor. The forces on the anchor are calculated assuming that the anchor is a sphere with this diameter.

`Cdn = constant expression`

The normal drag coefficient of the anchor. Also used for forces in the global horizontal directions.

`Cdt = constant expression`

The tangential drag coefficient of the anchor. Also used for forces in the global vertical direction.

`mu = constant expression`

The coefficient of static friction between the anchor and the bottom. In future versions of *cable* this may be used in calculating the holding power of the anchor. It is currently unused in all problem types.

2.3.8 System layout

2.3.8.1 Single point, simply connected systems

The geometry of the most model systems 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 in the above described sections of 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)
    }
```

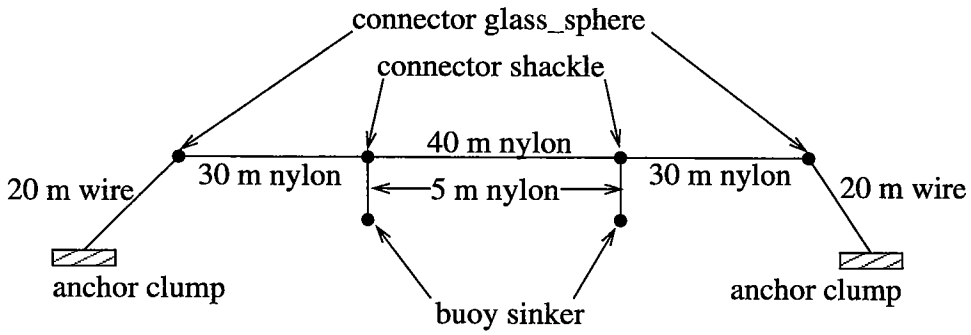


Figure 2.2: Geometry of the branched layout described in the text.

```
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.

2.3.8.2 Branched and multileg systems

Any system that cannot be described as a single, simply connected string of segments between two terminals must be described using branches. Branches can be used to describe multileg systems and system that have strings of segments hanging from other segments. Branches were originally added to *cable* to handle this latter case for horizontal array moorings. The geometry shown in figure 2.2 is described by a modified version of the layout with both ends anchored described above:

```
Layout
    terminal = {
       anchor = clump
    }
    segment = {
       length     = 20
       material   = wire
       nodes      = (40, 1.0)
    }
```

```
connector = glass_sphere
segment = {
   length     = 30
   material   = nylon
   nodes      = (30, 1.0)
}
connector = shackle
branch = {
   segment = {
      length = 5
      material = nylon
      nodes = (5, 1.0)
   }
   terminal = {
      buoy = sinker
   }
}
segment = {
   length = 40
   material = nylon
   nodes = (40, 1.0)
}
connector = shackle
branch = {
   segment = {
      length = 5
      material = nylon
      nodes = (5, 1.0)
   }
   terminal = {
      buoy = sinker
   }
}
segment = {
   length     = 30
   material   = nylon
   nodes      = (30, 1.0)
}
connector = glass_sphere
```

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

The strings hanging off the horizontal member are defined by adding a `branch=` specification after a `connector=`. For multileg moorings multiple branches can leave from the same connector by adding more `branch=` specifications. Within a branch definition segments and connectors are strung together to define the branch just as they are on the main system. A branch definition must end with a terminal. The terminal can contain an anchor or a buoy, or it can rejoin the main system to form a loop using `node=`.

2.3.8.3 Layout parameters

`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 specifi-

cation 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 :` $(n_1^1, n_2^1, \ldots)$, `string :` $[n_{start}, n_{step}, n_{stop}], \ldots$
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. For irregularly spaced attachments the list of nodes can be given explicitly inside parentheses. For regularly spaced attachments you can use the square bracket construct to define the starting node, interval, and ending node for the placement of an attachment.

`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. A connector must precede a branch (i.e., moments must be released at a point where more than two segments come together).

`branch = { branch definition }`
Specifies a branched series of segments and connectors that leaves from the main system. Each branch definition must end with a terminal.

2.3.8.4 Terminal parameters

`terminal = { terminal definition }`
A terminal definition must come both at the beginning and end of the list of segments and connectors and at the end of every branch. 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=`, `x-thrust=`, `y-thrust=`, `z-thrust=`, `tension=`, `pay-rate=` and `release-time=`. Main terminal definitions must contain at least an `anchor` or `buoy` definition. Branch terminals must have

an anchor, buoy or node= statement.

`anchor = string`

Specifies the anchor 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).

`z = constant expression`

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

`x = constant expression`

The x location (2D in-plane horizontal) of an anchor in the global coordinate space.

`y = constant expression`

The y location (3D out-of-plane horizontal) 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 one another could be used to specify circular or elliptical tow patterns). In order to get a valid static (steady-state) solution, at least one component 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.

`x-thrust = variable expression`

This the time varying force applied on a buoy terminal in the global x direction. It is intended to be used with ROV towing problems.

`y-thrust = variable expression`

Time varying thrust in the global y direction.

`z-thrust = variable expression`

Time varying thrust in the global x 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. 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.

`node = integer`

In a branch terminal definition this option is used in lieu of an anchor or buoy to specify that the branch rejoins the main segment string to form a loop. The node number must be a valid global node number on the main system.

`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.

2.3.9 The end statement

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

Chapter 3

The *cable* Application

3.1 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
```

[1]There 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
```

3.2 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 3.1. 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. The status message window provides a 100 line circular buffer for algorithmic and solution progress information. You can use the arrow buttons next to the message window to scroll forward and backward through this buffer. Note that with the information and control dialog enabled, *cable* does not automatically exit after the solution is complete. The dialog will remain on the screen until the **quit** button is pressed unless the `-quit` command-line option was specified.

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.

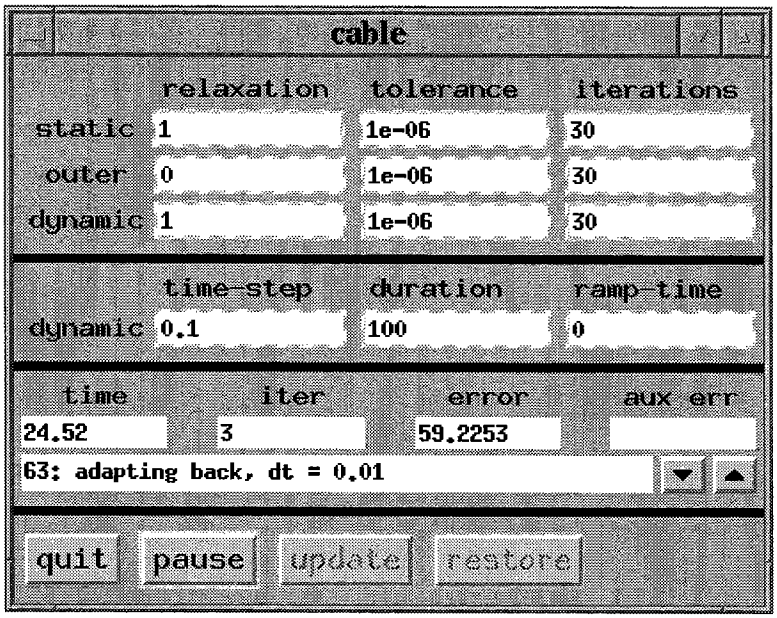


Figure 3.1: *cable*'s graphical information and control dialog.

