TD_xxx: Matlab code for simulation of floating body motions in the time domain

Andrew Hamilton
MBARI

August 13, 2012

Contents

1 Documentation

1.1 Introduction

TD_xxx is a collection of matlab code for computing the motions of one or more floating bodies in the time domain due to excitation by incoming waves. As illustrated in Figure 1, the code takes as inputs the characteristics of the various elements that make up the system (buoy size, tether characteristics, etc) and produces a time history of the motions and forces that will result from a specific set of environmental conditions (sea-state, wind speed, etc). The code is designed to perform these computations in three dimensions with six degrees of freedom, but to date only computations in two dimensions (heave, surge and pitch) have been performed as this tends to be a more useful case.

At its core, the code solves the equations of motion of one or more rigid bodies that are subject to external forces defined individually for each body (gravity, wave forcing, wind forcing, etc) and forces due to connections between each body (tethers consisting of springs, damping elements, or arbitrary force/elongation relationships). Use of the code involves defining the characteristics of each rigid body (mass, size, waterplane area, drag coefficients, etc), and specifying forces that act on each body (gravity, buoyancy, wave forces, tether forces, etc). With this definition complete the code integrates the equations of motion and produces a time-history of motions and forces for each body. Figure 1 illustrates the flow of the solution.

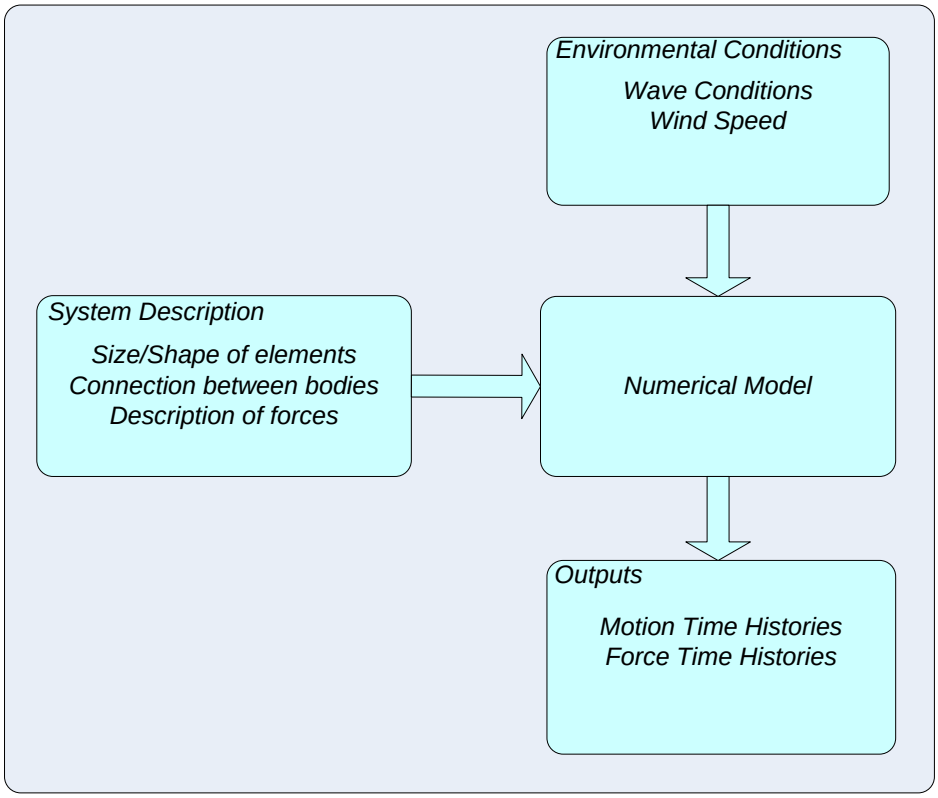

Figure 1: Heave motion results for the SingleBodyEx example.

Several important forcing functions are supplied with the code. The most complicated and challenging to implement is the wave forcing which consists of wave excitation forces, wave-radiation forces, and wave-diffraction forces, following standard linear wave theory. The wave forces are computed from the impulse response functions for the specific body that must be pre-computed by a program such as WAMIT. With those functions supplied, the computation and application of all wave forces is done automatically. Less complicated but equally important are the forces on each body due to gravity, buoyancy, wind, and currents.

