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

Andrew Hamilton

February 25, 2009

Contents

Chapter 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. 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.

This documentation describes the main objects and functions, and describes how they can be combined and extended to solve a variety of problems. Two examples are included that illustrate the solution of the motion of a single body due to incoming waves, and a more involved example that illustrates the coupled motion solution of two bodies connected by a tether modeled as a spring/damper.

1.2 Adding new force functions

Chapter 2

Examples

2.1 Single Body Example

The file *TD_SingleBodyEx.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. The body used is a truncated cylinder of depth 1m and diameter 2m. The hydrodynamic coefficients for this geometry as computed by wdra3d are in a file called t_cyl.out (see wdra3d documentation for details).

First the time step size (dt) and the number of timesteps to compute (N) are established.

```
clear all
close all

dt = 0.1;          %timestep size
tf = 50;
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,'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',3220.13,'S',3.14159,'disp',3.14159,'g',9.81);
Buoy = set(Buoy,'xG',[0 0 -.25]);
Buoy = set(Buoy,'xB',[0 0 -0.5]);
Buoy = set(Buoy,'S11',0.79,'S22',0.79);   %m^4
I = zeros(3);
I(1,1) = 1613.25; I(2,2) = 1615.53; I(3,3) = 1613.25;
Buoy = set(Buoy,'I',I);
clear I;
```

The frequency domain hydrodynamic coefficients as generated from wdra3d are read from the file "t_cyl.out" 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.
disp('Reading frequency domain hydrodynamic coefficients');
FD_coefs = ReadData('tcyl.out');
for i = 1:6
    mu_inf(i) = real(FD_coefs.F{i,i}(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 computed. This requires a time specification tau that represents the time axis of the impulse response function, this should be long enough to allow all of the impulse response functions to decay to zero.

```
%Compute Impulse Response Functions from Frequency Domain Results
tau = 0:dt:10;
disp('Computing impulse response functions');
L = ComputeIRF_coefs(FD_coefs.w,FD_coefs.F,tau);
```

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 4 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
t = get(Buoy,'t');
T = 6;  %seconds
wave = IncWave;
wave = set(wave,'T',T);
disp('Computing incident wave profile');
eta0 = eta(wave,0,[t t(end)+tau]);
```

Seperate 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 "Annonymous 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 annonymous 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,n) Mem_Radiation(body,n,L));
Buoy = add_Force_fcn(Buoy,@(body,n) F_Exciting(body,n,L,eta0));
```

The problem is now completely set up and the final step is to solve the equations of motion at each subsequent timesteps. The code below implements a 4th order Runge-Kutta scheme which performs 4 right hand side (RHS) evaluations at each timestep. These RHS evaluations are done at different fractional timesteps and with different state vectors as arguments. The RHS method of the RigidBody object computes the right-hand side as a sum of all the forces added to the object with the specified arguments. The RHS method uses the state-vector (sv) property as the solution argument and the specified timestep as the timestep argument. The timestep argument can be a real number with the fractional part indicating where in the current timestep the right hand side should be evaluated. For forces that don't depend explicity on the time argument (such as F_Hydrostatic and F_Weight), this fractional part is ignored. Force functions that depend explicitly on time (such as M_Radiation and F_Exciting) utilize the fractional timestep information as needed.

```
%Set Initial Conditions %Offset position from equilibrium position.
Buoy = set_pos(Buoy,[0 0 0 0 0 0],1);

%Solve for motion
state = zeros(12,1);
for n = 1:get(Buoy,'N')-1
  disp(['Timestep = ' num2str(n)]);
  state(1:6) = get_pos(Buoy,n);
  state(7:12) = get_vel(Buoy,n);

  Buoy = set(Buoy,'sv',state);
  k1 = dt*RHS(Buoy,n);

  Buoy = set(Buoy,'sv',state+k1/2);
  k2 = dt*RHS(Buoy,n+.5);

  Buoy = set(Buoy,'sv',state+k2/2);
  k3 = dt*RHS(Buoy,n+.5);
  Buoy = set(Buoy,'sv',state+k3);
  k4 = dt*RHS(Buoy,n+1);

  R = k1/6+k2/3+k3/3+k4/6;

  state = state + R;

  Buoy = set_pos(Buoy,state(1:6),n+1);
  Buoy = set_vel(Buoy,state(7:12),n+1);
  Buoy = set_accel(Buoy,R(7:12)/dt,n+1);
```

```
end
```