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 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.

In a dynamic problem, the time-step can also be changed while the solution is in progress. Care must be taken so that the solution remains on the sampling grid. Also, changes to the time-step do not take effect instantaneously. If the adaptive time-stepping algorithm is active then the changed time-step will not take effect until the solution is back on the original grid. When the solution is on the regular temporal grid, the updated time-step takes effect one step after the change is made. So if the current time is 3.55 and the time-step is changed from 0.01 to 0.05, the next solution will be at 3.56, not 3.60 as was probably intended. You can pause and unpause the solution as needed to make sure that time-step changes are made in the step prior to a step on the new solution grid.

3.3 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 2) 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.

3.4 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.

`-initial filename`
: if given, indicates that the static solution should use the solution from `filename` as the initial guess, foregoing catenary or shooting solutions. The output filename and the initial filename cannot be the same. Like the load filename the solution contained in `filename` must contain a complete static solution for the exact problem geometry defined by the current input file.

`-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.

`-quit`
: forces the program to exit without waiting for the **quit** button after an error or once the solution is complete. This is only meaningful when the graphic information and control dialog is enabled.

`-auto`
: overrides the analysis parameters in the input file and forces an auto-mode static solution. Only useful for `surface` or `subsurface` problems.

`-dynstat filename`

causes the final solution in a dynamic problem to be written as a static solution in the separate output file given by `filename`. This is useful for calculating static solutions using dynamic relaxation.

`-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).

`-terminals` boolean option that adds all terminal nodes to the list of output nodes at which temporal results will be written to the output file. This can be useful for branch problems.

`-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.

`-decimate x` specifies the increment to use when writing spatial (static and snapshot) results. The default is 1 (every node will be written). The decimation factor must be set such that the node 1 and node n are included in the output. Decimation will produce possibly strange results in branched problems. Static solutions cannot be loaded from a file containing a decimated result.

`-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 4).

`-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.

3.5 Interpreting the output from *cable*

The output files that come out of *cable* are written in a custom binary format outlined in figure 3.5. 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 4).

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

problem type indicator
8 bits used to indicate the way that the results should be interpreted. Least significant bit is always 1. Second bit indicates that 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. Sixth bit indicates a multi-leg or branched problem. The last two 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.

{branch information}
Optional - only present if the branch info bit is set in the problem type indicator byte. The first four bytes are the number of branches as an integer. The starting node of each branch is then given as a four byte integer. Results are written sequentially by branch, with the mainline first. So if branch 1 starts at 101 and branch 2 starts at 201, then results for branch 1 are stored in array indices (unit offset) 101 through 200.

$s(1)\ \{x(1)\ y(1)\ z(1)\}\ \{T(1)\ S_n(1)\ S_b(1)\}\ \{M_t(1)\ M_n(1)\ M_b(1)\}\ \{B_0(1)\ B_1(1)\ B_2(1)\ B_3(1)\}\ \ldots$

$s(n)\ \{x(n)\ y(n)\ z(n)\}\ \{T(n)\ S_n(n)\ S_b(n)\}\ \{M_t(n)\ M_n(n)\ M_b(n)\}\ \{B_0(n)\ B_1(n)\ B_2(n)\ B_3(n)\}$
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

$'t'\ \{x(1)\ y(1)\ z(1)\}\ \{u(1)\ v(1)\ w(1)\}\ \{T(1)\ S_n(1)\ S_b(1)\}\ \{M_t(1)\ M_n(1)\ M_b(1)\}\ \{B_0(1)\ B_1(1)\ B_2(1)\ B_3(1)\}\ \ldots$

$\{x(n_o)\ y(n_o)\ z(n_o)\}\ \{u(n_o)\ v(n_o)\ w(n_o)\}\ \{T(n_o)\ S_n(n_o)\ S_b(n_o)\}\ \{M_t(n_o)\ M_n(n_o)\ M_b(n_o)\}\ \{B_0(n_o)\ B_1(n_o)\ B_2(n_o)\ B_3(n_o)\}$

$'s'\ \{x(1)\ y(1)\ z(1)\}\ \{u(1)\ v(1)\ w(1)\}\ \{T(1)\ S_n(1)\ S_b(1)\}\ \{M_t(1)\ M_n(1)\ M_b(1)\}\ \{B_0(1)\ B_1(1)\ B_2(1)\ B_3(1)\}\ \ldots$

$\{x(n)\ y(n)\ z(n)\}\ \{u(n)\ v(n)\ w(n)\}\ \{T(n)\ S_n(n)\ S_b(n)\}\ \{M_t(n)\ M_n(n)\ M_b(n)\}\ \{B_0(n)\ B_1(n)\ B_2(n)\ B_3(n)\}$
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 3.2: The binary file format for *cable* results files.

Chapter 4

Post-processing *cable* Results

4.1 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.

4.1.1 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 (`x`, `T`, `Mn`, etc.). Node–time histories are written to variables with names subscripted by `t` (`x_t`, `T_t`, `Mn_t`, etc.). Snapshots are given names subscripted with `s` (`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 4.1. Also included in the Matlab file are variables with the sample rate (`dt`), snapshot rate (`snap_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.

	2-D Results		3-D Results	
information	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 4.1: The names that *res2mat* assigns to Matlab variables. These same names are used with the `-variables` option in *res2asc* to specify which variables to tabulate.

4.1.2 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)`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 total horizontal position of one of our output nodes we would need to do the following

```
>> plot(t, x_t(:, 3) + x(nodes(3)));
```

Unless otherwise requested with the `-totals` command-line switch, variables are written to the Matlab file in dynamic form (total - static value). 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.

4.1.3 *res2mat* command line parameters

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

`-in results file ...`
: the name(s) of the file containing the *cable* results. Multiple input filenames are allowed. If multiple inputs are given no output names can be explicitly specified. They will be automatically constructed.

`-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. If no output names are provided, names will be constructed from the input names.

`-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.

`-totals`
: Boolean option to write dynamic results as totals (static + dynamic).

`-ft`
: Boolean option to convert length units from meters to feet. Only useful if the results in the input file are stored in meters.