A second type of force that applies to pairs of bodies is also included. These forces are defined as functions of the relative positioning of two bodies and apply an equal an opposite force to specified attachment points on each body. The simplest example of this type of force is a spring tether that connects two bodies. In this case the force that applies to each body is a function of the spring constant and the relative position of each body. As the bodies move apart the spring force goes up, and as the bodies move together the spring force goes done. Nonlinearities can be included such as a slack condition in which the tether exerts no force on the bodies if they are closer together than some specified distance. As present, there is no facility to solve for the cable-dynamics of the tethers themselves. The effect of drag on the tether can be included in a lumped sense as a force that acts on each body, but not in a way that captures the deformation of the tether from a straight line. Therefore, the code is best suited for relatively short tethers with minimal drag forcing that would cause the tether to deviate from a straight line between the two bodies.

1.2 Code Specifics

The code utilizes Matlab's object-oriented features and provides classes and functions to allow the solution of problems involving coupled motions of more than one body, subject to a variety of external forcing types. There are two main objects, one which represents the incoming sea-state (IncWave), and the other that represents each rigid body (RigidBody). In a typical scenario there will be only one IncWave object and one or more RigidBody instances. Once the user sets up the characteristics of the objects that represent a specific system, the code computes the resulting motion and force histories. As the simulation progresses, the motion histories of each body are kept within the RigidBody objects. The functions that implement forces on each body take one or more RigidBody objects as arguments and therefore have access to the time history of that bodies motions. The wave-radiation forces in particular make use of this in particular since the wave forcing at any given time depends on the bodies position in at all past times as well as it's current position, velocity, and acceleration.

2 Examples

This section contains two examples that illustrate the use of the code. The listing shown is the top level matlab script that the user generates for a given problem. This script calls the underlying objects and functions that make up the simulation toolkit. The first example is a single floating body subject to incoming waves. The second example adds a submerged body that is tethered to the floating body by a spring-damper system. These input files are typical of the user generated component of a TD_xxx solution. If a solution only requires existing force functions, generating this top level file is all that is needed to simulate a new problem. If special forcing functions are required to capture the physics of a situation, then those forcing functions need to also be written as described in section 3

2.1 Single Body Example

The file *TD_SingleBodyExample.m* contains an example that uses the above classes and functions to solve the wave-excited time-domain motion problem for a freely floating body. In this example the body used is a truncated cylinder of depth 1m and diameter 2m. The hydrodynamic coefficients for this geometry have previously been computed by WAMIT are stored in the directory "WAMITResults" (see WAMIT documentation for details).

First the water density (rho), simulation time (tf), time step size (dt) and the number of timesteps to compute (N) are established.

```
clear all
close all

rho = 1025;      %Define water density (kg/m^3)
tf = 60;        %Set simulation duration in seconds
dt = .1;         %Set time-step size in seconds
N = tf/dt;       %Number of timesteps to compute
```

A RigidBody object is defined and the timestep size and physical characteristics of the buoy are assigned.

```
%Set up "Buoy"
Buoy = RigidBody;
Buoy = set(Buoy,'name','Surface Buoy');
Buoy = set(Buoy,'dt',dt,'N',N);       %Set timesteps
Buoy = set(Buoy,'dof',6);             %With 6 DOF.
Buoy = set(Buoy,'Fixed',0);           %Set free to move
Buoy = set(Buoy,'mass', 2050);        %Set mass in kg
Buoy = set(Buoy,'S',5.0);             %Set waterplane area in m^2
Buoy = set(Buoy,'disp',2.0);          %Set displacement in m^3;
Buoy = set(Buoy,'g',9.81);               %Set gravitational acceleration constant
Buoy = set(Buoy,'x0',[0 0 0]);        %Equilibrium Position in global coordinates
Buoy = set(Buoy,'xG',[0 0 -0.38]);    %Set location of center of gravity in meters
Buoy = set(Buoy,'xB',[0 0 -0.22]);    %Set location of center of buoyancy in meters
Buoy = set(Buoy,'S11',1.92,'S22',1.92);  %S11 = S22 = (-1)(-1)(3.14159) m^4
```

```
I = zeros(3);
I(1,1) = 702.6; I(2,2) = 702.6; I(3,3) = 588.7;
Buoy = set(Buoy,'I',I);                    %Set moments of inertial
clear I;
Buoy = set(Buoy,'Area',[1.935 1.935 4.9]);  %Set submerged projected area in m^2
Buoy = set(Buoy,'Cd',[10.0 1.5 1]);  %Set drag coefficients in three directions (x,
Buoy = set(Buoy,'AreaW',[.875 .875 0]); %Set area subject to wind forcing
Buoy = set(Buoy,'CdW',[1.5 1.5 0]);  %Set wind drag coefficients
```