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 Buoy motion results
x = get(Buoy,'x');
xdot = get(Buoy,'xdot');
xddot = get(Buoy,'xddot');
figure
plot(t,x(:,3),t,xdot(:,3),t,xddot(:,3));
legend('pos (m)','vel(m/s)','accel(m^2/s)');
xlabel('time (s)');
```

The above code computes the motions of a floating truncated cylinder due to an incoming wave-field of unit amplitude and a period of 6 seconds. The incident wave field amplitude ramps up to its final amplitude over several wave periods, as defined by the IncWave object. Figure 2.1 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.

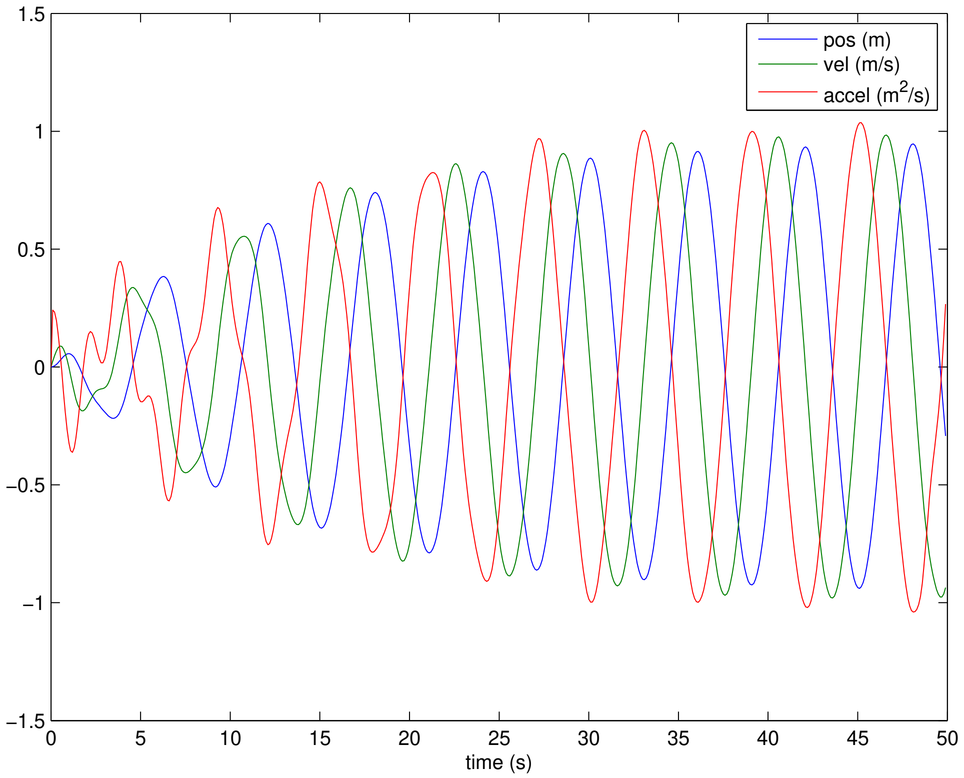

Figure 2.1: Heave motion results for the SingleBodyEx example.

2.2 Two Body Example

Chapter 3

Reference

3.1 Classes

In additon 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.

3.1.1 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.

3.1.2 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 bodies position, velocity, acceleration, and time. As described in section 3.2 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 3.2 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.

3.2 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, accleration 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) – Conver fequency 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 coeeficients in F are computed at.