`-lbs`
: Boolean option to convert force units from Newtons to pounds. Only useful if the results in the input file are stored in Newtons.

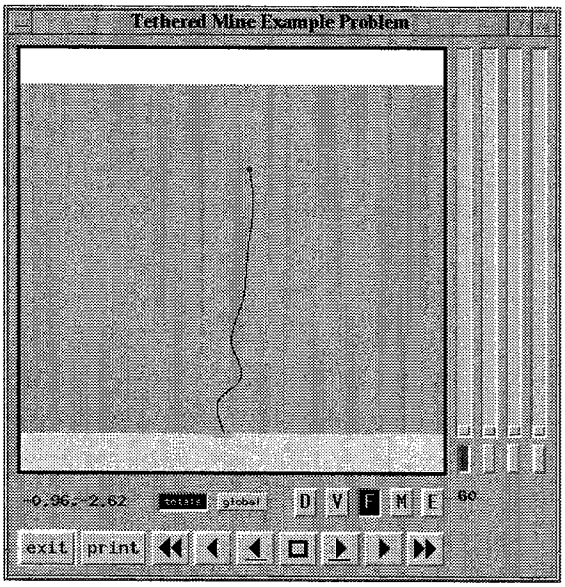


Figure 4.1: The main window of *animate*.

4.2 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.

4.2.1 The main animation window

On start-up the *animate* main window (figure 4.1) 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 button[1]. Each marker is identified with a unique color – this is the color with

[1]Most 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

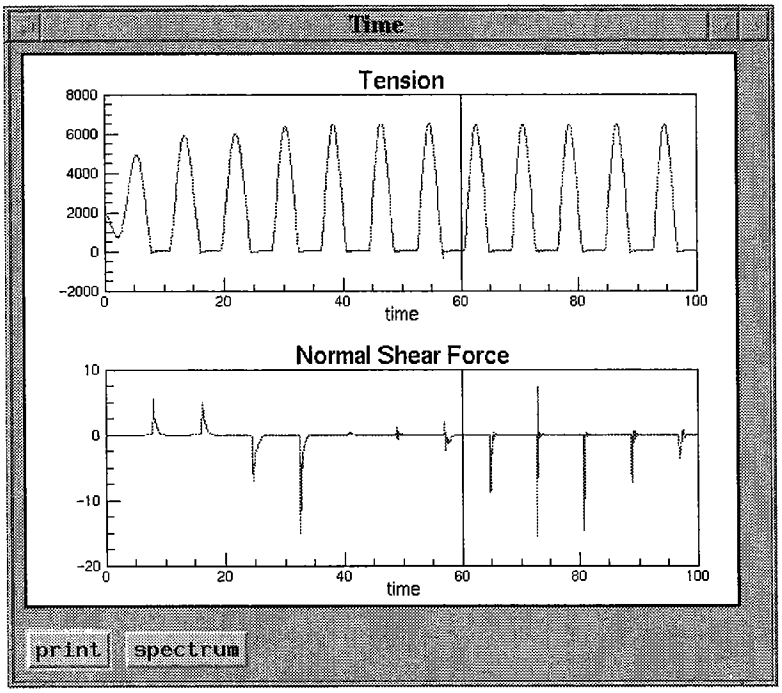


Figure 4.2: A time plot of the forces at marked nodes.

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 will also be generated. A temporal plot of the forces for the animation shown in figure 4.1 is shown in figure 4.2. There is only one curve on each graph because we currently have only one marked node. The vertical black bars on each graph indicate the current time point. The results plotted here are the total force (static + dynamic value). Transformation between total and dynamic only and local and global coordinates can be made by clicking the appropriate button next to the plot controls. Spatial distribution plots are updated at the same rate as the main animation. Plots can be popped down simply by disengaging the appropriate lettered button in the main window.

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 at the bottom of the main window (figure 4.1). 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.

4.2.2 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 [r] with the focus on the graph area and the mouse point on the appropriate graph (you may need to use [tab] to get the focus on the graph area).

4.2.3 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. This option can also be toggled while the program is running by clicking the *globals* button in the main window.

`-totals`
: boolean option to plot quantities as static + dynamic result. In normal operation only the dynamic portion is plotted for node–time histories. This option can also be toggled while the program is running by clicking the *totals* button in the main window.

`-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.

`-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^o.

`-yrot y` the initial y-axis rotation for 3D perspective view. The default is 40^o.

`-zrot z` the initial z-axis rotation for 3D perspective view. The default is 0^o.

`-zscale s` the initial z-axis scaling ("eye distance") for 3D perspective view. The default is 0.4.

4.3 ASCII output

The post-processing application *res2asc* can be used to convert binary *cable* results to tabular ASCII data. Each of the results matrices (static, node-time histories, and spatial snapshots) are saved to separate files. The variables that are output can be controlled using the `-variables` command-line option. The list of variables available for output depends on whether the solution in the binary results file is from the 2D or 3D algorithm and whether the `-global` command-line flag is given. The variable names for use with the `-variables` option are the same as those listed in table 4.1. The variable `s` is also available for output with static and snapshot results. The time is always output as the first column in node-time history results.

4.3.1 *res2asc* command line parameters

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

`-in filename`
: the name of the file containing the *cable* results.

`-static filename`

the name to use in creating the ASCII output file for static results.

`-time filename`

the name to use in creating the ASCII output file for node-time histories.

`-snap filename`

the name to use in creating the ASCII output file for time history snapshot results.

`-variables v1 v2 v3 ...`

list of variable names that you want output into the table.

`-global` Boolean option to write 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 write dynamic results as totals (static + dynamic).

`-ft` Boolean option to convert length units from meters to feet. Only useful if the results in the input file are stored in meters.

`-lbs` Boolean option to convert force units from Newtons to pounds. Only useful if the results in the input file are stored in Newtons.

Chapter 5

cable's Windows Interface

5.1 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*, *res2mat*, and *res2asc* to solve the problem and post-process the results, all from within a single Windows 95/98/2000/NT based application. Figure 5.1 illustrates this interaction between the various component programs.

The main editor window, with one of the example problems loaded, is shown in figure 5.2. 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 **File** and **Insert** menus and for invoking the various component programs.

5.2 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 or database objects that the encapsulator provides.