The frequency domain hydrodynamic coefficients as generated from WAMIT are read and the infinite frequency added mass coefficients are assigned to the buoy RigidBody object. As documented above, the function "ReadData" used here takes a filename as an argument and returns a structure containing the needed data read from the file.

```
%Read Frequency Domain Hydrodynamic Data from file.
filename = ['WAMIT_Results\SingleBodyExample' ];
disp('Reading frequency domain hydrodynamic coefficients');
FD_coefs = ReadWamitFD(filename);
for i = 1:6
    for j = 1:6
        mu_inf(i,j) = rho*FD_coefs.X{i,j}(2);
    end
end
Buoy = set(Buoy,'mu_inf',mu_inf);
clear mu_inf;
```

The impulse response functions that represent the floating bodies wave-hydrodynamic behavior is read and computed by the function "ComputeWamitIRF_coefs".

```
%Compute Impulse Response Functions from Frequency Domain Results
disp('Computing impulse response functions');
L = ComputeWamitIRF_coefs(filename);
```

The incident wave profile is computed next by using the IncWave object. In this example, a monochromatic wave-profile is chosen, with a period of 8 seconds. The time-steps that the wave elevation is computed at is generated by the "Buoy" object, ensuring that the time-steps of the incident wave profile match that of the solution. Because the forces on the body due to the incident wave persist after the peak of the wave has passed the centerline of the body, a longer incident wave profile is needed than the length of the simulation. The time specification in the call to "eta" reflects this.

```
%Compute incident wave profile
wave = IncWave;
wave = set(wave,'A',1);          %Amplitude = 1 meter
wave = set(wave,'T',8);          %Period = 10 seconds
wave = set(wave,'Type',1);       %Type 1 = monochromatic waves
wave = set(wave,'beta',-1);      %Waves moving towards negative x

```

```
disp('Computing incident wave profile');
t1= dt:dt:L{7,1}.breaks(end);
t = get(Buoy,'t');  %Get timesteps from Buoy object.
eta0 = eta(wave,0,[t t(end)+t1]);
```

Separate functions are assigned to the body that compute each of the various forces that act on the body. The "F_HydroStatic" and "F_Weight" functions depend only on characteristics of the Buoy object, so don't require any additional arguments. The function that computes the memory component of the radiation forces requires knowledge of the bodies impulse response functions. Matlabs "anonymous function" feature is used to pass this additional argument to the function as a parameter, while still allowing the Mem_Radiation function to be called later with the standard argument list. The function that computes the wave-exciting force requires the impulse response functions and wave elevation history so these are also passed as parameters using the anonymous function feature.

```
%Add forces to Buoy
Buoy = add_Force_fcn(Buoy,@F_HydroStatic);
Buoy = add_Force_fcn(Buoy,@F_Weight);
Buoy = add_Force_fcn(Buoy,@(body,RB,n) Mem_Radiation(body,RB,n,L));
Buoy = add_Force_fcn(Buoy,@(body,RB,n) F_Exciting(body,RB,n,L,eta0));
Buoy = add_Force_fcn(Buoy,@(body,RB,n) F_Drag_Inc(body,RB,n,'Surface Buoy',wave));
```

The problem is now completely set up and the final step is to solve the equations of motion at each subsequent timesteps after an initial condition is established. The "Solve_RK4" function performs the solution and requires that all objects in the system be assembled in a single Matlab "cell array". Because there is only one rigid body in this example the cell array RB only has one cell. After the solution is complete the cell array contents are copied back for convenience in plotting the solution.

```
RB{1} = Buoy;

%Solve for motion
RB = Solve_RK4(RB);

Buoy = RB{1};
```

At this point the solution is complete and the motion history of the floating body is contained in the Buoy object. The "get" method of the Buoy object is used to make a copy of these values for plotting.

```
%Plot Results
x = get(Buoy,'x');
xdot = get(Buoy,'xdot');
xddot = get(Buoy,'xddot');
eta0 = eta(wave,0,t);
figure('Position',[520 548 864 550]);
plot(t,x(:,3),t,xdot(:,3),t,xddot(:,3));
set(gca,'FontSize',14);
```

```
legend('pos (m)','vel(m/s)','accel(m^2/s)','Location','NorthWest');
ylabel('Heave motions')
xlabel('time (s)');
print('-dpdf','SingleBodyExampleMotions');
```