You can insert objects from predefined databases by choosing **Object browser** from the **Insert** menu or by choosing the toolbar button that has all four object types on it. The object browser, pictured in figure 5.3 allows you to browse through all of the materials,

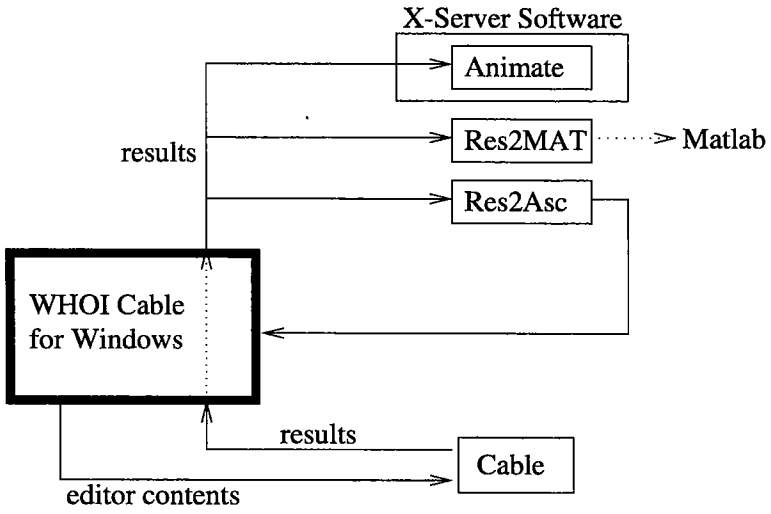


Figure 5.1: The relationships between the WHOI Cable component programs.

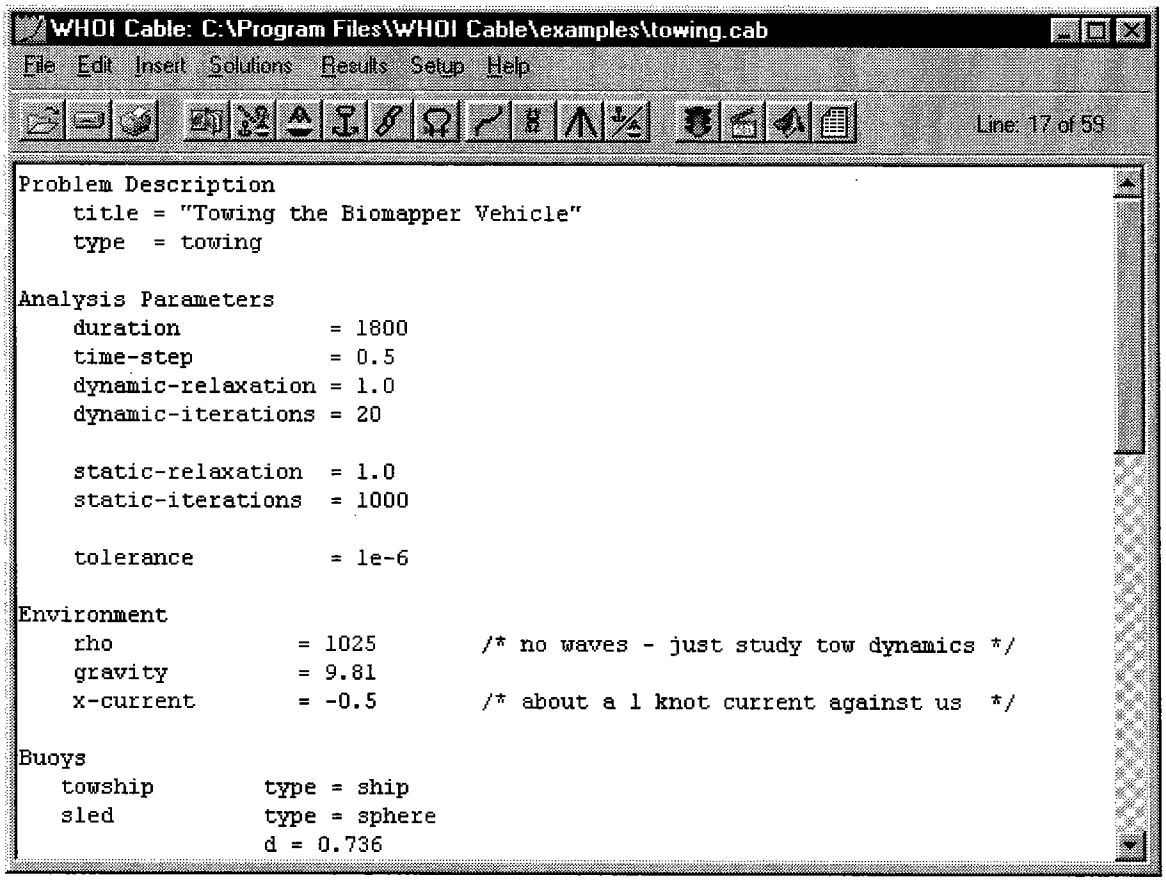


Figure 5.2: The main window of the WHOI Cable Windows interface.

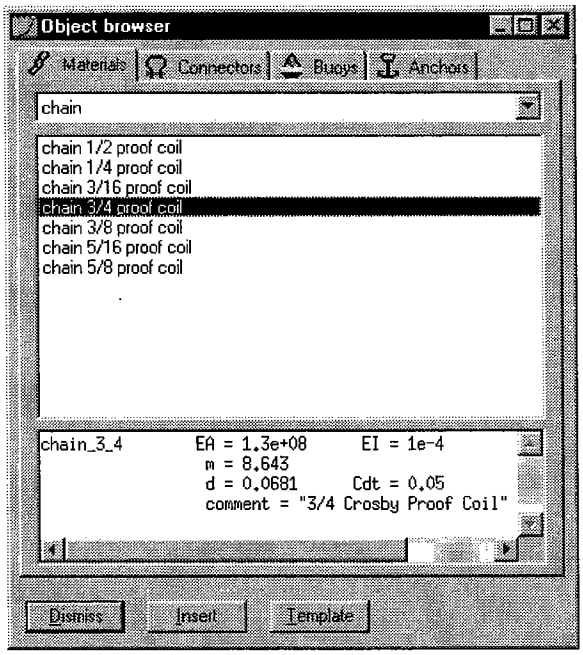


Figure 5.3: The database object browser in the WHOI Cable Windows interface.

connectors, buoys, and anchors that are currently defined in the database files. To change between object types click on the tabs at the top of the browser. Within each object type the database may be divided into sections; the section can be changed by selecting from the pull down list beneath the tabs. All of the objects defined in the current section for the current object type are shown in the large list in the middle of the browser. In the example in figure 5.3 the chain section of the materials database has been selected and all of the predefined chain types are listed. The complete definition for the currently selected object is shown in the text box beneath the main list. To use a definition highlight the object and click **Insert** or double click on the object. The definition will be inserted at the current cursor position in the main input file editor window. For objects not defined in the database, you can insert a blank template for the current object type by clicking the **Template** button.

Templates are also available 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 description 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, branches, 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.

5.3 Solving a problem

In the command-line version of *cable*, described in chapter 3, 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 5.4. 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. Checking the **AutoSolve** static solution box is equivalent to the `-auto` option in the command line version of *cable*. 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, terminal nodes, 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 a list of output nodes must also be specified. If no time series time

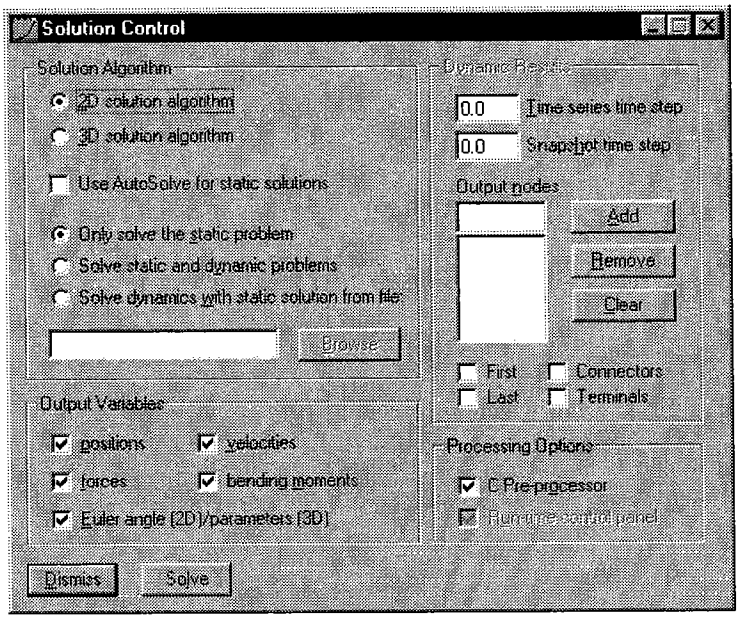


Figure 5.4: The solution control dialog available by selecting **Controls** from the **Solutions** menu.

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 3.1 and 3.4 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 5.6 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 5.5 for details on how and when you should save files and how temporary files are used within the encapsulator.

5.4 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.

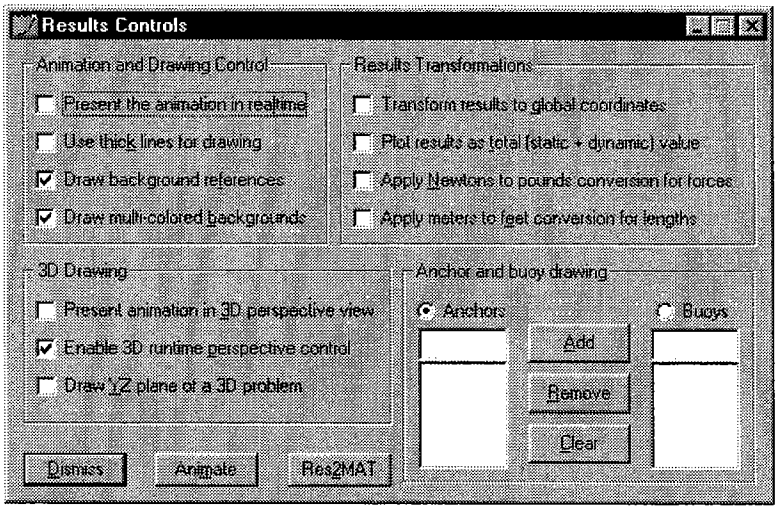


Figure 5.5: The results control dialog available by selecting **Controls** from the **Results** menu.

Details of using *animate* are provided in section 4.2. The dialog to control how *animate* is invoked is shown in figure 5.5 and is raised by selecting **Controls** from the **Results** menu.

The controls in the upper left frame of figure 5.5 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 and length units should be converted from Newtons to pounds and meters to feet (these only makes sense if the original force units are in Newtons and meters 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.

The four check boxes for results transformations on this control 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.

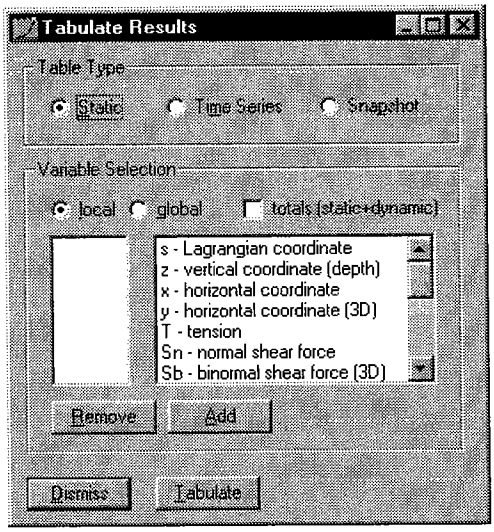


Figure 5.6: The results tabulation dialog in the WHOI Cable Windows interface.

For more immediate access to numerical rather than graphical results, *res2asc* can be used. Under the Windows interface, the ASCII tabulated results will appear directly in a separate output window. From that window they can be saved or printed. To run *res2asc* select **Tabulate** from the **Results** menu. The dialog pictured in figure 5.6 will appear to allow you to specify what type of results you want tabulated (static, time series, or snapshots) and what variables to include in each table. To add a variable select it from the list on the right and click the **Add** button. The list of available variables can be changed by toggling the **local** and **global** radio buttons.

5.5 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 5.2), 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.

5.6 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 5.1 details this complete command structure.

5.7 Installing WHOI Cable for Windows

5.7.1 System requirements

WHOI Cable for Windows is only available for 32-bit Windows (95/98/2000/NT), Intel-based platforms. Some 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 Pro/II/III architectures.

As illustrated in figure 5.1, PC X-server software must be installed if you want to take advantage of the *animate* post-processing program. Numerous companies market inexpensive PC X-server software. 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. There is an effort underway to port the free servers from the XFree86 projects to Windows. Check http://sourceware.cygnus.com/cygwin/xfree/ for the latest information on that effort.

Menu	Item	Menu key	Keyboard shortcut	Toolbar icon	Other
File	New	Alt-f, Alt-n			
	Open	Alt-f, Alt-o	Ctrl-o	file folder	
	Save	Alt-f, Alt-s	Ctrl-s	disk drive	
	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	printer	
	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	Object browser	Alt-i, Alt-w		chain+buoy+anchor+shackle	
	Section Template	Alt-i, Alt-t		pages	
	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 branch	Alt-i, Alt-h		triple point	
	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
	Tabulate	Alt-r, Alt-t	Ctrl-b	table	
	Controls	Alt-r, Alt-c	Ctrl-t		
Setup	Files	Alt-u, Alt-f			
	Fonts	Alt-u, Alt-n			
Help	Keyword	Alt-h, Alt-k	F1		
	About	Alt-h, Alt-a			

Table 5.1: Complete command structure for the WHOI Cable for Windows encapsulator.