The above code computes the motions of a floating truncated cylinder due to an incoming wave-field of unit amplitude and a period of 8 seconds. The incident wave field amplitude ramps up to its final amplitude over several wave periods, as defined by the IncWave object. Figure 2 shows the computed heave motion, which increases in amplitude and achieves a steady state motion after the transient motions introduced by the ramp die away. The drag forces on the buoy included with the "F_Drag_Inc" function are nonlinear and as a result the solution is not sinusoidal as would be the case in a perfectly linear problem.

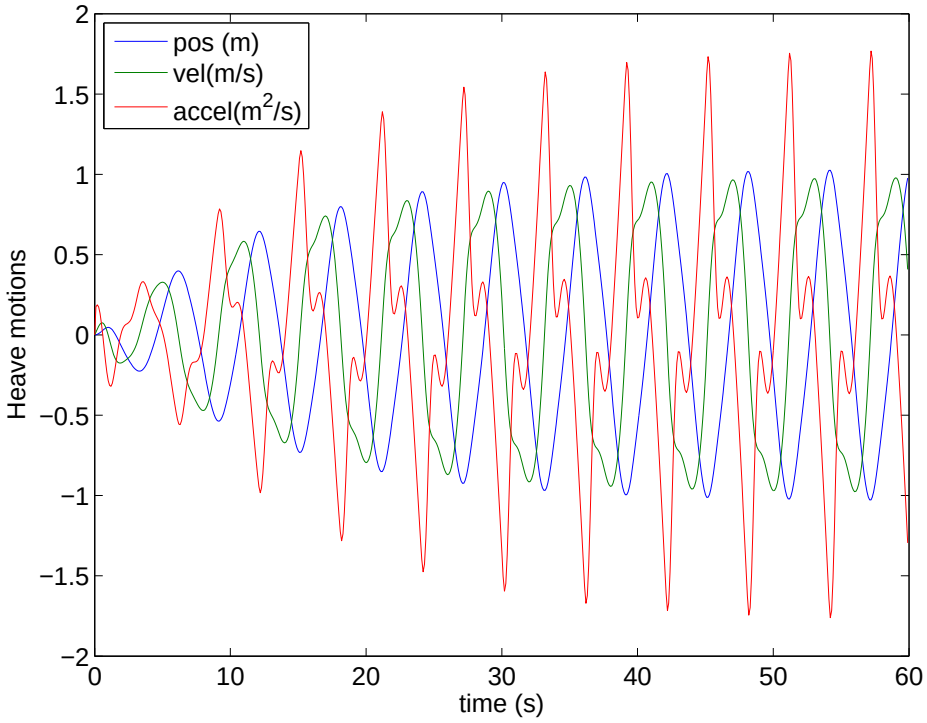

Figure 2: Heave motion results for the SingleBodyEx example.

2.2 Two Body Example

3 Adding new force functions

4 Reference

4.1 Classes

In addition to the methods described below for each class, Each class has methods to *set/get* properties and display the object. Additional methods for each class are documented below.

4.1.1 IncWave

The IncWave class contains parameters that describe an incident wave field and methods that return the appropriate wave elevation at a specified location and time. Currently two types of incident wave are supported, a wave field that is characterized by the Pierson-Moskowitz spectrum, and a monochromatic incident wave profile. In each case, the wave elevation is zero everywhere at time zero and ramps to the full amplitude specified over the first few cycles of motion.

Properties

- Type – Wave type: 0=Pierson-Moskowitz Spectrum, 1 = monochromatic wave.
- A – Wave Amplitude
- T – Wave Period
- g – Acceleration due to gravity in appropriate units relative to A and T.

Methods

- IncWave() – Constructor, creates default object with Type = 1, $A = 1$, $T = 2$, $g = 9.81$
- eta(p,x,t) – Method to compute and return wave elevation at specified position and time. Method returns an array of values with the number of rows equal to the dimension of x and the number of columns equal to the dimension of t. The p argument is an IncWave object.