5.7.2 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. To install WHOI Cable download the file `wcblXXX.exe` from `ftp.whoi.edu` using the login name `cable` and the password provided on your license agreement. Replace `XXX` in the filename above with the highest version number available. Start the installation process by executing the downloaded file, either by double clicking or selecting **Run** from the Windows **Start** menu. After installation you can safely remove `wcblXXX.exe`.

Your PC X-server software should generally be running before you run WHOI Cable. At startup, WHOI Cable will check to see that it can communicate with the server. If there is a problem with the server the program will suggest possible remedies. If no communication can be established the *animate* options will not be available. In order for the encapsulator to interact with the X-server, you should check 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.

5.7.3 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 available[1] ghostscript package for printing and viewing Postscript files on non-Postscript devices.

[1] from ftp.cdrom.com in pub/simtelnet/win95/print for example

5.7.4 Modifying the installation

5.7.4.1 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. Using the file setup dialog you can specify locations for the *cable*, *res2mat*, *animate*, *res2asc*, 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 \.

5.7.4.2 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.

5.7.4.3 Database files

At startup, WHOI Cable reads four files from the database directory to load objects into the object browser. The files are `material.db`, `connect.db`, `buoy.db`, and `anchor.db`. Each database file contains regular WHOI Cable object definitions delimited by special lines to provide descriptive names and sectioning information. There must be at least one section in each file. Section headings are marked by the presence of ** at the beginning of a line. Individual object definitions within a section are preceded by a line marked by a single * and a descriptive phrase. Until another section line (marked by **) or object name line (marked by *) is read, subsequent lines are assumed to be a part of a valid WHOI Cable object definition. Additional sections and objects can be added to the database files with any text editor so long as this formatting is maintained. Section names and object descriptions within sections will be sorted by WHOI Cable upon insertion into the browser. The database files are read only at program startup.

Chapter 6

Solving a Problem with WHOI Cable

In the following sections brief descriptions of a typical solution path for many different problem types are provided. Throughout the descriptions references are made to the example input files distributed with WHOI Cable. These files can be found in the `examples` subdirectory of the WHOI Cable installation directory.

6.1 Subsurface single point moorings

Subsurface single point moorings are typically the easiest problems to solve with WHOI Cable. Because the buoy is fully submerged the forces at the top of the mooring are easy to calculate and outer iterations are not required. In the `mine.cab` example, the default static solution procedure (catenary initial guess and a static relaxation solution) requires just two iterations to converge. In subsurface problems with excess line on the bottom `static-relaxation` may need to be reduced. For particularly complex geometries where reducing the relaxation factor does not help or produces very slow convergence, automatic dynamic relaxation is available. While this approach is very robust, it can take some time to converge, particularly on very long moorings.

For dynamic solutions, Morison forcing is usually most appropriate for subsurface problems. `mine.cab` illustrates the application of Morison forcing with surface waves propagating in both horizontal directions. In a two-dimensional simulation the out-of-plane propagating wave is not used.

6.2 Single point surface moorings

6.2.1 Taut moorings

Taut surface moorings with simple geometries and no excess line on the bottom are also typically very easy to solve with WHOI Cable. The fastest solution path is usually a shooting method initial guess followed by static relaxation. The static relaxation procedure will use outer iterations to calculate the draft of the surface buoy. With a shooting method initial guess the outer iterations are minimized because the calculated draft from the shooting method is fairly accurate. This is the approach taken in `taut.cab`. When a catenary solution is used to initialize the static relaxation procedure, the procedure to bracket the draft can be very slow. Automatic dynamic relaxation solutions are also available for surface moorings, but again can be slow to converge for very long moorings.

The dynamic forcing method for surface moorings is usually either `velocity` or `wave-follower`. With `velocity` the motion of the buoy in all three directions can be specified with `x-input`, `y-input`, and `z-input`. Remember that even though the forcing method is called `velocity` the inputs are defined in terms of displacement amplitude and period. If a time series of buoy velocities are known they can be specified in a separate file using `velocity-file`. For `wave-follower` the vertical motion of the buoy is governed by the waves described by `x-wave` and `y-wave`. The buoy is free to respond to horizontal forcing due to time varying wind or current in each of the horizontal directions, but horizontal forces based on waves are not part of the model. If you want to model pure heave motion with horizontal motions constrained use `velocity` and `z-input` rather than `wave-follower`.

6.2.2 Catenary moorings

The most reliable static solution procedure for shallow water catenary moorings is automatic dynamic relaxation. For `swex.cab` a chain catenary mooring in 40 m of water with a very low current, this is the only procedure that works well. Because of the low current, the curvature at the bottom is quite high. This makes the resolution of the touchdown point difficult and thus complicates the static solution. Automatic dynamic relaxation works by increasing the current until an easy static solution can be obtained (using a shooting method and a one step static relaxation). This static solution is then used as the initial condition in a dynamic solution with no waves and the current set to its real value. Over the course of the dynamic solution the mooring will relax back to its true equilibrium position under the imposed current. When the mooring has come to equilibrium the solution automatically stops. During this process WHOI Cable specifies a set of analysis parameters that are designed to make this process as robust as possible. After the procedure has converged the

parameters will revert back to their original values.

When currents are higher, as in `cmo.cab`, regular static solutions are more reliable. The example in `cmo.cab` can be solved with static relaxation using either a catenary or a shooting initial guess. Automatic dynamic relaxation is relatively fast for these shallow water moorings, however, and thus could be used even in this case without a significant performance penalty.

Both `swex.cab` and `cmo.cab` demonstrate the use of experimentally recorded time series of buoy motion in the dynamic simulations. During both the SWEX and CMO experiments time series of acceleration were recorded at the buoy. The heave component of the acceleration was then integrated into time series of velocity. These velocity time series are contained in the files defined by the `velocity-file` statements.

6.2.3 S-tether, inverse catenary, and lazy wave moorings

Like catenary moorings, s-tether moorings are most difficult to solve when currents are very low. In shallow water these difficulties can be avoided by using automatic dynamic relaxation as with catenary moorings. This is the best approach for the problem in `shallow_s.cab` for example. Because these moorings are often employed in deep water, however, it can sometimes be faster to employ regular static relaxation with carefully chosen analysis parameters. For `deep_s.cab` in 3500 m of water, the fastest solution uses a catenary initial guess and static relaxation with outer iterations. The baseline static relaxation factor is set to 0.1. Using this technique a solution is obtained in 44 outer iteration steps with between approximately 20 and 1000 iterations at each step. A shooting initial guess will not work for this problem. Automatic dynamic relaxation does work for this problem but because of the long time scales associated with the motion of this very long mooring the procedure takes a substantial amount of time (over 5000 seconds of simulation time).

6.2.4 Moorings with line floating on the surface

WHOI Cable can model moorings with line floating on the surface. Like other complex geometry surface moorings these problems are most reliably solved with automatic dynamic relaxation but may be solvable using regular static relaxation with a small relaxation factor (< 0.1) and large numbers of iterations. Both `seatex.cab` and `shallow_s.cab` (which in addition to an s-tether configuration has some line floating on the surface) are best solved with automatic dynamic relaxation.

Dynamic solutions will not be accurate for most moorings with floating line because WHOI Cable does not have a model for the wave induced motion of the line. However, if

these moorings are properly designed then dynamic motions of the mooring are small and are typically not a driving factor in design decisions.

6.3 Multipoint and branched moorings

6.3.1 Subsurface

Using `branch` definitions WHOI Cable can model both multipoint moorings and horizontal array/aquaculture longline moorings. For both cases the problem type should be set to `positioned`. In multipoint moorings like those defined in `multileg.cab` or `multileg_3d.cab` all of the legs join at a central point. This point is defined by a connector definition (rather than a buoy definition). Each branch leaving from this central connector defines a leg, as do the segments running from the first terminal to the connector and the final segments running from the connector to the second terminal. Thus a four-legged mooring will have two branches defined.

When defining horizontal array moorings branches are used to model the lines that hang down off the horizontal member. In `horiz.cab` each of these hanging strings leaves from a connector and is terminated with a sinker weight that is defined as a buoy. In `loops.cab` the branches are used to form loops that run from one point on the mainline to a second point on the mainline. Looped problems can be very difficult to solve because the force at the second end of the loop must be adjusted until the end is colocated with the user specified node on the mainline. To get a static solution for `loops.cab` adaptive relaxation had to be turned off.

The only static solution procedure available when branches are present in a problem is static relaxation with a catenary initial guess. Outer iterations are used to calculate the reaction forces at each of the anchors so that they will be placed at the location specified in the terminal definition. In these problems `static-outer-relaxation` is used after each outer iteration to calculate a correction to the applied terminal forces based on the errors in the anchor positions. A large relaxation factor will speed the solution but may lead to instabilities when the forces change dramatically from step to step. It is generally safest to set it small (between 5 and 20 maybe) initially and then increase it as the errors become smaller.

The Morison forces caused by subsurface wave particle velocities and accelerations are calculated for connectors. Just like a single point subsurface mooring then, the best choice for dynamic forcing in these cases is `morison`.

6.3.2 Surface

Both of the cases using branches described above were based on the `positioned` problem type. In general, however, the problem type really just specifies the treatment of the second terminal (at the end of the main string of segments). Thus, branches can be used with other problem types as well. The position of anchored branch terminals are only resolved in `surface` and `positioned` problems, however.

Multipoint moorings with a surface expression are most easily solved with a little bit of trickery. We cannot simply use a connector as the buoy as with subsurface problems because connectors floating on the surface are difficult to resolve in the static problem and are not accurately forced in the dynamic problem. Instead we define the problem as we would a regular surface problem with a dummy connector just below the surface buoy. A very short segment runs from this connector to the buoy at the second terminal to define the mainline of the problem. Additional legs can be defined as branches leaving from this dummy connector. In some cases the resulting problem may be solvable using regular static relaxation.

In many cases, however, the need to resolve both buoy draft and the anchor positions at the branch terminals leads to convergence difficulties in the outer iterations. One approach to overcoming these difficulties is to take advantage of symmetry and use a manual dynamic relaxation process. This approach was used successfully for the surface bi-moor problem in `bimoor.cab`. The static solution is obtained in three phases.

In the first phase the problem type is set to `positioned`, the current is turned off, the branch (the second leg) is commented out, and the second terminal with the buoy is placed at a horizontal position in the middle of the anchors and a vertical position at the approximate draft. An easy estimate of the draft can be calculated by balancing buoyancy and the weight of all mooring legs. Solving this problem with regular static relaxation gives the solution for one leg.

In the second phase, the branch is uncommented, the problem type is set to `general`, the reaction forces calculated at the first anchor in phase one are applied to the second anchor, and an appropriate vertical force is applied at the buoy. The appropriate force is simply the reaction force from phase one multiplied by the total number of legs. This problem is then solved with regular static relaxation.

Finally, the current is turned back on, the problem type is set to `surface` and the solution from phase two is loaded as the initial condition (with `-load`) in a dynamic relaxation solution using the `-dynstat` option. With the problem type reset to `surface` the actual buoy draft will be used to calculate topside forces and the system will come to a true equilibrium state.

With a complete static solution from the dynamic relaxation procedure the dynamic problem with waves can be solved just as if this was a regular `surface` problem. The entire process is obviously more complicated than any of the previously described static solution paths. It is described here both as an example of a path for this particular kind of problem and also to illustrate the wide range of possibilities that exists for determining the solution to complex problems.

6.4 Towing problems

6.4.1 Ship maneuvering

Steady towing problems are solved as static problems. They are typically very quick to solve with either catenary or shooting initial guesses and static relaxation. For maneuvering problems with time varying `x-speed` and/or `y-speed` at the second terminal (ship end) the steady state (static) solution is based on the speed at $t = 0$. The initial cable length is the summed length of all the segments in the problem. A `pay-rate` specification changes the length of the cable in the dynamic simulation. The example problem `towing.cab` illustrates varying the depth of the tow body (tow-yow'ing) using either sinusoidally varying `x-speed` or `pay-rate`. In `tow3d.cab` a circular ship maneuver is used in a three-dimensional simulation to quickly drop the tow body in depth without changing speed or paying out.

6.4.2 Tow body maneuvering

In many towing applications the ship is held fixed and the tow body is maneuverable under its own power. By specifying a current profile and zero ship speed it is possible to calculate the motions of such a vehicle given time varying thrust in the three global directions at the first terminal (tow body end). The example `rov.cab` illustrates using a thrust definition to calculate the steady state position of a deployed ROV when it is applying 100 pounds of thrust against the imposed current. In some three dimensional problems these sorts of problems can be difficult to solve with regular static relaxation. Because the steady state information is typically more interesting than dynamic results, shooting method solutions are often a good choice as the final static solution. In cases where the steady state configuration is two-dimensional, but three-dimensional dynamic results are desired you can use a two-dimensional static solution to initialize the three-dimensional dynamic solver.

6.4.3 Cable laying

Cable laying or anchor lowering problems can be defined just like any other towing problem with pay-out. A depth must be defined and variations in the bottom topography can be specified with `bottom-elevation`. Once the tow body (which must be defined as a buoy) at the first terminal strikes the bottom it is assumed to be anchored to the bottom and cannot move. An example of this kind of problem is given in `lowering.cab`.