4.1.2 RigidBody

The RigidBody class represents the characteristics, position, and velocity of a rigid body. The class can be assigned one or more forces that act on the body and that are a function of the bodys position, velocity, acceleration, and time. As described in section 4.2 below, functions external to the class are used to compute these forces and can depend on parameters that are not properties of the RigidBody class. Each RigidBody object contains the position, velocity, acceleration, and force history for the body it represents. External functions that compute forces on the body have access to these to allow forces to be dependent on past motions of the bodies involved as well as the current state.

Properties

- *dof* – Degrees of freedom, this is the number of positions, velocities, and acclerations that are kept track of by this object.
- *dt* – Timestep size.
- *n_tsteps* – Timestep size.

- t – Array of times the position is represented at (i.e. 0:dt:n_tsteps*dt).
- $x0$ – (x,y,z) components of the equilibrium position of the body in the global coordinate system.
- x – dof components of the bodies position at each timestep. Is an array with dimension [n_tsteps X dof].
- $xdot$ – dof components of the bodies velocity at each timestep. Is an array with dimension [n_tsteps X dof].
- $xddot$ – dof components of the bodies acceleration at each timestep. Is an array with dimension [n_tsteps X dof].
- $Fixed$ – Flag indicating if the bodies position is to be considered prescribed or is to be solved for.
- $mass$ – Bodies mass.
- $disp$ – Bodies displacement at equilibrium.
- S – Bodies undisturbed waterplane area.
- $S11$ – 2nd moment of waterplane area, about x-axis.
- $S22$ – 2nd moment of waterplane area, about y-axis.
- xG – Bodies center of gravity (in body fixed coordinates)
- xB – Bodies center of buoyancy (in body fixed coordinates in equilibrium position)
- g – Acceleration due to gravity. (default = 9.81)
- rho – Density of fluid body is floating in. (default = 1025)

Methods

- RigidBody() – Constructor, creates default object
- MassMatrix(p) – Returns the mass matrix for the body based upon the properties $mass$, xG, and I.
- add_Force_fcn(p,fcn) – Adds a function to the list of functions that are called by the object when evaluating the forces acting on the body. fcn is a function handle for a function that takes a RigidBody object and a timestep specification as arguments. i.e. fcn(RigidBody,n). Additional parameters can be passed to the fcn through the "anonymous function" features of matlab. See section 4.2 for details.
- SumForces(p,n) – Iterates through all force functions that have been added to the object, calls those functions to compute the force acting on the body at timestep n, and creates a sum that represents the total force vector acting on the body at timestep n.
- RHS(p,n) – Combines the bodies mass matrix, infinite frequency added mass matrix, and the result from SumForces into the a right hand side vector evaluated at timestep n.

4.2 Functions

There are a number of functions that take various objects as arguments but that aren't part of a class. Each RigidBody object can be assigned a collection of functions that compute the force on the body based upon the bodies properties (position, velocity, acceleration history are most useful), and a timestep specification. In order to present a uniform interface to the RigidBody class, these functions must have an argument list of (p,n), where p is a RigidBody object and n is the timestep specification. If one of these functions requires additional parameters that are not a property of the RigidBody class, these parameters can be passed to the function by using the "anonymous function" feature of matlab. In this way, the additional parameters are specified when the function is added to the RigidBody class (using add_Force_fcn). Because the parameters are assigned to the function at this time, the parameters are fixed and can not change through the simulation. The examples below (section 2.1 below illustrate this.

Functions

- F_Weight(body,n) – Returns force and moment vector due to the bodies weight.
- F_Hydrostatic(body,n) – Returns force and moment vector due to the hydrostatic force, which depends on the bodies position for surface piercing bodies with $S > 0$.
- F_Exciting(body,n,L,eta0) – Returns the wave exciting force and moment vector due to the incident and diffracted wave field. This force depends upon the local wave elevation $eta0$, and the incident wave impulse response function, contained in L.
- Mem_Radiation(body,n,L) – Returns the memory component of the force due to the waves radiated by the bodies motion. This force depends upon the radiation impulse response function, contained in L.
- ReadData(filenm) – Reads frequency domain hydrodynamic data from file created by wdra3d program.
- L = ComputeIRF(w,F,tau) – Convert frequency domain hydrodynamic data to time domain impulse response functions (L) as a function of timesteps specified in tau. w is a vector of frequencies that the hydrodynamic coefficients in F are computed at.