6.5 Launch, recovery, and failure problems

6.5.1 Mooring deployment

Mooring deployment problems are defined with the problem type `deployment`. During a static solution these problems are actually treated as a special case of surface problems. The anchor is defined at the first terminal and is initially located on the surface. The speed given at the first terminal (the ship must be making some headway or there must be some current running) will be used to calculate the steady state configuration of the system, including buoy draft. For forward speed into the current that configuration will have the system streamed out behind the anchor. All of the static solution paths are available for deployment problems. For relatively short systems automatic dynamic relaxation is usually a good choice. It is also a good choice when a mooring has subsurface buoyancy elements that will float on the surface when the mooring is paid out but the anchor has not been released. Regular static relaxation (sometimes with a small relaxation factor) can also work. Either method works for the example in `deploy.cab`.

As soon as the regular dynamic solution starts the anchor is released from the ship and begins to fall. Because the dynamics and hydrodynamics of the anchor are a critical part of the system response during deployment, these problems are the only type in WHOI Cable that require a completely defined anchor. Once the anchor hits the bottom it is fixed to bottom. These simulations can take a significant amount of time simply because the physical response of these systems is such that they can take a long time to come to equilibrium once the anchor hits the bottom. Also, in deep water the anchor can take many minutes of simulation time to reach the bottom.

6.5.2 Cable breaking and mooring release

Cable breaking and mooring release problems are implemented by specifying a `release-time` at either terminal in any kind of dynamic simulation. After that point the buoy or anchor at that end will be dropped from the simulation and there will be a zero force boundary

condition imposed. The example `breaking.cab` illustrates this for the case of breaking ROV tow cable. For mooring release problems remember that the top (buoy) end is only free to move in the horizontal directions if `forcing-method` is `morison` or `wave-follower`. Note also that breaks can currently only occur at the two main terminals.

6.6 General tips and tricks

Regardless of problem type there are some general tricks that are helpful when convergence failures, singularities and instabilities occur. In this section these tricks are broken down into suggestions for static and dynamic problems.

6.6.1 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). You can also try to setting `static-initial-guess` to `shooting` if shooting method solutions are available for your problem type.

- 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.

- Low currents cause solution instabilities

  For problems with very little horizontal forcing and curves in the shape, the bending radius at the curves can be quite small, leading to difficulty in resolving static solutions. An effective solution technique for these problems is dynamic relaxation in which the the current is artificially magnified to obtain an initial solution and then set to its true value in a dynamic solution so that the system will come to the proper equilibrium position using the physical damping in the problem to smooth the solution progress. You can use the command line option `-dynstat` to achieve this manually or for many problem types you can use the `-auto` flag to automate the process.

- 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. These problems are also a good candidate for automatic dynamic relaxation (use the `-auto` command line flag with *cable*).

- 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. Using a the shooting method for the `static-initial-guess` can also speed things.

- 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.

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

  If the steady state configuration of the mooring is planar (2D) then you can use the 2D static solver to get a solution. A solution from the two-dimensional static solver can be used with the `-load` option to initialize a three-dimensional dynamic problem. This does not work going the other way. Static solutions from the 3D solver cannot be used to initialize a two-dimensional dynamic problem.

6.6.2 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`). In very rare cases you may need to adjust the parameters of the generalized-α time integration algorithm. Useful values of `dynamic-lambda` are typically in the range $-0.8 < \lambda_{1,2}^{\infty} < 0.3$. $\lambda_{1,2}^{\infty} = -0.5$ is the default.

- 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. In some cases these errors can also be eliminated by changing `dynamic-lambda`. $\lambda_{1,2}^{\infty}$ controls the frequency response of the time integration; depending on the properties and forcing conditions in your system it may be desirable to change the filtering by changing this parameter.

- 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.

- Problems with areas of high curvature, sharp bends, or slack regions are unstable

  Most of these types of instabilities can be related to the bending stiffness of the material. Try setting `EI` to a larger value. For most oceanographic materials `EI` can be as high as 10 - 100 Nm^2 without affecting the accuracy of the overall solution.

References

[1] J. Chung and G.M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation. *Journal of Applied Mechanics*, 60:371–375, 1993.

[2] Jason I. Gobat. *The dynamics of geometrically compliant mooring systems.* PhD thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution Joint Program, Woods Hole, MA, June 2000.

[3] Jason I. Gobat, Mark A. Grosenbaugh, and Michael S. Triantafyllou. WHOI Cable: Time domain numerical simulation of moored and towed oceanographic systems. Technical Report WHOI-97-15, Woods Hole Oceanographic Institution, November 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] C.Y. Liaw, N.J. Shankar, and K.S. Chua. Large motion analysis of compliant structures using Euler parameters. *Ocean Engineering*, 16:545–557, 1989.

[6] 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, 1989.

[7] Andrew H. Sherman. Algorithms for sparse Gaussian elimination with partial pivoting. *ACM Transactions on Mathematical Software*, 4:330–338, 1978.

[8] Athanassios A. Tjavaras. *Dynamics of highly extensible cables.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1996.

[9] William C. Webster. Mooring induced damping. *Ocean Engineering*, 22:57–591, 1995.

[10] 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 – July 1998

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

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

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. WHOI-2000-08	2.	3. Recipient's Accession No.

4. Title and Subtitle
WHOI Cable v2.0: Time Domain Numerical Simulation of Moored and Towed Oceanographic Systems

5. Report Date
July 2000

6.

7. Author(s) Jason I. Gobat and Mark A. Grosenbaugh

8. Performing Organization Rept. No.
WHOI-2000-08

9. Performing Organization Name and Address

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C) N00014-92-J-1269
(G) N00014-95-1-0106

12. Sponsoring Organization Name and Address

Office of Naval Research

13. Type of Report & Period Covered
Technical Report

14.

15. Supplementary Notes
This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-2000-08.

16. Abstract (Limit: 200 words)

This report describes version 2.0 of a numerical program 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- and multi-point mooring problems, systems with both ends anchored on the bottom, and towing and drifter problems. Forcing includes waves, current, wind, ship speed, and pay-out of cable. The programs that make-up WHOI Cable run under Unix and Windows. There is a familiar graphical interface available for Windows platforms. The report includes detailed instructions for formulating problem input files and running the programs.

17. Document Analysis **a. Descriptors**
mooring dynamics
cable dynamics
towed systems

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement

Approved for public release; distribution unlimited.

19. Security Class (This Report)
UNCLASSIFIED

20. Security Class (This Page)

21. No. of Pages
89

22. Price

(See ANSI-Z39.18) *See Instructions on Reverse*

OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce