WEC-Sim Training Course

Online Training Materials

PRESENTED BY

WEC-Sim Development Team

WEC-Sim Overview

What is WEC-Sim?

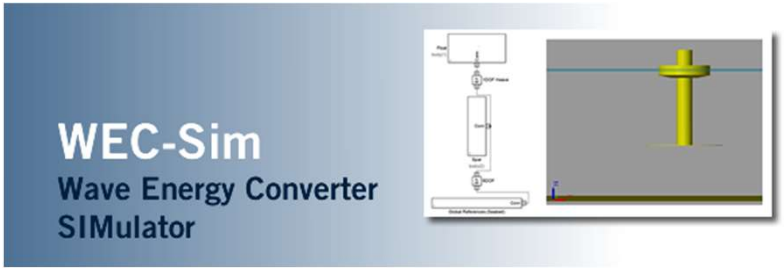
WEC-Sim (Wave Energy Converter Simulator)

- Simulates wave energy converter dynamics in operational waves
- Time-domain rigid body equation of motion solver based on Cummins' formulation
- Open source software developed in MATLAB/SIMULINK
  - Available at https://github.com/WEC-Sim/WEC-Sim
- Joint NREL/Sandia project funded by the US Department of Energy
- First Release: v1.0 in June 2014
- Current Release: v5.0.1 in Sept 2022

U.S. DEPARTMENT OF ENERGY

Sandia National Laboratories

NREL
NATIONAL RENEWABLE ENERGY LABORATORY

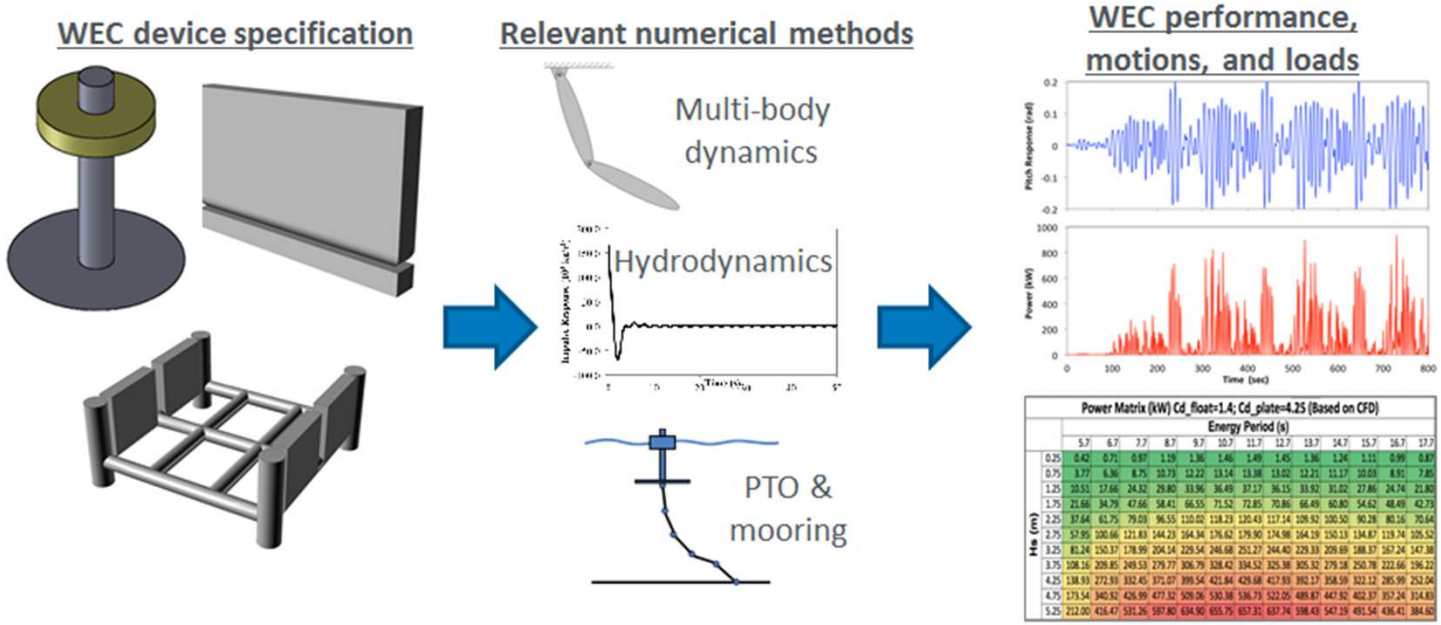


Apache 2.0

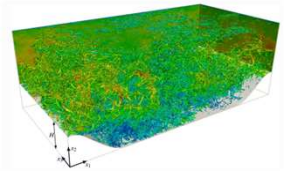
License:

Why use WEC-Sim?

- WEC-Sim has the ability to model the dynamics of devices that are comprised of rigid bodies, power-take-off (PTO) systems, and mooring systems.

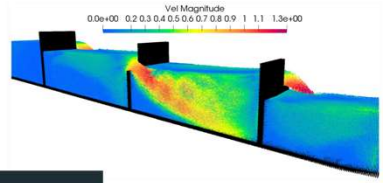


Why use WEC-Sim?



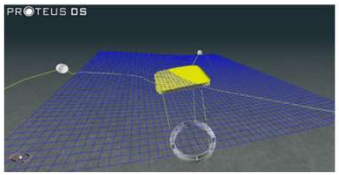
English, A., Domínguez, J.M., Vacondio, R. et al. (2022)

LES SPH (SPH-FLOW, SPHYSICS, …)



Krank, B., Kronbichler, M. & Wall, W.A (2018)

RANSe based approaches (STARCCM+, FLUENT, ISIS-CFD, ICARE-SWENSE, OPENFOAM…)



https://dsaocean.com/tag/oscilla-power/

Non-linear potential flow based approaches (LAMP3-4, AEGIR)

CPU time/ Simulation Time

10^{10}

10^{8}

10^{6}

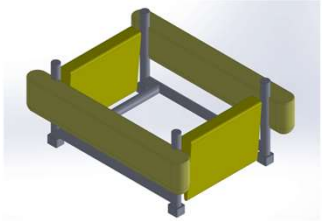
10^{4}

10^{2}

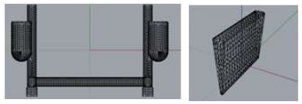
Real Time → 1

10^{-2}

10^{-4}



Linear **time domain** potential flow theory based BEM Codes & approaches based on **Morison** equations (LAMP1-2, Orcaflex, Deeplines, **WEC-SIM**, InWave, Proteus3D, …)



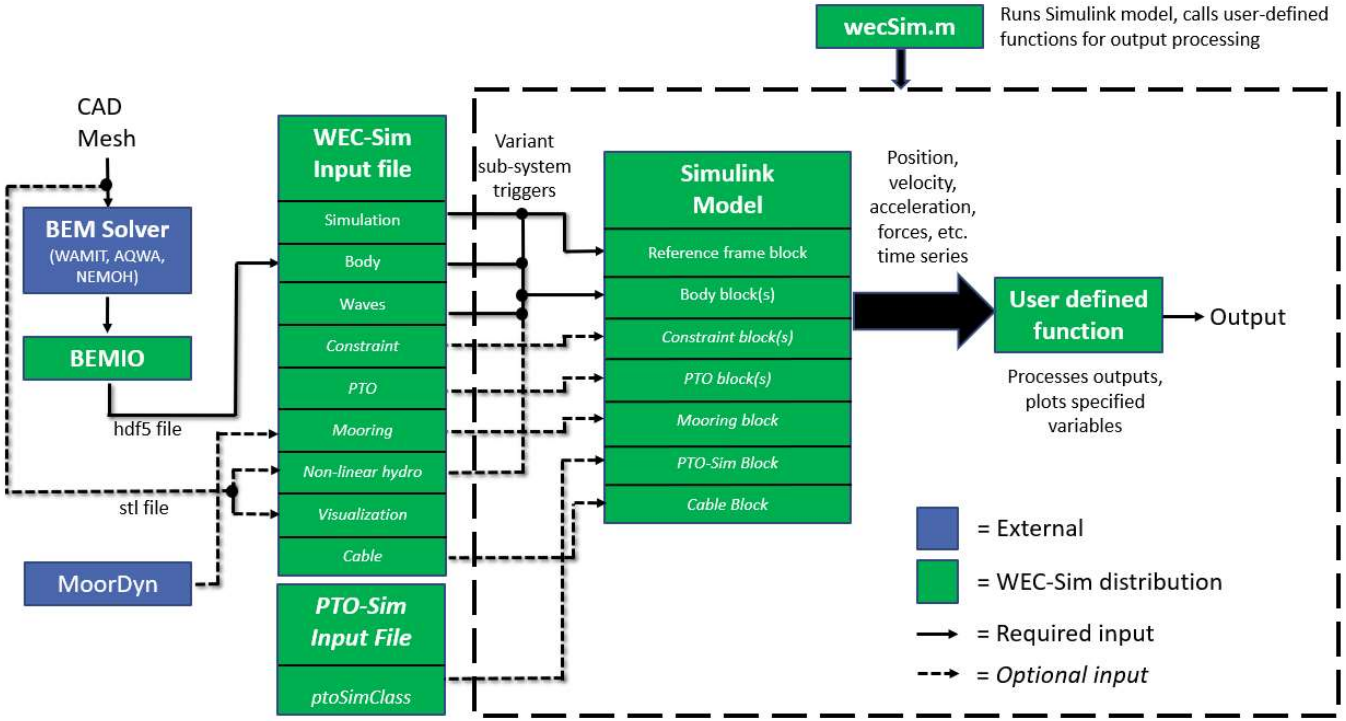
Linear **frequency domain** potential flow theory based BEM Codes (WAMIT, NEMOH, Capytaine…)

Low Fidelity

High Fidelity

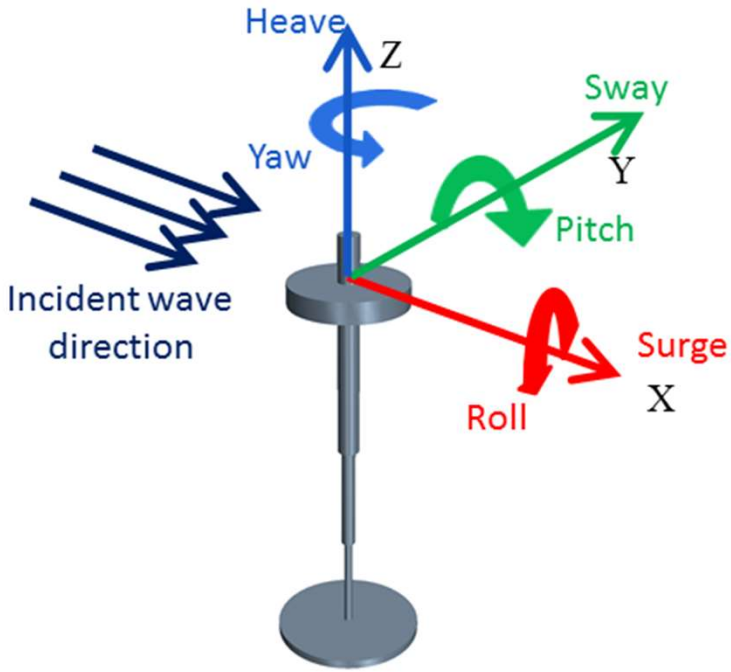
Why use WEC-Sim?

- WEC-Sim has the ability to model the dynamics of devices that are comprised of rigid bodies, power-take-off (PTO) systems, and mooring systems.
- WEC-Sim uses hydrodynamic coefficients derived from frequency-domain boundary element (BEM) simulations to model the relevant hydrodynamics.



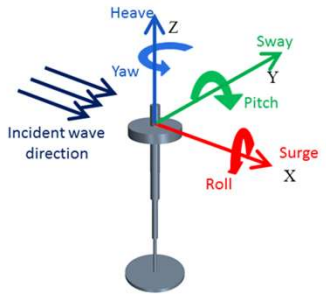
Why use WEC-Sim?

- WEC-Sim has the ability to model the dynamics of devices that are comprised of rigid bodies, power-take-off (PTO) systems, and mooring systems.
- WEC-Sim uses hydrodynamic coefficients derived from frequency-domain boundary element (BEM) simulations
- Time-domain simulations are performed by solving the governing WEC equations of motion in 6 degrees-of-freedom.



WEC-Sim Theory

- Dynamics simulated by solving time-domain equation of motion (Cummins, 1962)



$$m\ddot{x}(t) = f_{hs}(t) + f_{ex}(t) + f_{rad}(t) + f_{v}(t) + f_{pto}(t) + f_{m}(t)$$

Hydrostatic restoring force

Wave excitation & diffraction force (from BEM simulations)

Radiation force: added mass and radiation damping (from BEM simulations)

Viscous force

Power take-off force

Mooring force

- Use radiation and diffraction method and calculate the hydrodynamic forces from frequency-domain Boundary Element Method (BEM)

$$f_{rad}(t) = -A_{\infty}\ddot{X} - \int_0^t K(t-\tau)\dot{X}(\tau)d\tau$$

$$f_{ex}(t) = \Re\left[R_f F_X(\omega_r) e^{i(\omega_r t+\phi)} \int_0^{\infty} \sqrt{2S(\omega_r)d\omega_r}\right]$$

$$= \int_{-\infty}^{\infty} \eta(\tau) f_e(t-\tau)d\tau$$

WEC-Sim Software Requirements

CAD (Computer-aided design), e.g. Rhinoceros, SolidWorks, ANSYS, etc.

BEM (Boundary Element Method), e.g. WAMIT,Capytaine, NEMOH, AQWA

WEC-Sim (Wave Energy Converter Simulator)

- http://wec-sim.github.io/WEC-Sim/
- Requires MATLAB, Simulink, Simscape, and Simscape Multibody
- Includes BEMIO (Boundary Element Method Input/Output)

Required Toolbox	Oldest Compatible Version
MATLAB	Version 9.9 (R2020b)
Simulink	Version 10.2 (R2020b)
Simscape	Version 5.0 (R2020b)
Simscape Multibody	Version 7.2 (R2020b)

WEC-Sim Overview

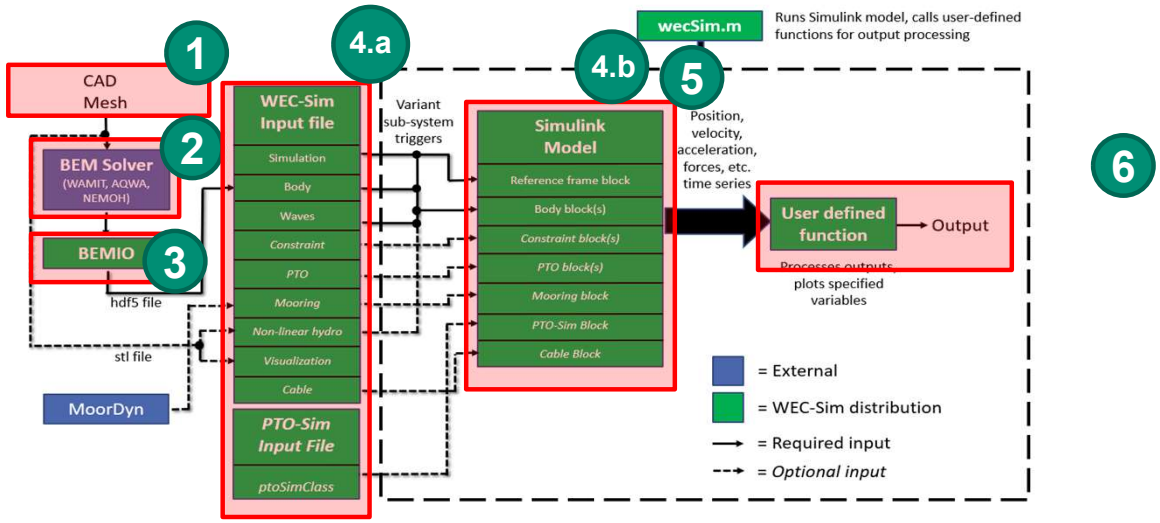
General steps to simulate a floating device using WEC-Sim

1. Generate device CAD geometry and mesh
2. Calculate hydrodynamic coefficients using a BEM code such as WAMIT, NEMOH, Capytaine, etc.
3. Run BEMIO to generate .h5 file

4.a Write WEC-Sim Input file

4.b Build the Simulink model

5. Run cases
6. Analyze WEC-Sim output

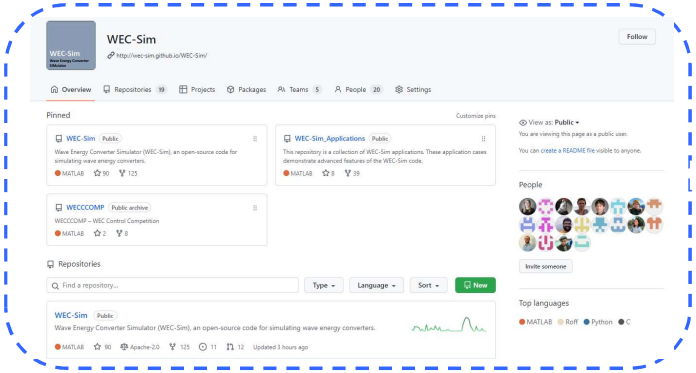


GitHub Repositories

https://github.com/WEC-Sim

WEC-Sim Source Code

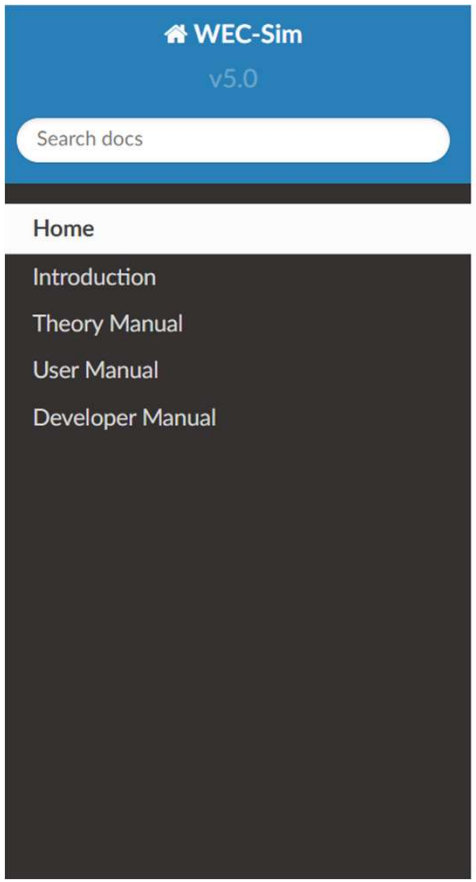
Additional Applications

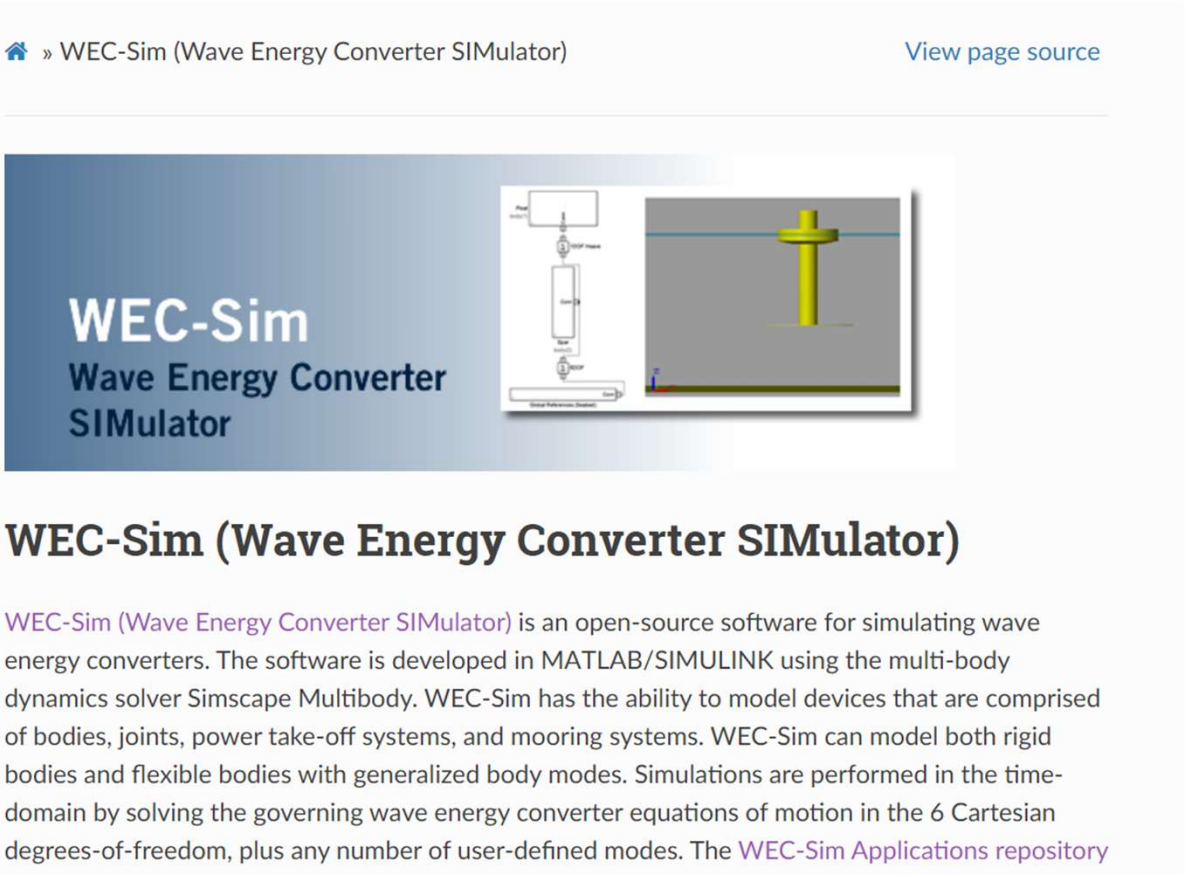


Repository Landing Page

Documentation

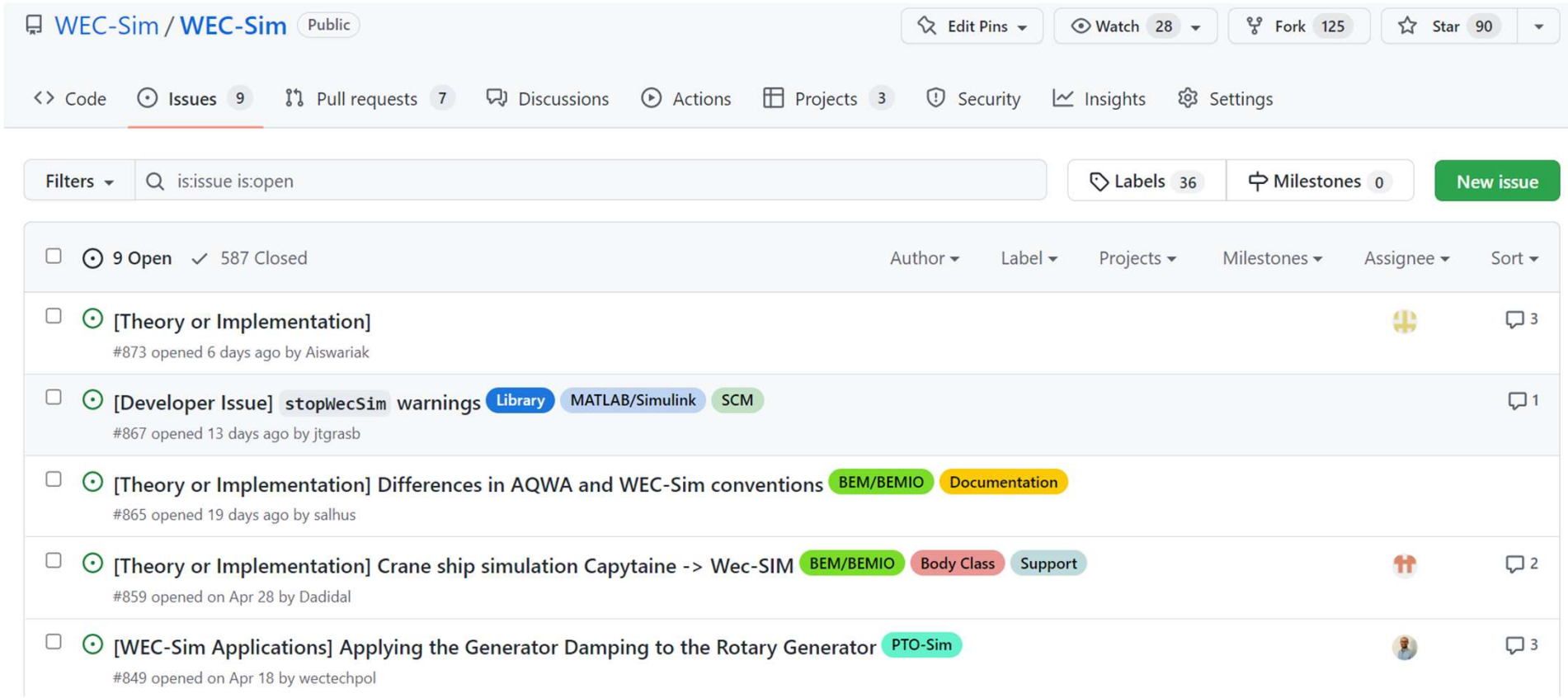
http://wec-sim.github.io/WEC-Sim/





Online Forum

https://github.com/WEC-Sim/WEC-Sim/issues



Theory & Workflow

What is WEC-Sim?

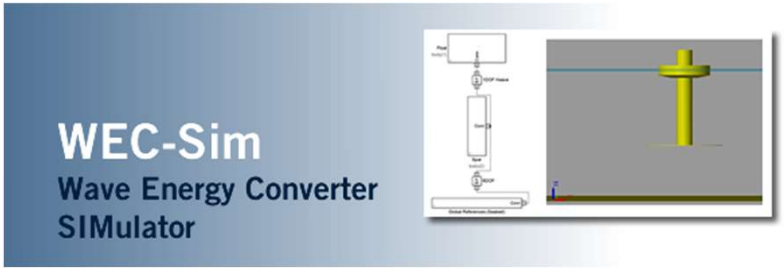
WEC-Sim (Wave Energy Converter Simulator)

- Simulates wave energy converter dynamics in operational waves
- Time-domain rigid body equation of motion solver based on Cummins' formulation
- Open source software developed in MATLAB/SIMULINK
  - Available at https://github.com/WEC-Sim/WEC-Sim
- Joint NREL/Sandia project funded by the US Department of Energy
- First Release: v1.0 in June 2014
- Current Release: v5.0.1 in Sept 2022

U.S. DEPARTMENT OF ENERGY

Sandia National Laboratories

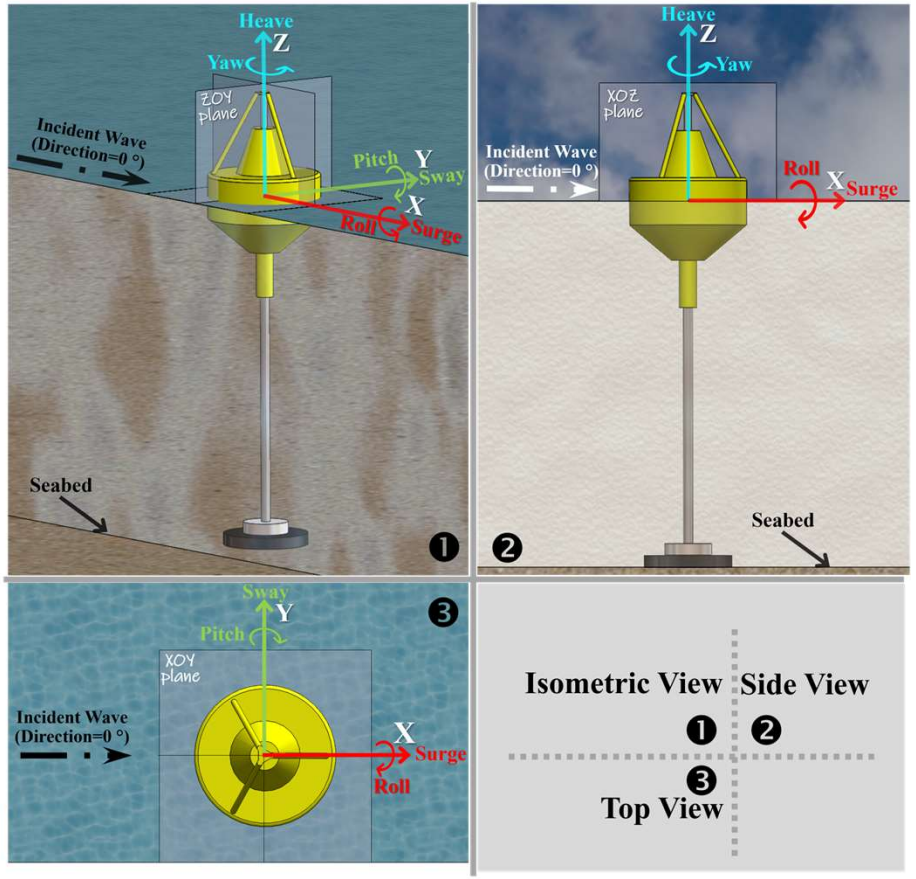
NREL
NATIONAL RENEWABLE ENERGY LABORATORY



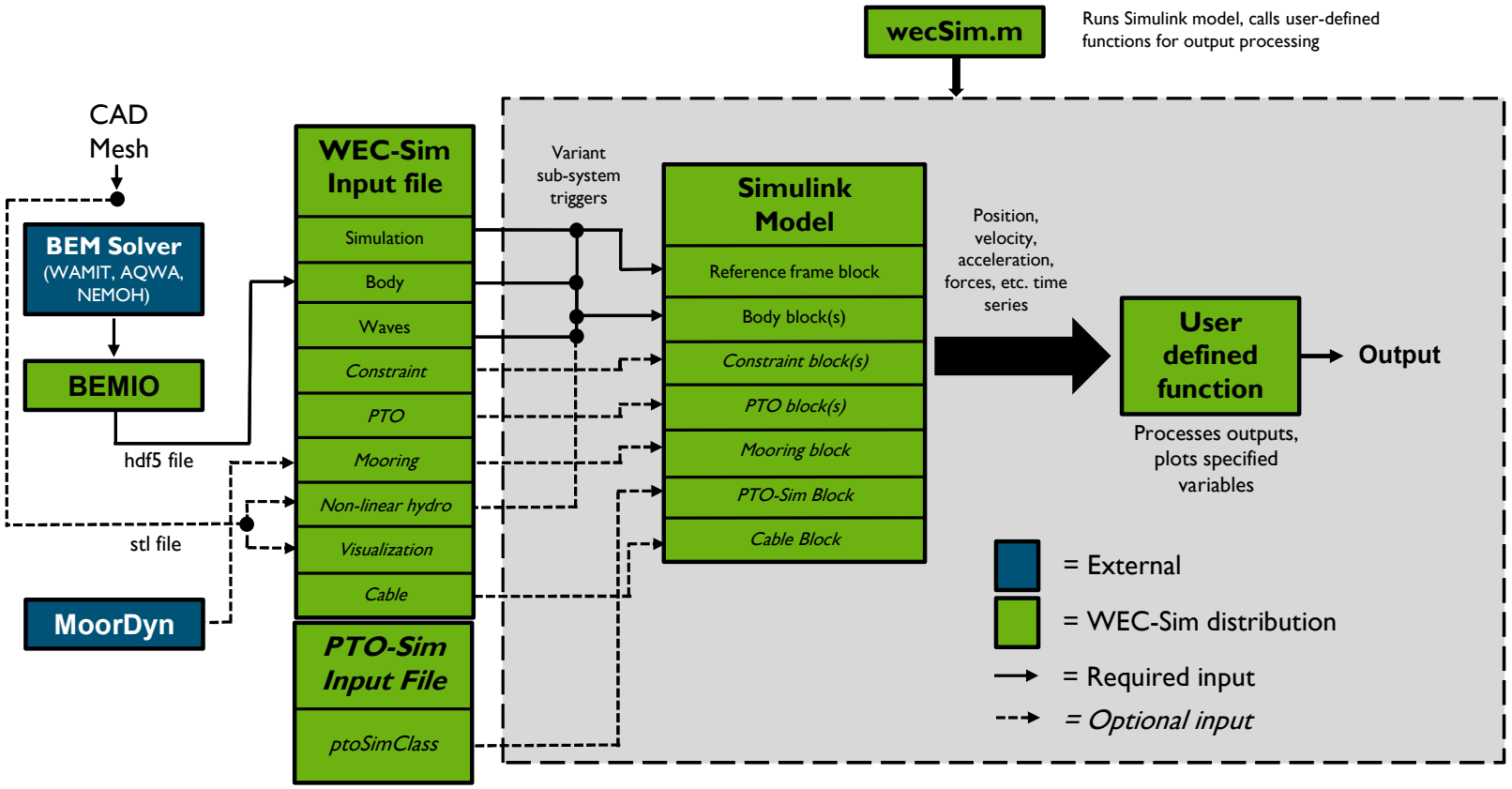
Coordinate System

- X-Axis is in the direction of wave propagation if the wave heading angle is equal to zero (following the coordinate system definition in WAMIT).
- Z-Axis is in the vertical upwards direction from a zero at the still water level, and the Y-Axis direction is defined by the right-hand rule.
- Position is described in a 6-element vector X. This convention is maintained for velocities, forces, etc.

Index	1	2	3	4	5	6
Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

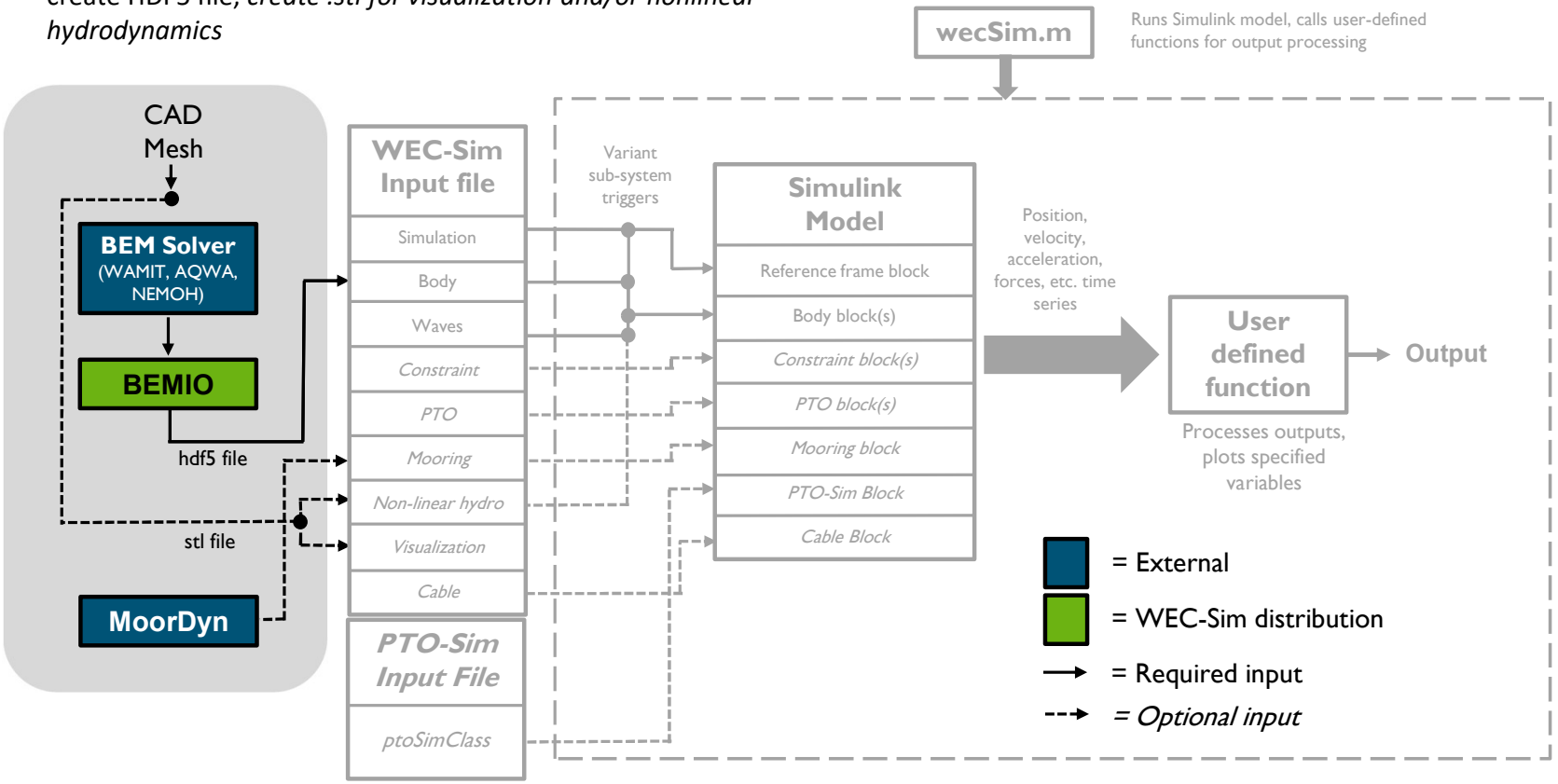


WEC-Sim Workflow



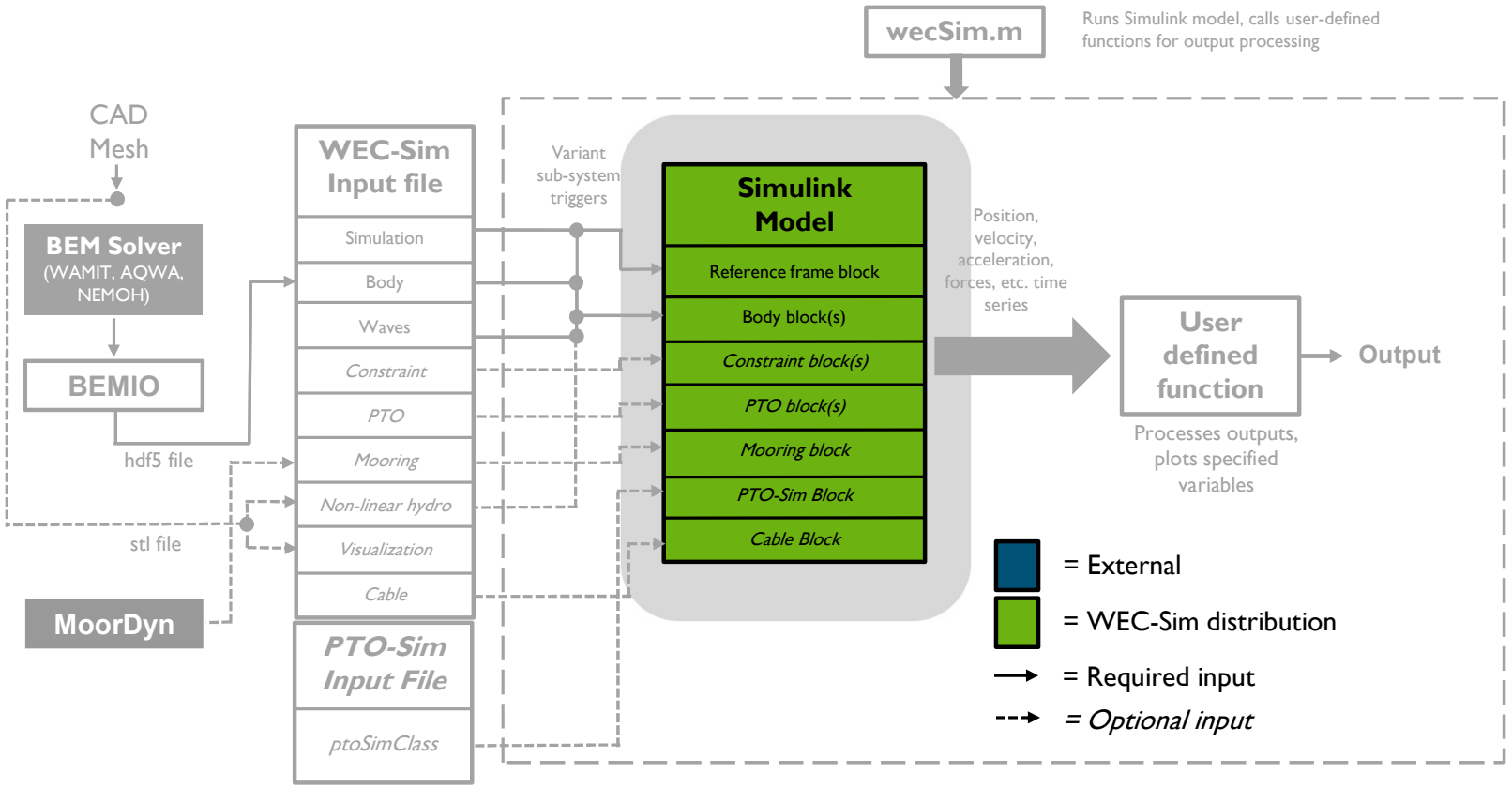
WEC-Sim Workflow

1). Generate 3-D mesh, calculate hydrodynamic coefficients, create HDF5 file, *create .stl for visualization and/or nonlinear hydrodynamics*



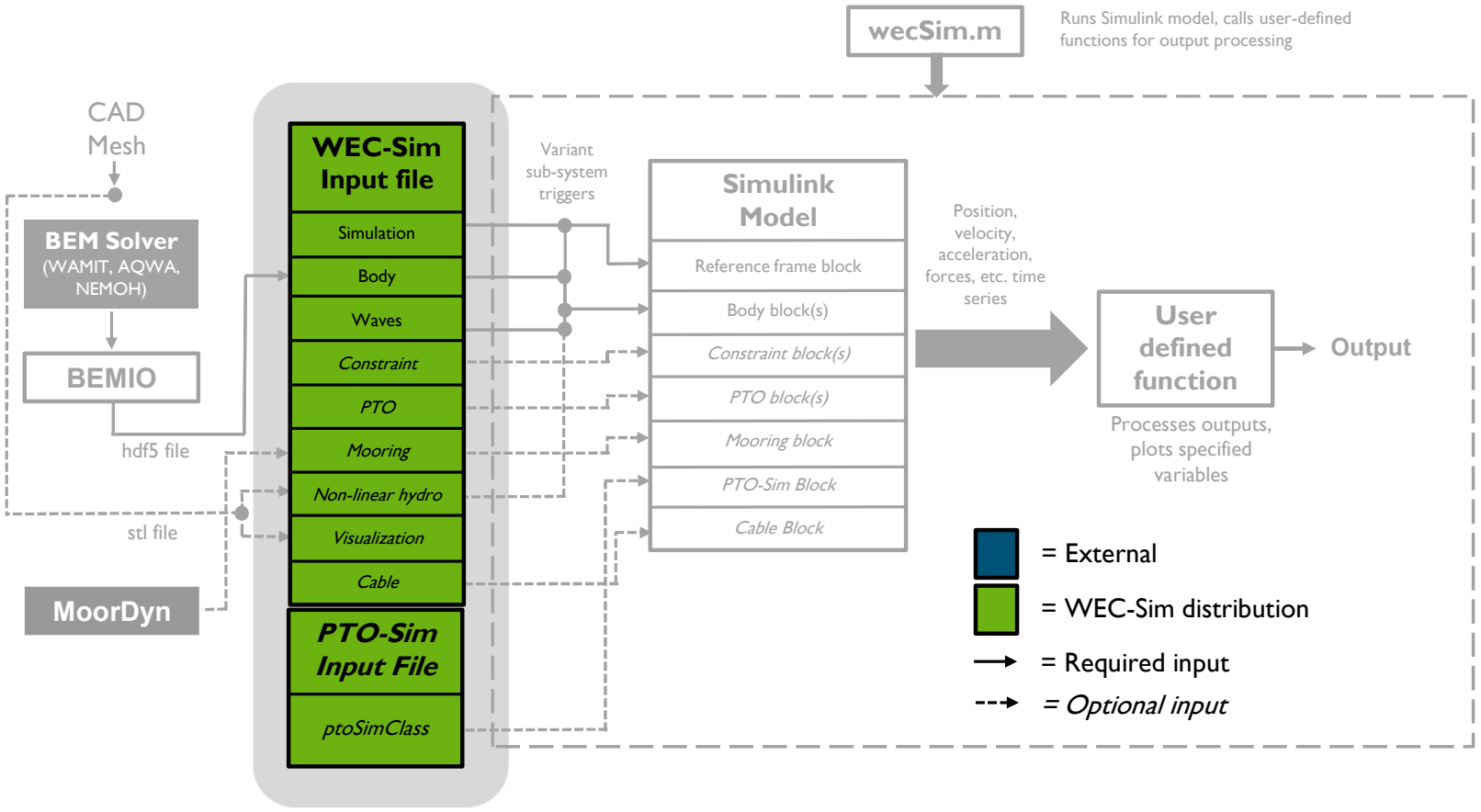
WEC-Sim Workflow

2). Build WEC-Sim model in Simulink



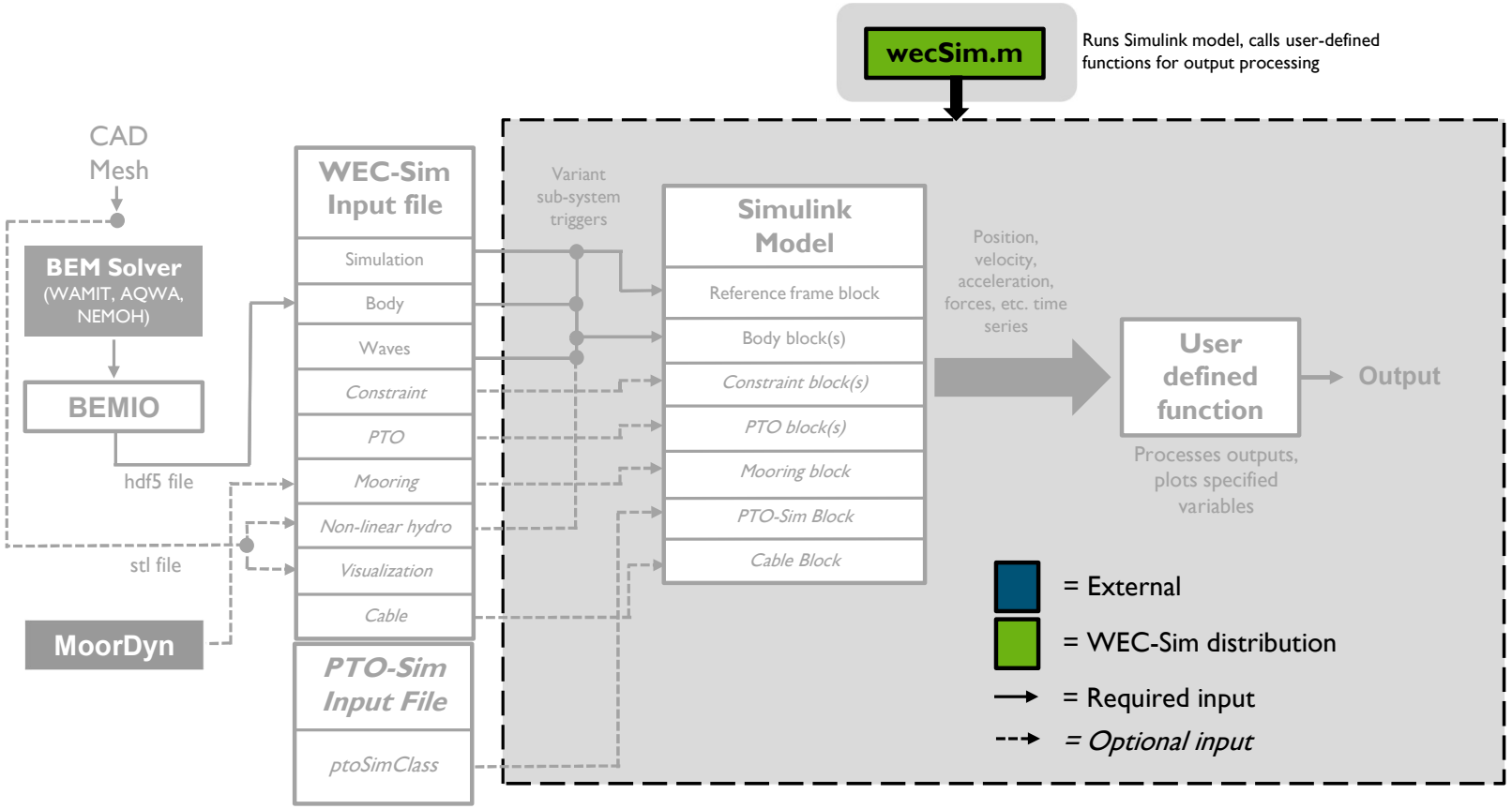
WEC-Sim Workflow

3). Write WEC-Sim input file



WEC-Sim Workflow

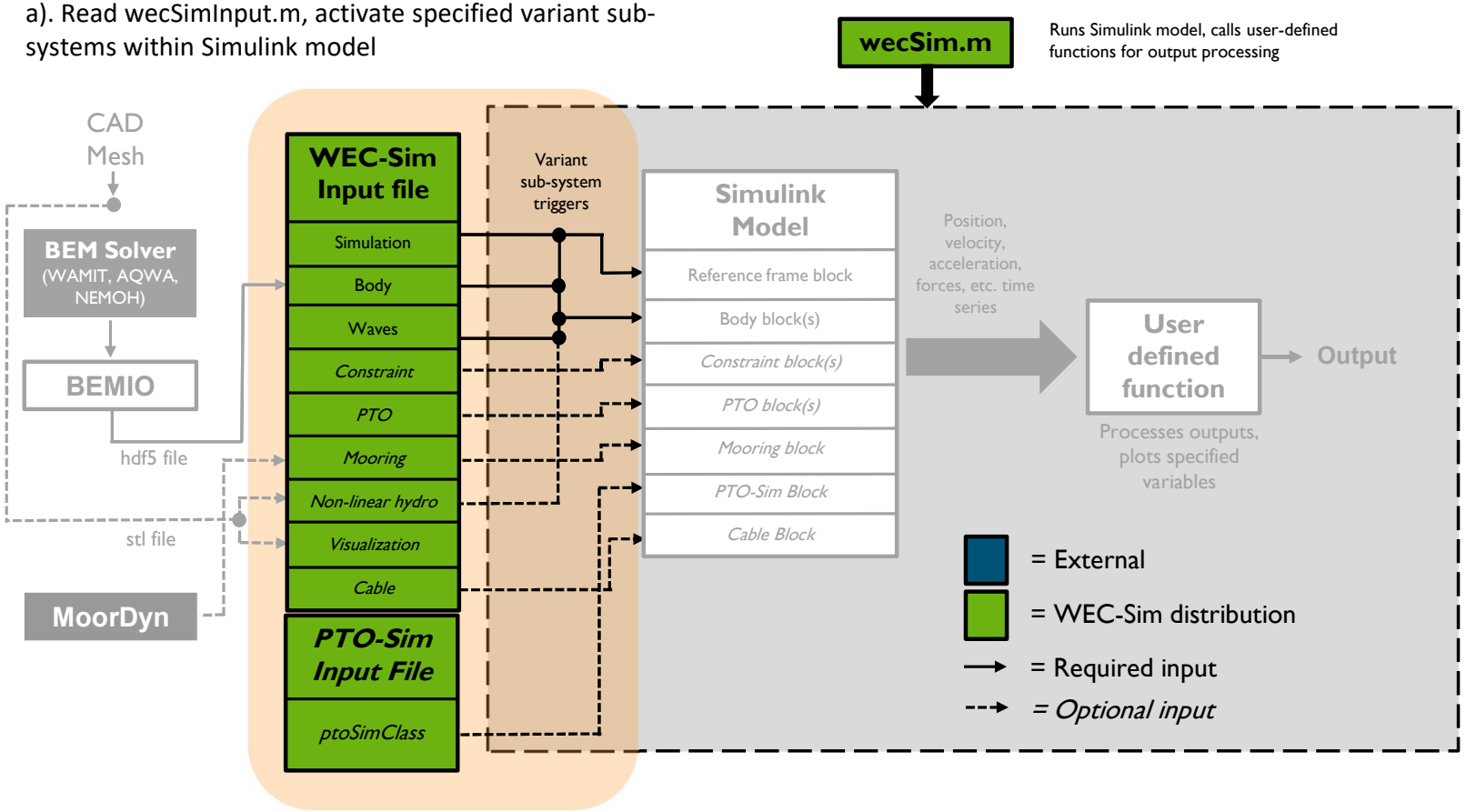
4). Execute wecSim.m



WEC-Sim Workflow

4). Execute wecSim.m

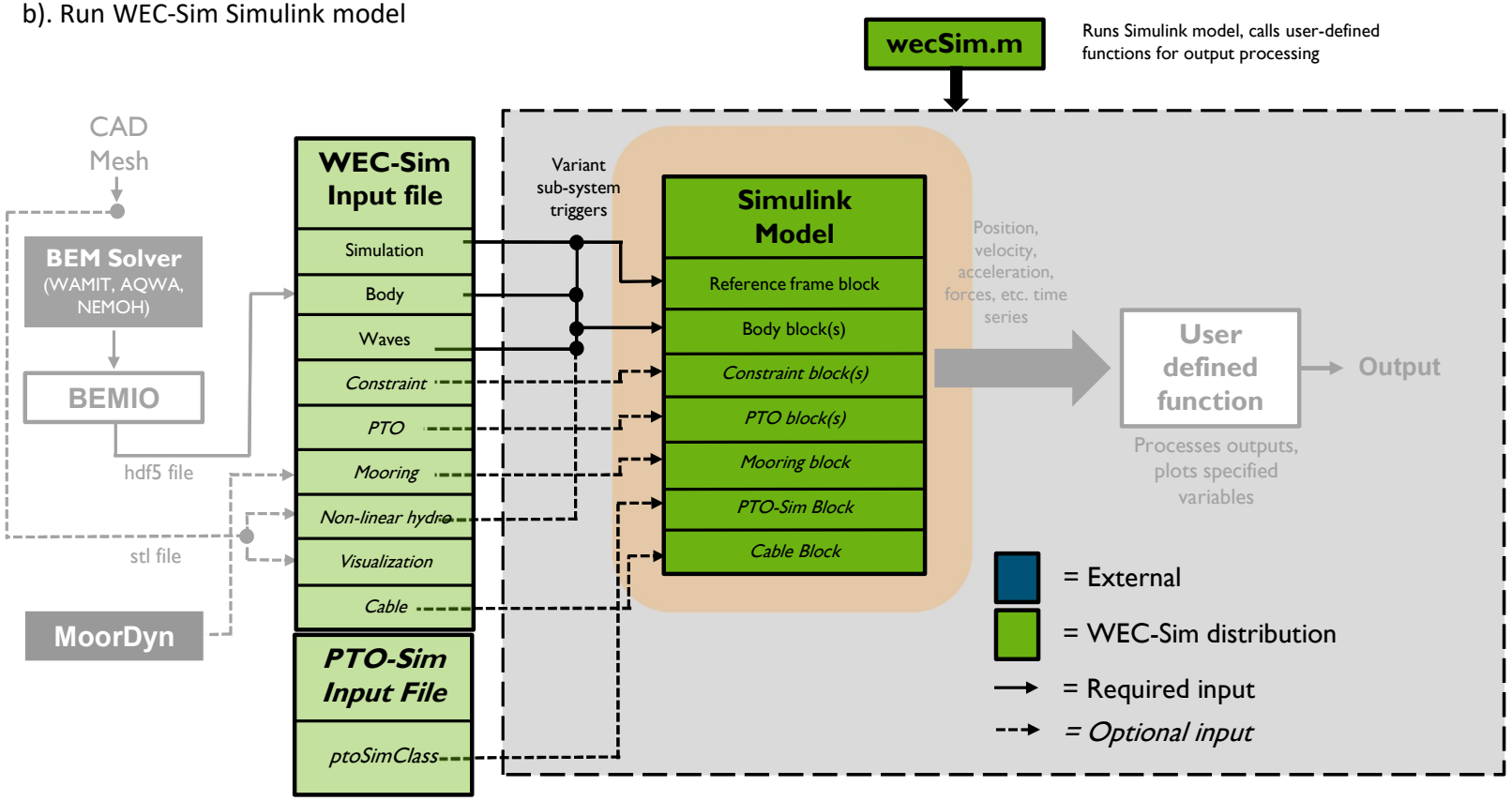
a). Read wecSimInput.m, activate specified variant sub-systems within Simulink model



WEC-Sim Workflow

4). Execute wecSim.m

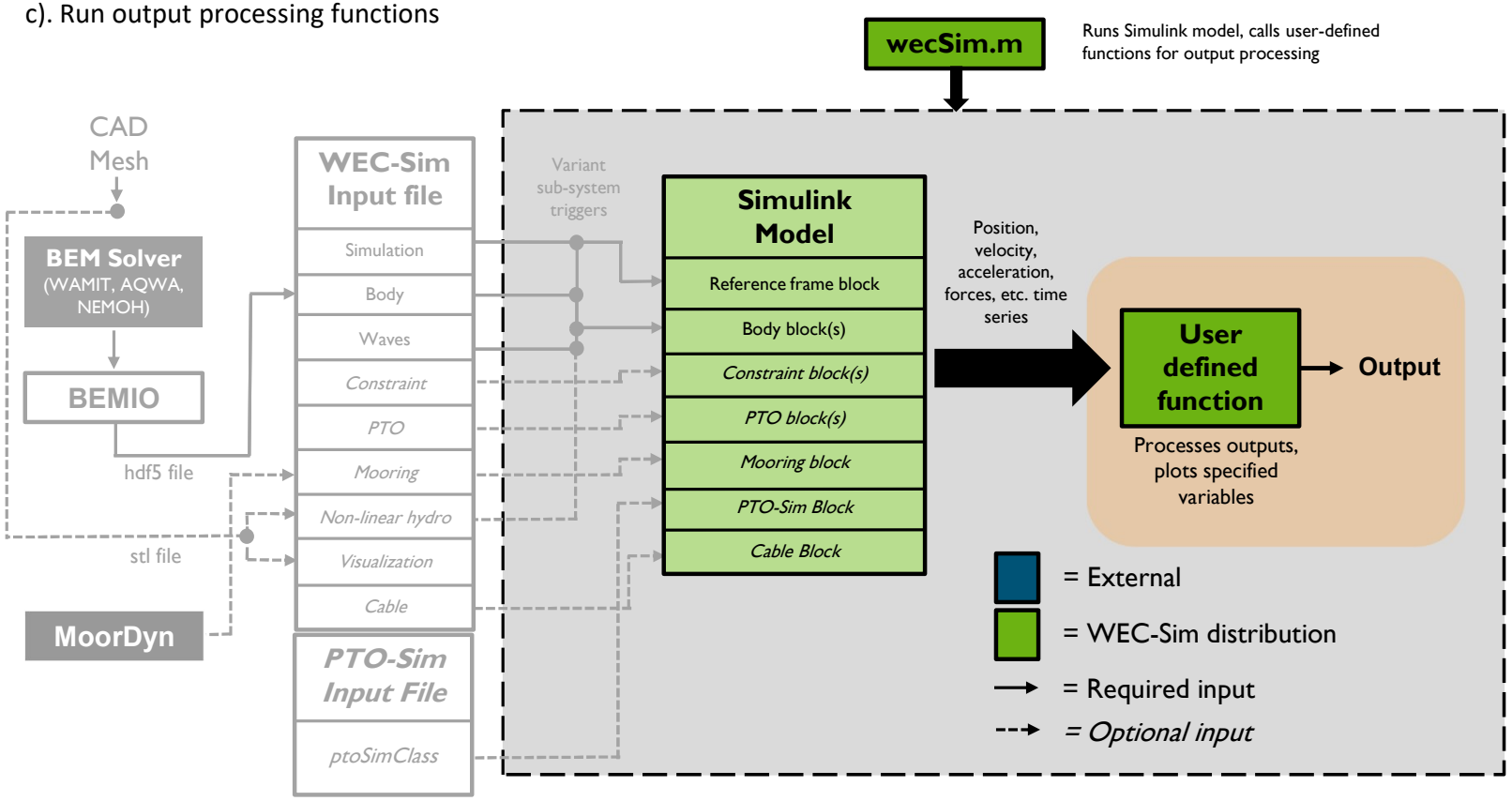
b). Run WEC-Sim Simulink model



WEC-Sim Workflow

4). Execute wecSim.m

c). Run output processing functions



General Equations of Motion

- Dynamics simulated by solving the time-domain equation of motion (Cummins, 1962)

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_{v}(t) + F_{PTO}(t) + F_{m}(t) + F_{nh}(t)$$

Hydrostatic restoring force

Wave excitation force (from BEM simulations)

Radiation forces of added mass and wave damping (from BEM simulations)

Viscous force

Power take-off force

Mooring force

Nonlinear hydrodynamic force

- Excitation and radiation forces are determined from hydrodynamic coefficients calculated from Boundary Element Method (BEM)

Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Static Mass

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- Static mass, or dry mass, is the mass (kg) (1-element) and inertia (3-element)* (kg-m2) of the dry WEC body.
  - In WEC-Sim v5.0.1 there is also the option to define a 3-element product of inertia.
- Specified for each body* in the wecSimInputFile.m as part of the bodyClass definition
- See Training Materials – Body Implementation for details
  - Usually: see Advanced Features → Body Features for special cases

```
%% Body Data
% Float
body(1) = bodyClass('hydroData/rm3.h5');
body(1).geometryFile = 'geometry/float.stl';
body(1).mass = 'equilibrium';
body(1).inertia = [20907301 21306090.66 37085481.11];

% Spar/Plate
body(2) = bodyClass('hydroData/rm3.h5');
body(2).geometryFile = 'geometry/plate.stl';
body(2).mass = 'equilibrium';
body(2).inertia = [94419614.57 94407091.24 28542224.82];
```

*The definition of body mass and inertia properties in the wecSimInputFile for the RM3 example. In this special 'equilibrium' case, the mass is set equal to the mass of the displaced volume of water, defined in the *.h5 file.*

Hydrostatic Forces

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- Hydrostatic restoring force is calculated as the product of a hydrostatic stiffness matrix and a vector of displacement*

$$F_{HS}(t) = K_{hs}X(t) = \begin{bmatrix} K_{1,1} & K_{1,2} & K_{1,3} & K_{1,4} & K_{1,5} & K_{1,6} \\ K_{2,1} & K_{2,2} & K_{2,3} & K_{2,4} & K_{2,5} & K_{2,6} \\ K_{3,1} & K_{3,2} & K_{3,3} & K_{3,4} & K_{3,5} & K_{3,6} \\ K_{4,1} & K_{4,2} & K_{4,3} & K_{4,4} & K_{4,5} & K_{4,6} \\ K_{5,1} & K_{5,2} & K_{5,3} & K_{5,4} & K_{5,5} & K_{5,6} \\ K_{6,1} & K_{6,2} & K_{6,3} & K_{6,4} & K_{6,5} & K_{6,6} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \end{bmatrix} = \begin{bmatrix} F_{HS,1}(t) \\ F_{HS,2}(t) \\ F_{HS,3}(t) \\ F_{HS,4}(t) \\ F_{HS,5}(t) \\ F_{HS,6}(t) \end{bmatrix}$$

- Elements of K_{HS} are defined for each body in its *.h5 file with BEM output information, specifying the *.h5 file is all that is needed in wecSimInputFile
- *By default: see Advanced Features → Non-linear hydrodynamics for alternative calculation method

Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Wave Excitation Forces

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- The BEM result provides complex excitation coefficients $f_{ext}(\omega, \theta)$. For a single frequency regular wave of height H, frequency ω, direction θ, ramp function $R_f(t)$, and the real component $\Re$:

$$F_{ext}(t) = \Re\left[R_f(t)\frac{H}{2}f_{ext}(\omega,\theta)e^{i\omega t}\right] = R_f(t)\frac{H}{2}\left[\Re\{f_{ext}(\omega,\theta)\}\cos\omega t - \Im\{f_{ext}(\omega,\theta)\}\sin\omega t\right]$$

- For j frequencies with amplitude spectral density S at phase φ:

$$F_{ext}(t) = \Re\left[R_f(t)\sum_{j=1}^{N} f_{ex}\left(\omega_j,\theta\right)e^{i(\omega_j t+\varphi_j)}\sqrt{2S\left(\omega_j\right)d\omega_j}\right]$$

Position X	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Wave Excitation Forces

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- For a wave defined as a time-series, the convolution of wave elevation $\eta(t)$ and the excitation impulse response function f_e calculated from f_{ext} gives an equivalent results:

$$F_{ext}(t) = R_f \int_{-\infty}^{\infty} f_e(t-\tau)\eta(\tau)d\tau$$

- The wave excitation coefficients are read from the *.h5 file.

- R_f is a ramp function that gradually increases the wave excitation from zero to the full value over a defined time period to help with simulation stability.

Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Radiation Forces

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- The BEM result provides complex frequency dependent radiation coefficients for added mass A and wave damping B.
- For a single frequency regular wave of height H and frequency ω:

$$F_{rad}(t) = -A(\omega)\ddot{X}(t) - B(\omega)\dot{X}(\mathrm{t})$$

- For a wave of multiple frequencies, the infinite frequency added mass A_∞ is used with the radiation impulse response* function K_r calculated from B:

$$F_{rad}(t) = -A_\infty\ddot{X} - \int_0^t K_r(t-\tau)\dot{X}(\tau)d\tau$$

- ***WEC-Sim can also approximate this integral via state-space approximation, see Theory → Numerical Methods → State Space**

Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Viscous Forces

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- Linear and quadratic viscous forces are calculated from coefficients and parameters provided in the wecSimInputFile.m.

$$F_{v,quad}(t) = -\frac{1}{2}\rho A C_d \dot{X}(t)\left|\dot{X}(t)\right|$$

$$F_{v,linear}(t) = -C_v \dot{X}(t)$$

$$F_v(t) = F_{v,linear}(t) + F_{v,quad}(t)$$

```
%% Body Data
% Float
body(1) = bodyClass('hydroData/rm3.h5');
body(1).geometryFile = 'geometry/float.stl';
body(1).mass = 'equilibrium';
body(1).inertia = [20907301 21306090.66 37085481.11];
body(1).initial.angle = pi/12;
body(1).linearDamping = zeros(6);
body(1).linearDamping(3,3) = 10;
body(1).quadDrag.cd = [0 0 1.3 0 0 0];
body(1).quadDrag.area = [0 0 314.16 0 0 0];
```

The definition of linear and quadratic damping parameters for the heave mode in the *wecSimInputFile.m* for the RM3 example.

- ***See also Advanced Features → Morison Elements for an alternative means of specifying quadratic damping and augment added mass**

Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Power Take Off (PTO)

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- Power take-off forces describe actuations between WEC bodies or WEC body and the fixed frame. If using the provided library blocks*, the PTO parameters are defined in the wecSimInputFile.m.
- $F_{PTO}(t) = -K_{PTO}X_{rel} - C_{PTO}\dot{X}_{rel}$
- X_{rel} is the relative motion between the nodes connected by the PTO.
- *PTOs can be modeled in a variety of ways and can leverage the full suite of Simulink and Simscape components and control tools. See also Advanced Features → PTO-Sim

```
% Translational PTO
pto(1) = ptoClass('PTO1');                  % Initialize PTO Class for PTO1
pto(1).stiffness = 0;                       % PTO Stiffness [N/m]
pto(1).damping = 1200000;                   % PTO Damping [N/(m/s)]
pto(1).location = [0 0 0];                  % PTO Location [m]
```

Specification of a single DOF translational PTO in the *wecSimInputFile.m* for the RM3 Example.

Mooring forces

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

- WEC-Sim supports a linear mooring matrix and MoorDyn (see Advanced Features → MoorDyn).
- Linear mooring matrix forces are calculated

$$F_m(t) = -K_{moor}X_{rel} - C_{moor}\dot{X}_{rel} + F_{preTension}$$

- X_{rel} is the motion of the components of the follower-side connections.

```
% Mooring Matrix
mooring(1) = mooringClass('Mooring1');    % initialize mooring
mooring(1).matrix.stiffness = zeros(6,6);
mooring(1).matrix.damping = zeros(6,6);
mooring(1).matrix.stiffness(3,3) = 1000;  % N/m applied to resist heave displacement
mooring(1).matrix.damping(3,3) = 250;     % N-m/s applied to resist heave velocity
mooring(1).matrix.preTension = [0 0 100 0 0 0]; % N pretension applied in heave
```

Specification of a mooring matrix in the *wecSimInputFile.m*

Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Non-linear hydrodynamic forces

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_{v}(t) + F_{PTO}(t) + F_{m}(t) + \boldsymbol{F_{nh}(t)}$$

- Non-linear hydrodynamic forces include non-linear Froude-Krylov and non-linear buoyancy forces that are calculated based on panel-method integration over body geometry defined in supplemental '*.stl' files from time-resolved undisturbed wave fields and body displacements.
- ***See also Advanced Features → Non-linear Hydrodynamics**

Position *X*	x (surge)	y (sway)	z (heave)	rx (roll)	ry (pitch)	rz (yaw)

Summary of equations

$$m\ddot{x}(t) = F_{hs}(t) + F_{ext}(t) + F_{rad}(t) + F_v(t) + F_{PTO}(t) + F_m(t) + F_{nh}(t)$$

Forcing Term	Condition	Theory
Radiation (F_{rad})	Regular Waves Irregular Waves	Sinusoidal Steady-State Response $F_{rad} = -A(\omega)\ddot{X} - B(\omega)\dot{X}$ Cummins Equation (Convolution Integral) $F_{rad} = -A_\infty \ddot{X} - \int_0^t K_r(t-\tau)\dot{X}(\tau)d\tau$ State Space Representation $\dot{X}_r(t) = A_r X_r(t) + B_r u(t); \int_0^t K_r(t-\tau)u(\tau)d\tau \approx C_r X_r(t) + D_r u(t)$
Wave Excitation (F_{ext})	Regular Waves Irregular Waves	Sinusoidal Steady-State Response $F_{exc}(t) = \Re\left[R_f(t)\frac{H}{2}F_{exc}(\omega,\theta)e^{i\omega t}\right]$ Wave Spectrum (e.g., JS; BS; PM) $F_{exc}(t) = \Re\left[R_f(t)\sum_{j=1}^{N} F_{exc}(\omega_j,\theta)\, e^{i(\omega_j t+\phi_j)}\sqrt{2S(\omega_j)d\omega_j}\right]$ Wave Elevation (Convolution Integral) $F_{exc}(t) = \int_{-\infty}^{\infty} f_e(t-\tau)\eta(\tau)d\tau$
PTO (F_{pto})		Linear Spring-Damper $P_{PTO} = C_{PTO}\dot{X}_{rel}^2$ $F_{PTO} = -K_{PTO}X_{rel} - C_{PTO}\dot{X}_{rel}$ Hydraulic PTO Mechanical PTO $P_{PTO} = -F_{PTO}\dot{X}_{rel}$ $F_{PTO} = f(X_{rel}, \dot{X}_{rel}, \ddot{X}_{rel}, \dots)$
Mooring (F_m)		Linear Mooring Matrix (i.e., stiffness, damping and pretension) Lumped-Mass Mooring Dynamics Model (MoorDyn)
Additional Added-Mass & Damping (F_v & F_{ME})		Linear & Quadratic Damping Forces $F_v = -C_v\dot{X} - C_d\rho A_d/2\,\dot{X}\,\lvert\dot{X}\rvert$ Morison Elements $F_{me} = \rho\forall\dot{v} + \rho\forall C_a(\dot{v} - \ddot{X}) + C_d\rho A_d/2\,(v - \dot{X})\,\lvert v - \dot{X}\rvert$
Nonlinear Hydrodynamic Forces (F_{nh})	Nonlinear Hydrodynamics	The additional term accounts for the difference between the nonlinear and linear hydrodynamic forces (buoyancy and the Froude-Krylov force components).

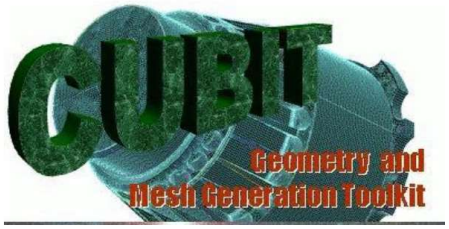
Run a WEC-Sim Simulation

1. Before funning: (Any Order)
   - Get a *.h5 file → Defines the hydrodynamic coefficients
   - Build a Simulink *.slx model → Describes device layout
   - Write wecSimInputFile.m → Defines dynamic parameters

- ***See Advanced Features → Non-linear hydrodynamics and MoorDyn for additional optional inputs**

Run a WEC-Sim Simulation

1. Before funning: (Any Order)
   - **Get a *.h5 file → Defines the hydrodynamic coefficients**
   - Build a Simulink *.slx model → Describes device layout
   - Write wecSimInputFile.m → Defines dynamic parameters



- From supported BEM codes: WAMIT, NEMOH, Capytaine, and AQWA
  - **See also Advanced Features → BEMIO**
- This BEM code will require mesh(es) to run!
- See User Manual → Workflow → Step 2

- ***See Advanced Features → Non-linear hydrodynamics and MoorDyn for additional optional inputs**

Run a WEC-Sim Simulation

2. Run WEC-Sim
   - >> wecSim or run from Simulink GUI (see Advanced Features →Running from Simulink)
   - WEC-Sim will then:

a) Clear existing variables that might conflict with those about to be loaded

b) Run *initializeWecSim.m*

Running initializeWecSim.m

a)	Reads *wecSimInputFile.m* (exactly how depends on how wecSim is being called)	**Line 55-78**
b)	Setup all objects defined in *.slx file (e.g. , constraints, ptos, moorings)	**Line 88-116**
c)	For each body, load nondimensional hydrodynamic coefficients	**Line 150-162**
d)	Setup simulation and waves, calculating wave time series	**Line 219-239**
e)	For each body, convert nondimensional hydro. Coeffs. to dimensional forces	**Line 249-256**

Running initializeWecSim.m

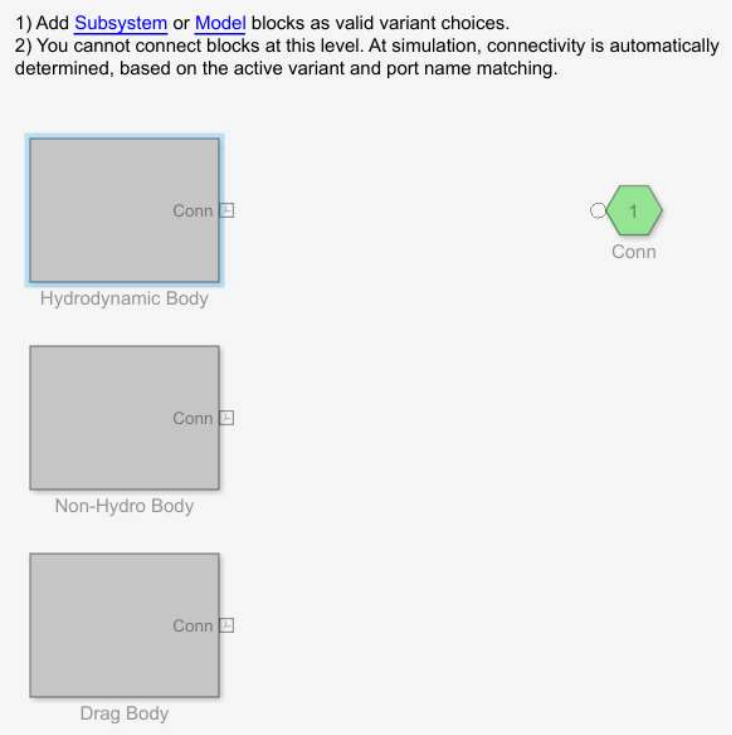
f) Diagnostic checks **Line 276-344**

g) Define variant sub-systems **Line 346-420**

A variant sub-system is Simulink block type that allows the same top-level model to follow several different execution pathways depending on the definition of a particular variable.

A variant sub-system of the Body/Rigid Body. Depending on the specification in **bodyClass**, the correct block will be connected based on variable assignment in lines 397-402.

This ensures the same *.slx model to run without modification, different sea-states and simulation parameters.



Running WecSim.m

h) After *initializeWecSim.m* finishes Run the Simulink simulation

i) Post process

- Collate outputs into the **responseClass** (default variable name = 'output')
- Save results
- Run *userDefinedFunctions.m*

***The execution pathway differs for the "Run from Simulink" options, but most function calls still originate for *initializeWecSim.m*. See Advanced Features → Running from Simulink**

Notes/Warnings/Errors

```
Error using assert
Input body.quadDrag.cd should be 1x6

Error in bodyClass/checkInputs (line 149)
            assert(isequal(size(obj.quadDrag.cd)==[1,6],[1,1]),'

Error in initializeWecSim (line 127)
    body(ii).checkInputs(simu.explorer);

Error in run (line 91)
evalin('caller', strcat(script, ';'));

Error in wecSim (line 44)
run('initializeWecSim');
```

- Many common errors will have informative error messages that illustrate the mistake, but not necessarily where the error enters in the wecSimInputFile.m.
- **In this case, the quadratic drag vector '*body(1).quadDrag.cd*' was the incorrect length.**

Further Reading

- Full documentation: http://wec-sim.github.io/WEC-Sim/master/index.html
- Specifically see: http://wec-sim.github.io/WECSim/master/user/advanced_features.html#advanced-features
- The headers of class object code populate API documentation
  - https://wec-sim.github.io/WEC-Sim/master/user/api.html
- Training Slides:
  - https://wec-sim.github.io/WEC-Sim/master/user/webinars.html#online-training-course

Installation

Installation Steps

- **Check MATLAB requirements**
- **Download WEC-Sim**
- **Add WEC-Sim to MATLAB path**
- **Verify WEC-Sim path**
- **Add WEC-Sim library to Simulink**
- **Test WEC-Sim installation**

Install MATLAB and Download WEC-Sim

Check MATLAB Requirements

Verify that the MATLAB required toolboxes are installed by typing `ver` in the MATLAB Command Window:

```
>> ver
-----------------------------------------------------------------------------------------
MATLAB Version: 9.9.0.1592791 (R2020b)
-----------------------------------------------------------------------------------------
MATLAB                                                Version 9.9         (R2020b)
Simulink                                              Version 10.2        (R2020b)
Simscape                                              Version 5.0         (R2020b)
Simscape Multibody                                    Version 7.2         (R2020b)
```

Required packages include:

MATLAB, Simulink, Simscape, Simscape Multibody

Versions supported:

2020b and later

Download WEC-Sim

A. Install WEC-Sim using git:

```
>> git lfs install
>> git clone https://github.com/WEC-Sim/WEC-Sim
```

B. Install WEC-Sim from the latest tagged WEC-Sim Release.

If you want to contribute to WEC-Sim development, go to this link:
https://wec-sim.github.io/WEC-Sim/master/developer/getting_started.html#dev-getting-started

Install WEC-Sim

1. Add WEC-Sim to MATLAB Path

Option 1. Automatically add the WEC-Sim source on MATLAB startup.

Open `$WECSIM/source/addWecSimSource.m` and copy contents to a new file called `startup.m`. Set the WEC-Sim path to the local `$WECSIM/source` directory, e.g. `wecSimSource = 'C:/User/Documents/GitHub/WEC-Sim/source`. Save `startup.m` to the MATLAB Startup Folder. Restart MATLAB, and the `$WECSIM/source` directory will automatically be added to the MATLAB path.

```
% This script adds the WEC-Sim source to the MATLAB path.

% Define WEC-Sim source and add to MATLAB path
wecSimSource = fullfile(pwd,'source');
addpath(genpath(wecSimSource));

% Allow opening of Simulink models saved in a newer version
set_param(0, 'ErrorIfLoadNewModel', 'off')

clear wecSimSource
```

2. Verify the path

WEC-Sim should be contained on the MATLAB path. Type “path” in the MATLAB Command Window:

```
>> path

        MATLABPATH

C:/User/Documents/GitHub/WEC-Sim/source
```

https://wec-sim.github.io/WEC-Sim/master/developer/getting_started.html#dev-getting-started

Install WEC-Sim

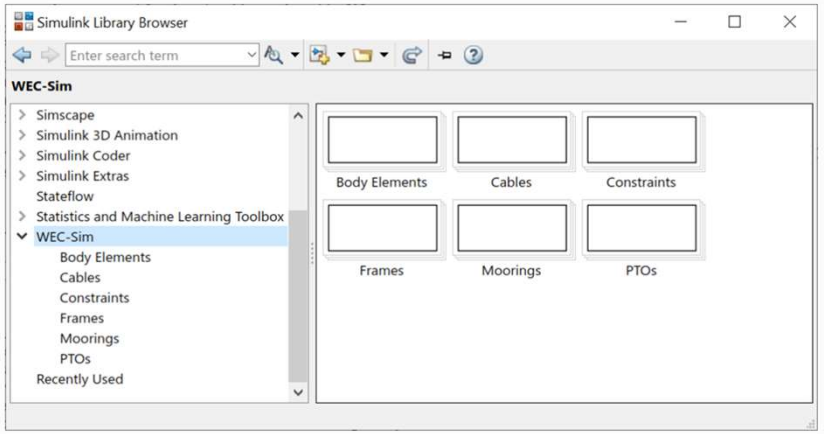
Add WEC-Sim library to Simulink

Step 3. Add WEC-Sim Library to Simulink

Open the Simulink Library Browser by typing `slLibraryBrowser` in the MATLAB Command Window:

```
>> slLibraryBrowser
```

Once the Simulink Library Browser opens, refresh the Simulink Library. The WEC-Sim Library should now be visible, as shown in the figure below. The WEC-Sim Library will now be accessible every time Simulink is opened. For more information on using and modifying library blocks refer to the Simulink Documentation.



https://wec-sim.github.io/WEC-Sim/master/developer/getting_started.html#dev-getting-started

Install WEC-Sim

WEC-Sim Installation

Go to `$WECSIM/examples/RM3/hydroData` Run `bemio.m`**Te**

Go back to `$WECSIM/examples/RM3` and type `wecSim` in the command window

This should run an example case using the Reference Model 3 (RM3) point absorber. A Mechanics Explorer window will open within the MATLAB window, and figures will be generated displaying simulation outputs.

https://wec-sim.github.io/WEC-Sim/master/developer/getting_started.html#dev-getting-started

Code Structure

WEC-Sim Directory Structure

WEC-Sim **source code** consists of:

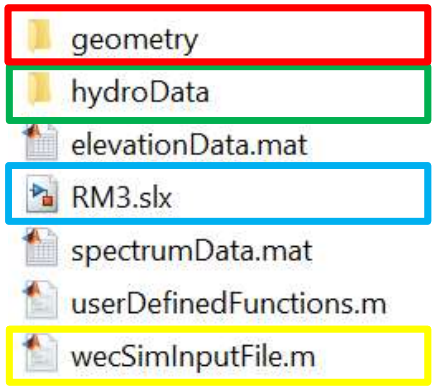
File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

*These are required inputs, but a number of optional additional inputs are possible, like custom wave spectra/ time-series.
See User Manual → Code Structure → spectrumImport

Use of Adv. Features (e.g. PTO-Sim and Moordyn) have additional requirements.
See Advanced Features

WEC-Sim **model files*** consist of:

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>



WEC-Sim Source Code

WEC-Sim Source Code

WEC-Sim **source code** consists of:

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

- Source code is included in the MATLAB path
- Can be executed from any directory
- <object>Class.m methods require the <object> to be initialized before use

WEC-Sim Executable

WEC-Sim/source/wecSim.m

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

Running ***wecSim.m***

- clears needed variables: **this will delete unsaved outputs from previous runs**
- Calls *initializeWecSim.m,* which reads *wecSimInputFile.m* and:
  - performs preprocessing calculations in each of the classes
  - selects and initializes variant subsystems in the Simulink model
- runs the time-domain simulation of the Simulink model

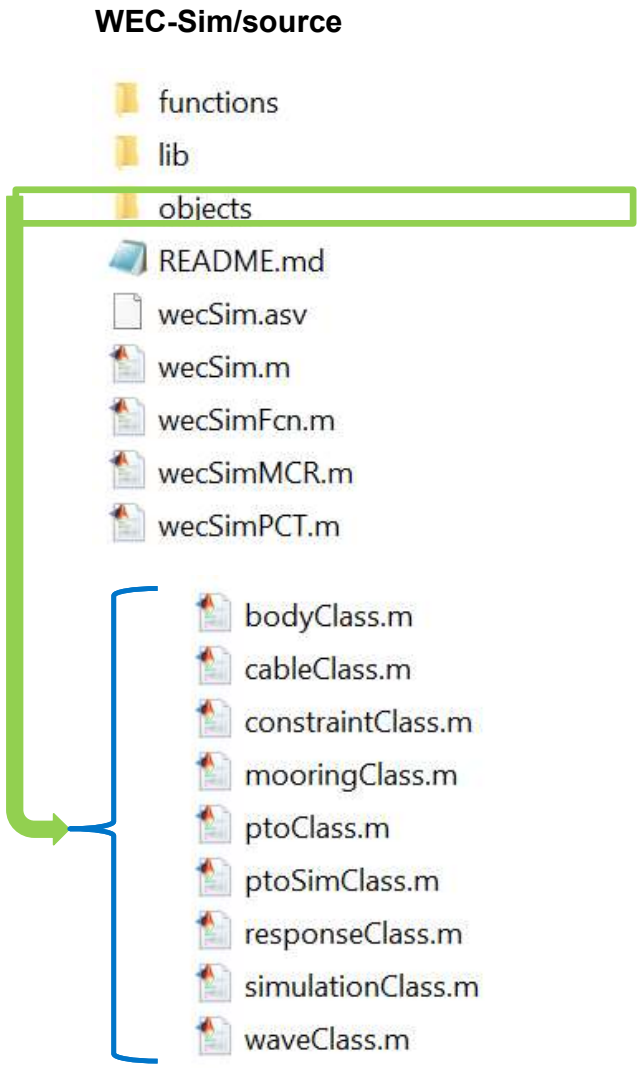
View ***wecSim.m*** and ***initializeWecSim.m*** from MATLAB Command Window
>>edit <filename>

*** See Training Materials → Theory and Workflow for detailed walkthrough**

WEC-Sim Objects

WEC-Sim/source/objects/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink



Define classes in the *wecSimInputFile.m*

The following classes create the WEC-Sim objects

- ***simulationClass.m, waveClass.m, bodyClass.m, constraintClass.m, ptoClass.m, mooringClass.m, cableClass.m***

WEC-Sim objects are required to run WEC-Sim simulations

- ***simu, waves, body(i), pto(i) OR constraint(i)***
- Additional object types can be defined if desired

View properties or open classes from MATLAB Command Window
>> doc <className>
>> open <className>

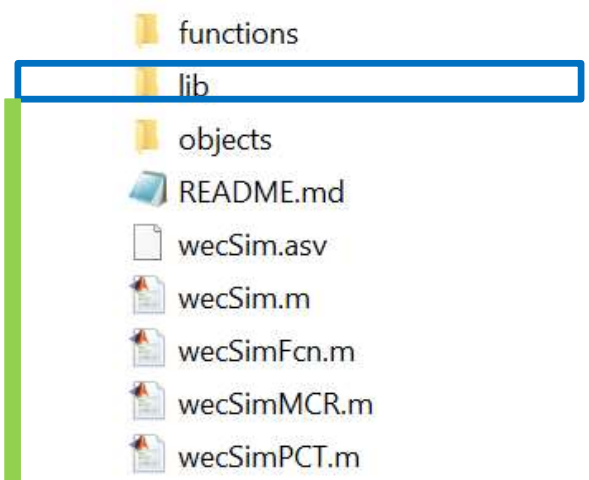
***See User Manual→Code Structure → Source Detail for more**

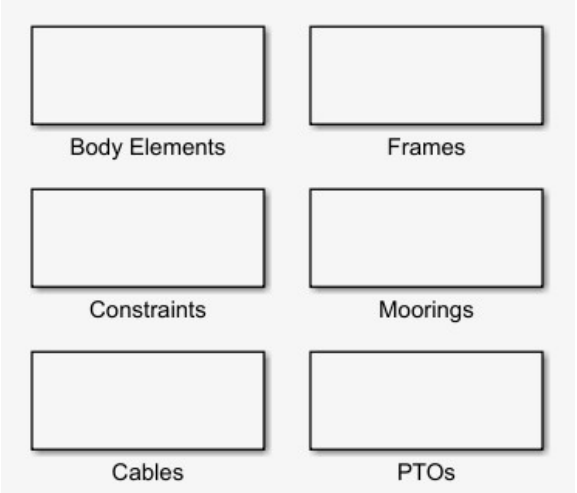
WEC-Sim Library Blocks

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim/source





WEC-Sim source code includes WEC-Sim library blocks:

- ***Body Elements, Constraints, Frames, Moorings, PTOs, Cables***

Define WEC dynamics in WEC-Sim Simulink model using WEC-Sim Library Blocks

- ***<SimulinkModelName>.slx***

View properties by double clicking on blocks, Ctrl+U to look under mask

All objects defined in the *wecSimInputFile* should also be blocks used in the Simulink model.

Simulink Mask/Model Functions

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim/source/functions/simulink

To facilitate version control, within the Simulink model, mask functions and MATLAB functions reference externally-housed functions.

Name of the MATLAB function call is the same as the external function.

Only necessary to change the external function to affect model.

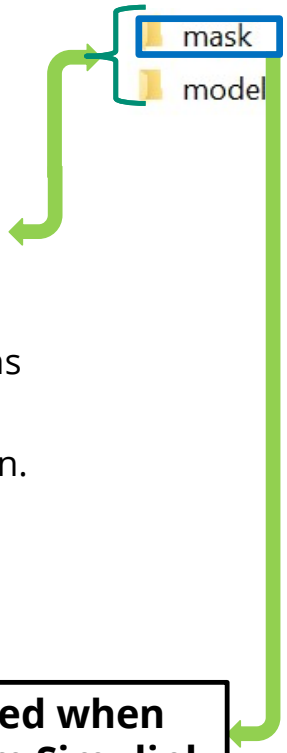

```
function dispPhase  = calcDispPhase(disp, enable, frequency, wavenumber, direction)
% initialize
dispPhase = zeros(length(frequency),length(direction));
% execute
dispPhase = calcDispPhase(disp, enable, direction, frequency, wavenumber);

end
```

Simulink Mask/Model Functions

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim/source/functions/simulink



To facilitate version control, within the Simulink model, mask functions and MATLAB functions reference externally-housed functions.

Name of the MATLAB function call is the same as the external function.

Only necessary to change the external function to affect model.

*** $/source/functions/simulink/mask functions are mostly used when running from the Simulink GUI. See Adv. Features → Run from Simulink**

WEC-Sim Model Files

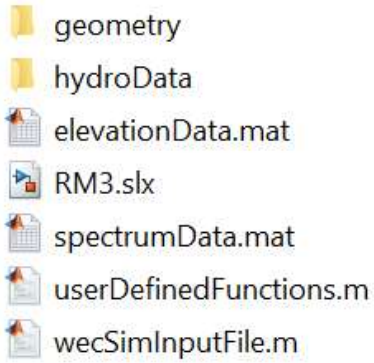
WEC-Sim Model Files

WEC-Sim **model files** consist of:

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>

- Model files are located in the case directory
- ***WEC-Sim models must be executed from the case directory***

WEC-Sim/examples/RM3



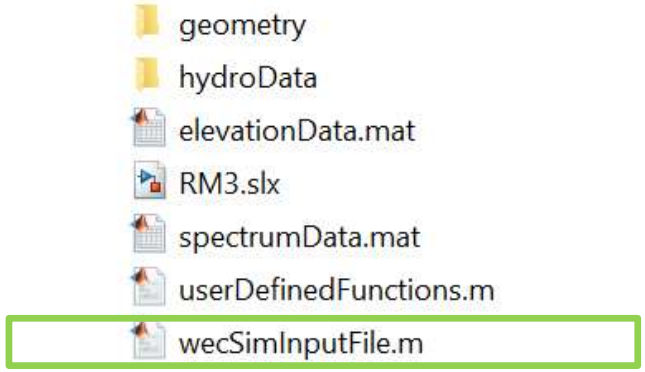
An example of a WEC-Sim case directory

WEC-Sim Input File

wecSimInputFile.m

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>

Case Directory



- Necessary by default (see Adv. Feat.→Sim. Feat.→Running from Simulink)
- Located in the case directory
- Initialize and define classes in the WEC-Sim input file
  - ***wecSimInputFile.m***
- WEC-Sim objects are required to run WEC-Sim simulations
  - ***simu, waves, body(i), pto(i)*** **OR** ***constraint(i)***

no (i): Only one per simulation

(i): More than one OK

*** Additional optional objects are also defined in the wecSimInputFile**

See User Manual→Code Structure→WEC-Sim Classes

WEC-Sim Input File

wecSimInputFile.m

- Initialize Simulation Class
- Set Properties of Simulation Class

```
%% Simulation Data
simu = simulationClass();
simu.simMechanicsFile = 'RM3.slx';
simu.mode = 'normal';
simu.explorer = 'on';
simu.startTime = 0;
simu.rampTime = 100;
simu.endTime = 400;
simu.solver = 'ode4';
simu.dt = 0.1;
```

- Initialize Wave Class
- Set Properties of Wave Class

```
%% Wave Information
% % noWaveCIC, no waves with radiation CIC
waves = waveClass('regularCIC');
waves.period = 6; % seconds
waves.height = 1; % meters
```

- Initialize Body Class Instances
- Set Properties of Body Class Instances

```
%% Body Data
% Float
body(1) = bodyClass('hydroData/rm3.h5');
body(1).geometryFile = 'geometry/float.stl';
body(1).mass = 'equilibrium';
body(1).inertia = [20907301 21306090.66 37085481.11];

% Spar/Plate
body(2) = bodyClass('hydroData/rm3.h5');
body(2).geometryFile = 'geometry/plate.stl';
body(2).mass = 'equilibrium';
body(2).inertia = [94419614.57 94407091.24 28542224.82];
```

- Initialize Constraint Class
- Set Properties of Constraint Class
- Initialize PTO Class
- Set Properties of PTO Class

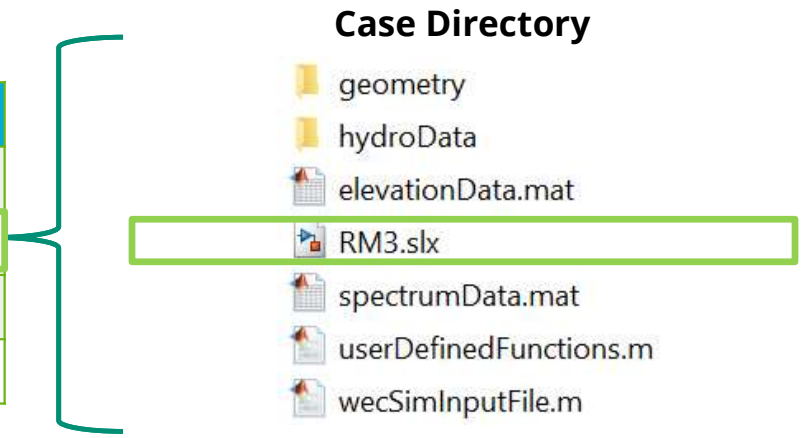
```
%% PTO and Constraint Parameters
% Floating (3DOF) Joint
constraint(1) = constraintClass('Constraint1');
constraint(1).location = [0 0 0];

% Translational PTO
pto(1) = ptoClass('PTO1');
pto(1).stiffness = 0;
pto(1).damping = 1200000;
pto(1).location = [0 0 0];
```

One can also define these parameters irectly from the Simulink GUI if desired sing the "Running From Simulink" work ow. See Adv. Features → Run from imulink

WEC-Sim Simulink File

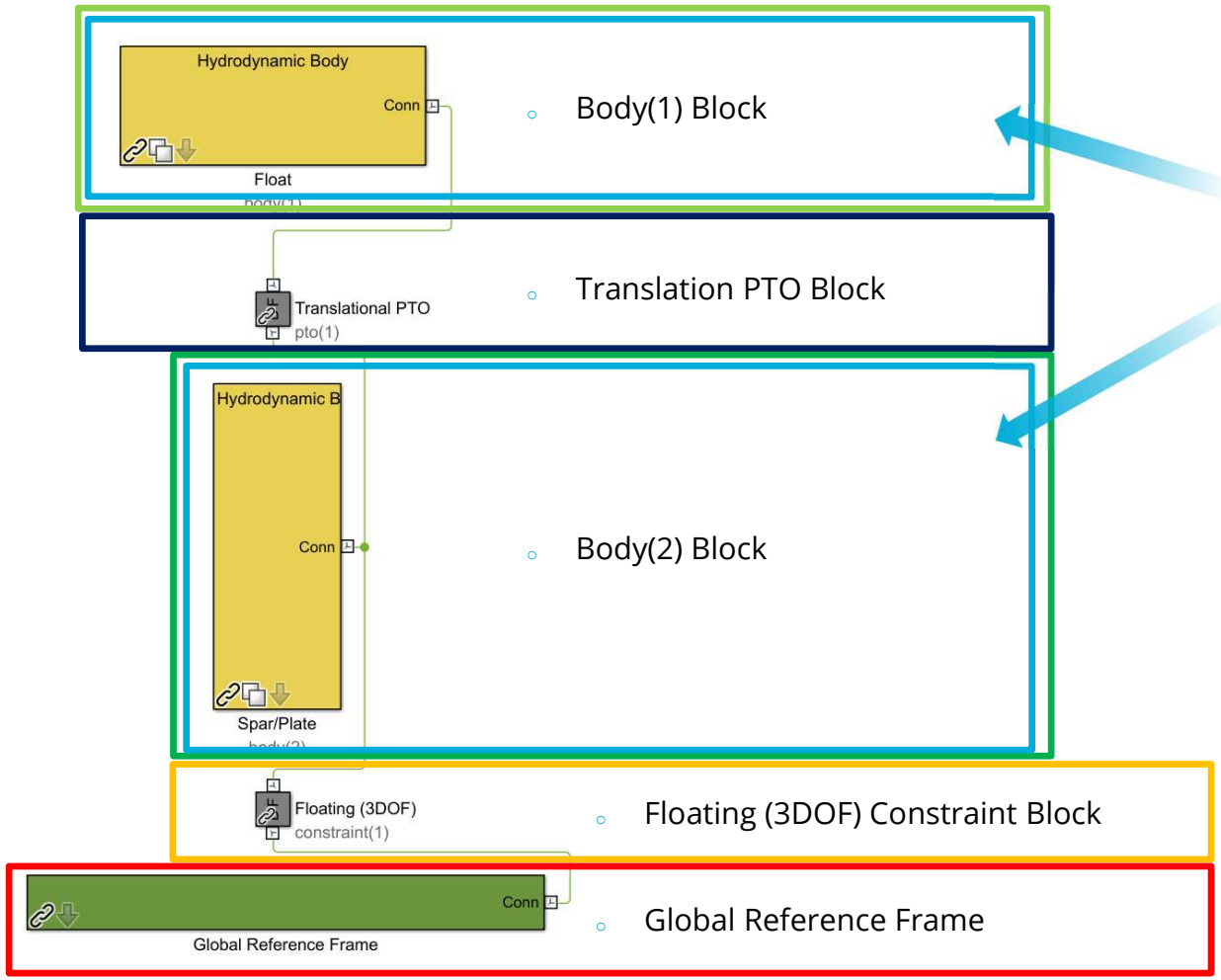
<simulinkModelName>.slx

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>



- Located in the case directory
- Define model file using WEC-Sim Library Blocks
  - ***<simulinkModelName>.slx***

WEC-Sim Simulink File



- Body(1) Block
- Translation PTO Block
- Body(2) Block
- Floating (3DOF) Constraint Block
- Global Reference Frame

```
%% Simulation Data
simu = simulationClass();
simu.simMechanicsFile = 'RM3.slx';
simu.mode = 'normal';
simu.explorer = 'on';
simu.startTime = 0;
simu.rampTime = 100;
simu.endTime = 400;
simu.solver = 'ode4';
simu.dt = 0.1;
```

```
%% Wave Information
% regular waves with radiation CIC
waves = waveClass('regularCIC');
waves.period = 6; % seconds
waves.height = 1; % meters
```

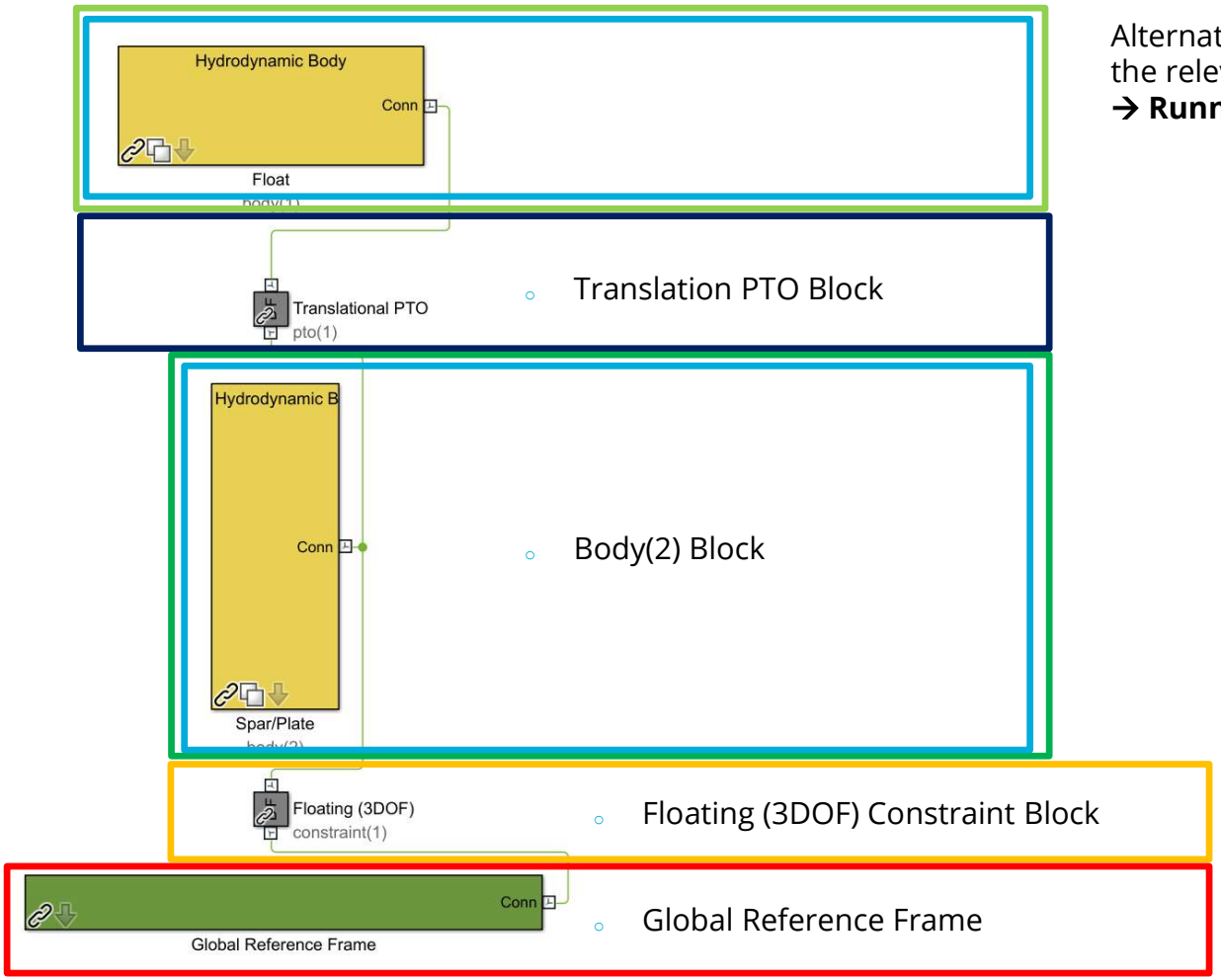
```
%% Body Data
% Float
body(1) = bodyClass('hydroData/rm3.h5');
body(1).geometryFile = 'geometry/float.stl';
body(1).mass = 'equilibrium';
body(1).inertia = [20907301 21306090.66 37085481.11];
```

```
% Spar/Plate
body(2) = bodyClass('hydroData/rm3.h5');
body(2).geometryFile = 'geometry/plate.stl';
body(2).mass = 'equilibrium';
body(2).inertia = [94419614.57 94407091.24 28542224.82];
```

```
%% PTO and Constraint Parameters
% Floating (3DOF) Joint
constraint(1) = constraintClass('Constraint1');
constraint(1).location = [0 0 0];
```

```
% Translational PTO
pto(1) = ptoClass('PTO1');
pto(1).stiffness = 0;
pto(1).damping = 1200000;
pto(1).location = [0 0 0];
```

WEC-Sim Simulink File



Alternatively, the parameters can be specified directly in the relevant Simulink blocks using **Advanced Features → Running from Simulink**

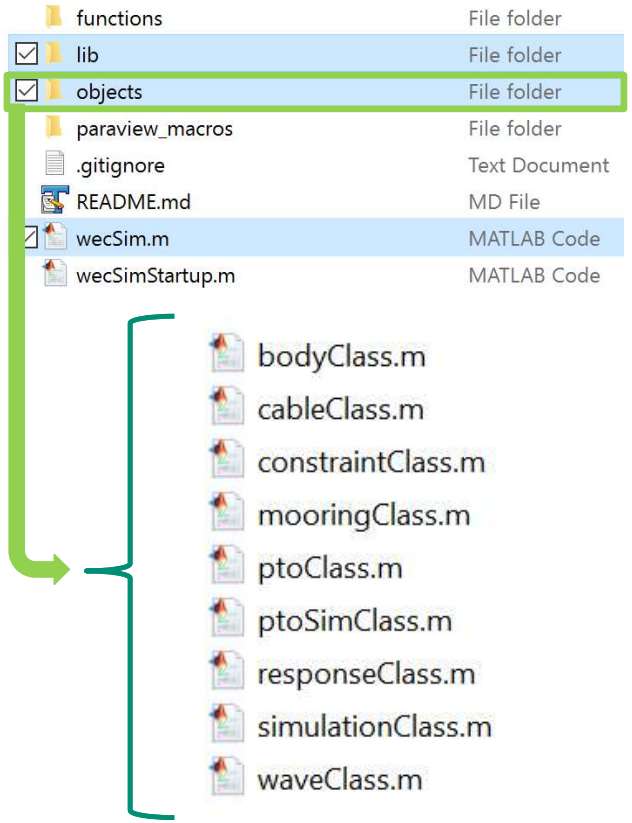
WEC-Sim Objects

WEC-Sim Objects

WEC-Sim/source/objects/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/Simulink

WEC-Sim/source



Define classes in the *wecSimInputFile.m*

The following classes create the WEC-Sim objects
- ***simulationClass.m, waveClass.m, bodyClass.m, constraintClass.m, ptoClass.m, mooringClass.m, cableClass.m***

WEC-Sim objects are required to run WEC-Sim simulations
- ***simu, waves, body(i), pto(i) OR constraint(i)***
- Additional object types can be defined if desired

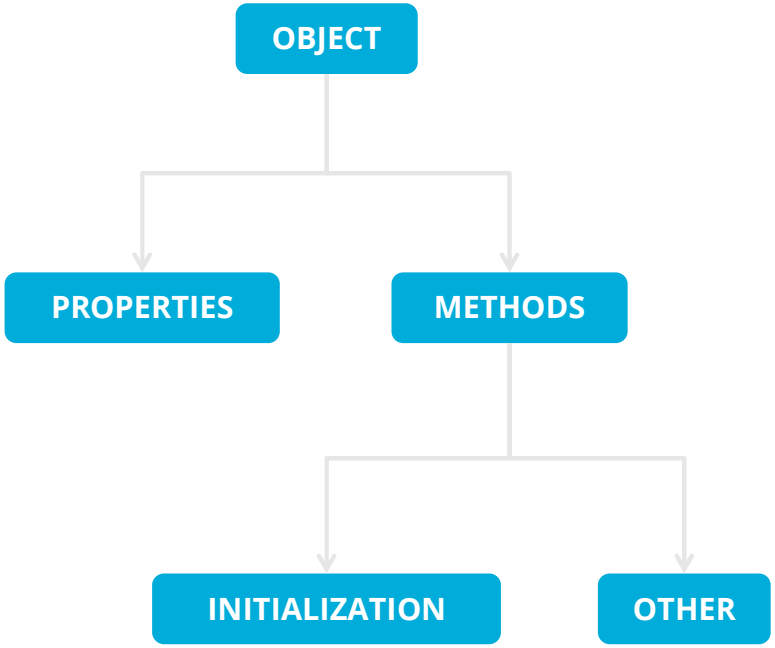
View properties or open classes from MATLAB Command Window
>> doc <className>
>> open <className>

***See User Manual→Code Structure → Source Detail for more**

WEC-Sim Objects

WEC-Sim has several different classes

- *simulationClass.m*
- *waveClass.m*
- *bodyClass.m*
- *constraintClass.m*
- *ptoClass.m*
- *mooringClass.m*
- *responseClass.m*
- *cableClass.m*



Each class contains:

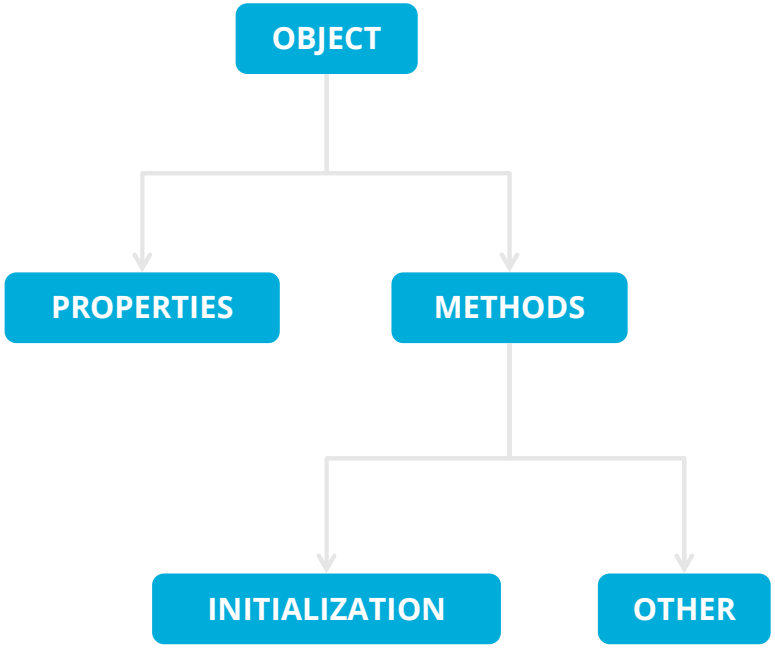
- Properties that can be defined and/or calculated
- Methods (aka functions) that can be executed

WEC-Sim input file determines which properties are defined and methods are executed

WEC-Sim Objects

Each class creates a corresponding object that will appear in the workspace

- *simulationClass.m* → *simu*
- *waveClass.m* → *waves*
- *bodyClass.m* → *body(i)*
- *constraintClass.m* → *constraint(i)*
- *ptoClass.m* → *pto(i)*
- *mooringClass.m* → *mooring(i)*
- *responseClass.m* → *output*
- *cableClass.m* → *cable(i)*



Some properties are used to specify a variant subsystem, e.g.

- *simu.b2b = 1;*
- *body(i).nhBody = 1;*
- *waves = waveClass('regular');*

For help, >>doc <name>Class

See also User Manual→ Code Structure → WEC-Sim Classes

WEC-Sim Class Descriptions

Simulation Class

simulationClass.m

The simulation class contains the simulation parameters and solver settings necessary to execute the WEC-Sim code.

Required Properties:

- simMechanicsFile
- startTime, endTime, dt, rampTime, cicEndTime
  - (many have default values)

*** See User Manual → Code Structure → Simulation Class**

and

API → Simulation Class

```
>>simu

simu =

  simulationClass with properties:

      adjMassFactor: 2
                b2b: 0
              cicDt: 0.1000
         cicEndTime: 60
         domainSize: 200
                 dt: 0.1000
              dtOut: 0.1000
            endTime: 400
           explorer: 'on'
            gravity: 9.8100
          mcrMatFile: [1×0 char]
        mcrExcelFile: [1×0 char]
               mode: 'normal'
          morisonDt: 0.1000
        nonlinearDt: 0.1000
           paraview: [1×1 struct]
           pressure: 0
            rampTime: 100
      rateTransition: 'on'
        reloadH5Data: 0
                rho: 1000
       saveStructure: 0
            saveText: 0
       saveWorkspace: 1
```

Wave Class

waveClass.m

The wave class contains all wave information necessary to define the incident wave condition for the WEC-Sim time-domain simulation. In the Simulink model, wave forces are applied inside the body(i) blocks.

Required Properties:

- type
- Each wave 'type' has different required properties

Wave Type	Required Properties
`noWave`	`waves.period`
`noWaveCIC`	
`regular`	`waves.height` , `waves.period`
`regularCIC`	`waves.height` , `waves.period`
`irregular`	`waves.height` , `waves.period` , `waves.spectrumType`
`spectrumImport`	`waves.spectrumFile`
`elevationImport`	`waves.elevationFile`

*** See Training Videos→ Wave Implementation, User Manual → Code Structure → Wave Class, and API → Wave Class**

>>waves

```
waveClass with properties:

              bem: [1×1 struct]
          current: [1×1 struct]
        direction: 0
    elevationFile: 'NOT DEFINED'
            gamma: [1×0 double]
           height: 1
           marker: [1×1 struct]
           period: 6
        phaseSeed: 0
     spectrumFile: 'NOT DEFINED'
     spectrumType: 'NOT DEFINED'
              viz: [1×1 struct]
       waterDepth: 200
           spread: 1
        amplitude: 0.5000
        deepWater: 1
           dOmega: 0
            omega: 1.0472
            phase: 0
            power: 5.7437e+03
         spectrum: []
             type: 'regularCIC'
          typeNum: 11
      waveAmpTime: [4001×2 double]
   waveAmpTimeViz: []
       wavenumber: 0.1118
```

Body Class

bodyClass.m

The body class contains the mass and hydrodynamic properties of each body that comprises the WEC being simulated.

Required Properties:

- mass: value, 'equilibrium'
- inertia
- product of inertia (v5.1.0 release)
- geometryFile (**This is used for visualization and some Adv. Feat.)**
- h5File (**This contains hydrodynamic data from BEM)**

***See Training Videos → Body Class Implementation**

User Manual→Code Structure → Source Details Body Class

And

API → Body Class

>>body

```
body =

  1×2 bodyClass array with properties:

    centerBuoyancy
    centerGravity
    dof
    excitationIRF
    flex
    gbmDOF
    geometryFile
    h5File
    hydroStiffness
    inertia
    initial
    largeXYDisplacement
    linearDamping
    mass
    meanDrift
    morisonElement
    name
    nonHydro
    nonlinearHydro
    quadDrag
    paraview
    viz
    volume
    yaw
    dofEnd
    dofStart
```

Constraint and PTO Classes

constraintClass.m ***ptoClass.m***

Constraint blocks connect WEC bodies to one another (and possibly to the seabed) by constraining DOFs. PTOs do the same and can also apply force along their DOF of action. Unique blocks are available for different DOF restriction (e.g., rotational, translational, spherical)

Constraint and PTO Class required properties:

- name
- location

Additional PTO Class properties that describe applied force. The length of these fields must match the number of unconstrained DOF in the PTO.

- stiffness (*non-negative)*
- pretension
- damping

*** For additional information, see:**
User Manual → Code Structure → Constraint Class
User Manual → Code Structure → PTO Class
API → Constraint Class
API→ PTO Class

*** For component-level PTO design, see also Adv. Features → PTO-Sim**

>>constraint

```
constraint =

  constraintClass with properties:

      hardStops: [1×1 struct]
        initial: [1×1 struct]
       location: [0 0 0]
           name: 'Constraint1'
    orientation: [1×1 struct]
         number: 1
```

>> pto

```
>> pto

pto =

  ptoClass with properties:

                damping: 1200000
  equilibriumPosition: 0
            hardStops: [1×1 struct]
              initial: [1×1 struct]
             location: [0 0 0]
                 name: 'PTO1'
          orientation: [1×1 struct]
           pretension: 0
            stiffness: 0
               number: 1
```

Mooring Class

mooringClass.m

Mooring class defines the mooring system as either a linear mooring matrix or a MoorDyn model. It is designed to couple a WEC body, PTO, or Constraint to the sea-bed

Mooring types:

- matrices
- MoorDyn

Properties for matrix:

- name
- location
- Matrix
  - stiffness
  - damping
  - pretension

For additional information, see:

User Manual→ Code Structure→ Mooring Class

Advanced Features → Mooring Features→ MoorDyn

API→ Mooring Class

>>mooring

```
mooring =

  mooringClass with properties:

          initial: [1×1 struct]
         location: [0 0 0]
           matrix: [1×1 struct]
          moorDyn: 0
      moorDynLines: 0
      moorDynNodes: [1×0 double]
              name: 'Mooring1'
       orientation: [0 0 0 0 0 0]
            number: 1
```

>>mooring.matrix

```
ans =

  struct with fields:

        damping: [6×6 double]
      stiffness: [6×6 double]
     preTension: [0 0 100 0 0 0]
```

Cable Class

cableClass.m

Cable class describes a compliant cable that connects two constraints or PTOs. The constraint/PTO defines how the cable connection is allowed to move. If the cable is not in tension, it does not transmit force between the connection points.

Cable required properties:

- stiffness
- damping

By default, cable length and end locations, are determined from the connected constraints/PTOs, assuming zero pretension.

See also: WEC-Sim Applications/Cable

Advanced Features →Cable Features

User Manual → Code Structure → Cable Class

```
>>cable

cable =

  cableClass with properties:

         damping: 100
         inertia: [1 1 1]
         initial: [1×1 struct]
     cableLength: 17.8000
   linearDamping: [0 0 0 0 0 0]
            mass: 1
            name: 'Cable'
     orientation: [1×1 struct]
        paraview: 1
      preTension: 0
        quadDrag: [1×1 struct]
       stiffness: 1000000
             viz: [1×1 struct]
            base: [1×1 struct]
        follower: [1×1 struct]
        location: [999 999 999]
          volume: []
          number: []
```

Response Class (Output Structure)

responseClass.m

'output' created at the end of a WEC-Sim simulation. It contains all the output time-series and methods to plot and interact with the results.

output = responseClass instance

- Contains all time series from simulation
- Contains all time-series calculations
- Methods for quick plotting

Properties are all defined objects, each with their own sub-fields.

This structure is created before *userDefinedFunctions* runs, so *userDefinedFunctions* can reference **output.**

For additional information, see:
User Manual → Code Structure → Response Class and API → Response Class

```
>>output
    output =

      responseClass with properties:

            bodies: [1×2 struct]
            cables: [1×1 struct]
       constraints: [1×1 struct]
           moorDyn: [1×1 struct]
           mooring: [1×1 struct]
              ptos: [1×1 struct]
            ptosim: [1×1 struct]
              wave: [1×1 struct]

>>output.ptos(1)
ans =

  struct with fields:

              name: 'PTO1'
              time: [4001×1 double]
          position: [4001×6 double]
          velocity: [4001×6 double]
      acceleration: [4001×6 double]
        forceTotal: [4001×6 double]
     forceActuation: [4001×6 double]
    forceConstraint: [4001×6 double]
forceInternalMechanics: [4001×6 double]
powerInternalMechanics: [4001×6 double]
```

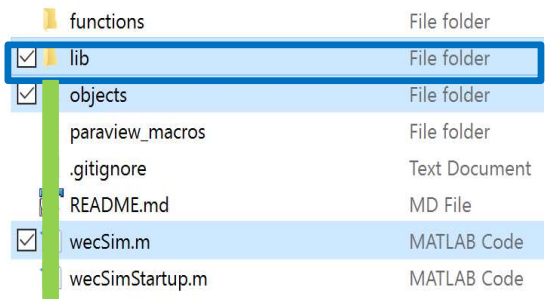
WEC-Sim Library

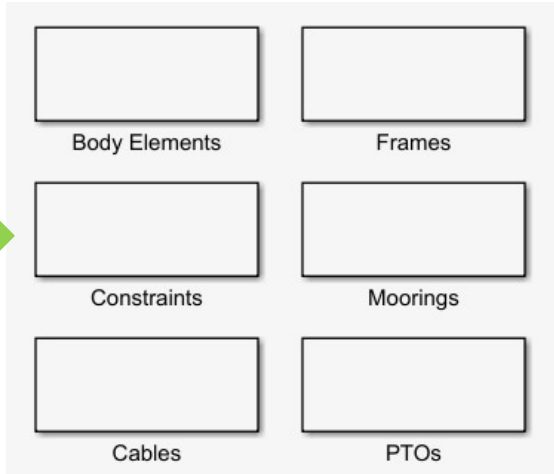
WEC-Sim Library Blocks

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim/source





WEC-Sim source code includes WEC-Sim library blocks:

- ***Body Elements, Constraints, Frames, Moorings, PTOs, Cables***

Define WEC dynamics in WEC-Sim Simulink model using WEC-Sim Library Blocks

- ***<SimulinkModelName>.slx***

View properties by double clicking on blocks, Ctrl+U to look under mask

https://www.mathworks.com/help/simulink/block-masks.html

All objects defined in the *wecSimInputFile* should also be blocks used in the Simulink model.

WEC-Sim Library

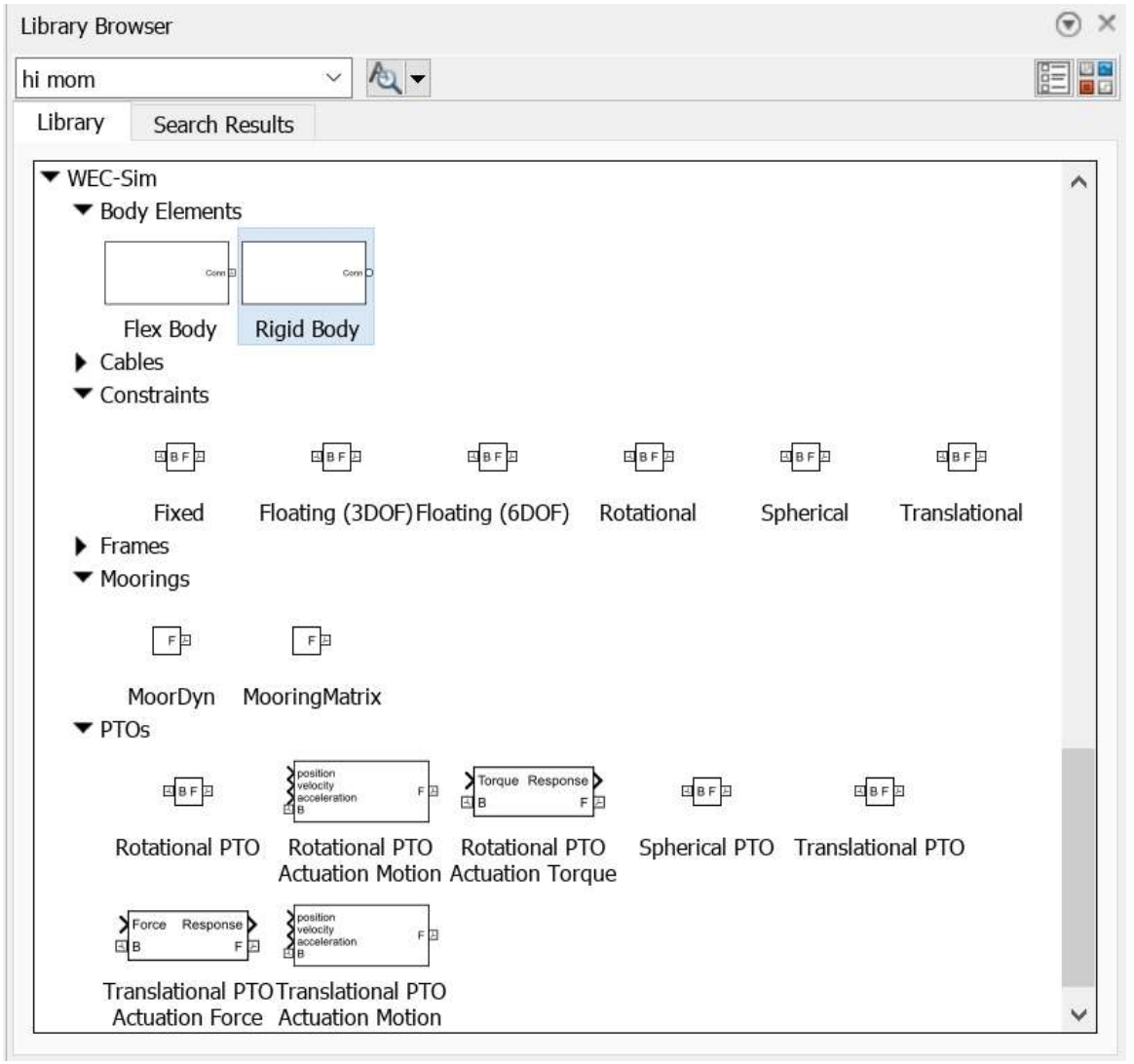
WEC-Sim/source/lib/

WEC-Sim Library

- Drag & Drop library
- "Source Code" blocks

Simulink Model

- Made of WEC-Sim library blocks
- Blocks cannot have the same name: model will automatically number repeated block types.

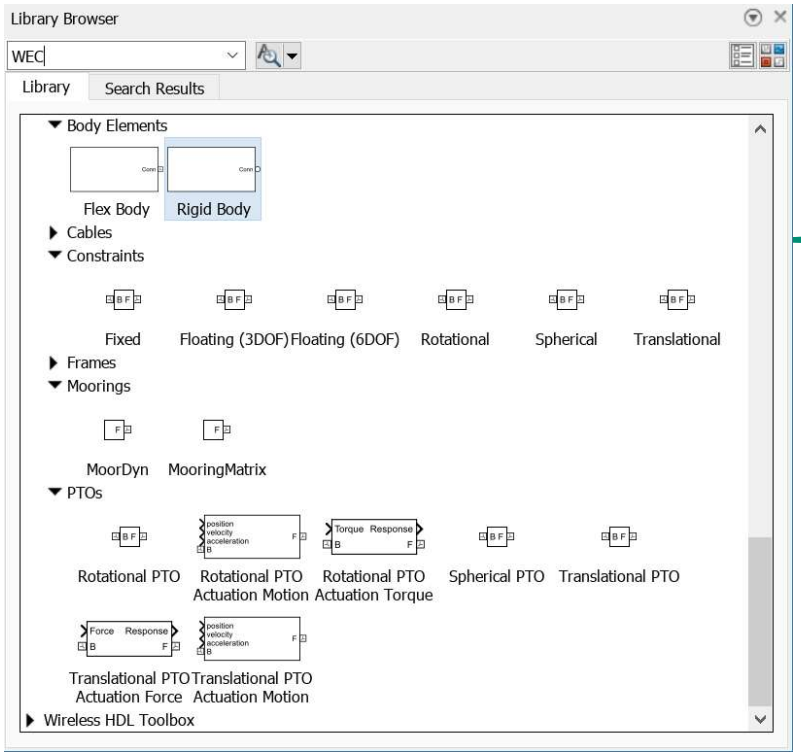


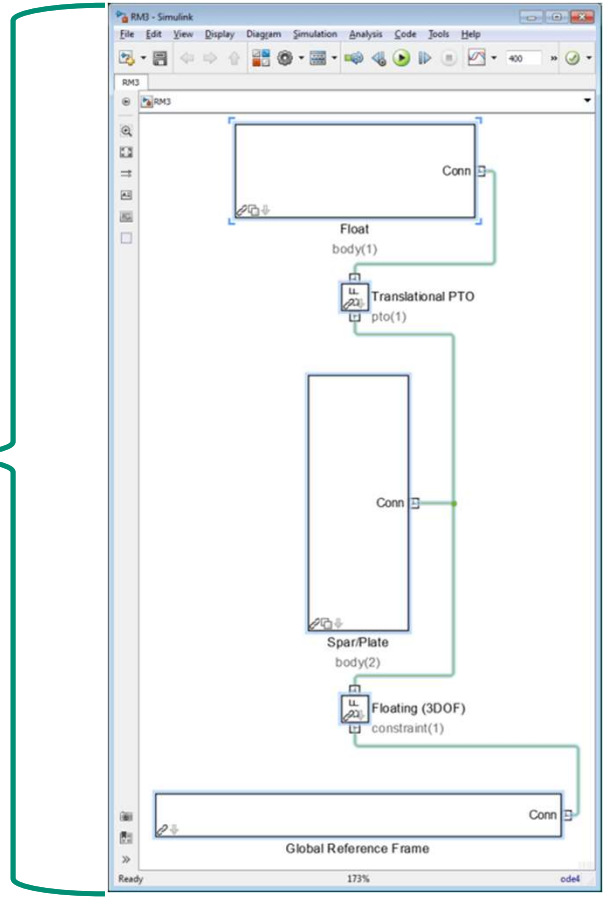
WEC-Sim Simulink File

<simulinkModelName>.slx

WEC-Sim Simulink Model

- Created with WEC-Sim Simulink Library Blocks
- Free to incorporate other Simscape/Simulink components



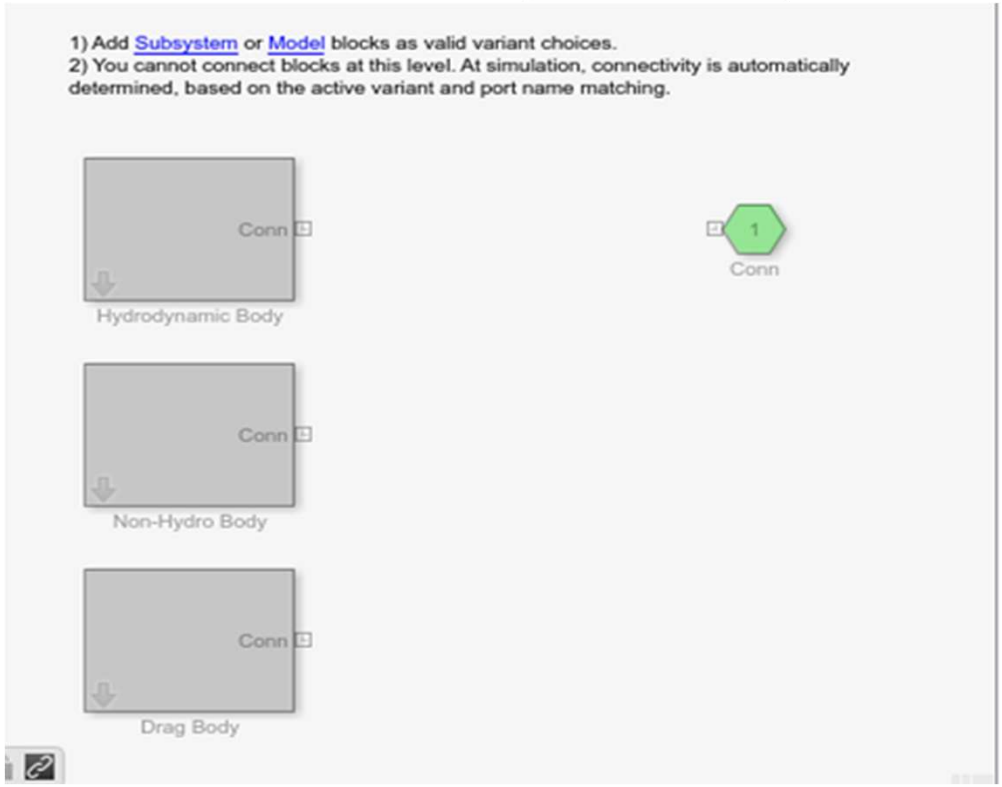


Variant Subsystems

WEC-Sim/source/lib/

Many library blocks contain 'Variant Subsystems'

- Variant subsystems allow multiple implementations to exist within a single model, with one active at a time.
- You can programmatically swap out the active implementation and replace it with one of the other implementations without modifying the model.
- The specification of active subsystems happens within *initializeWecSim.m*.

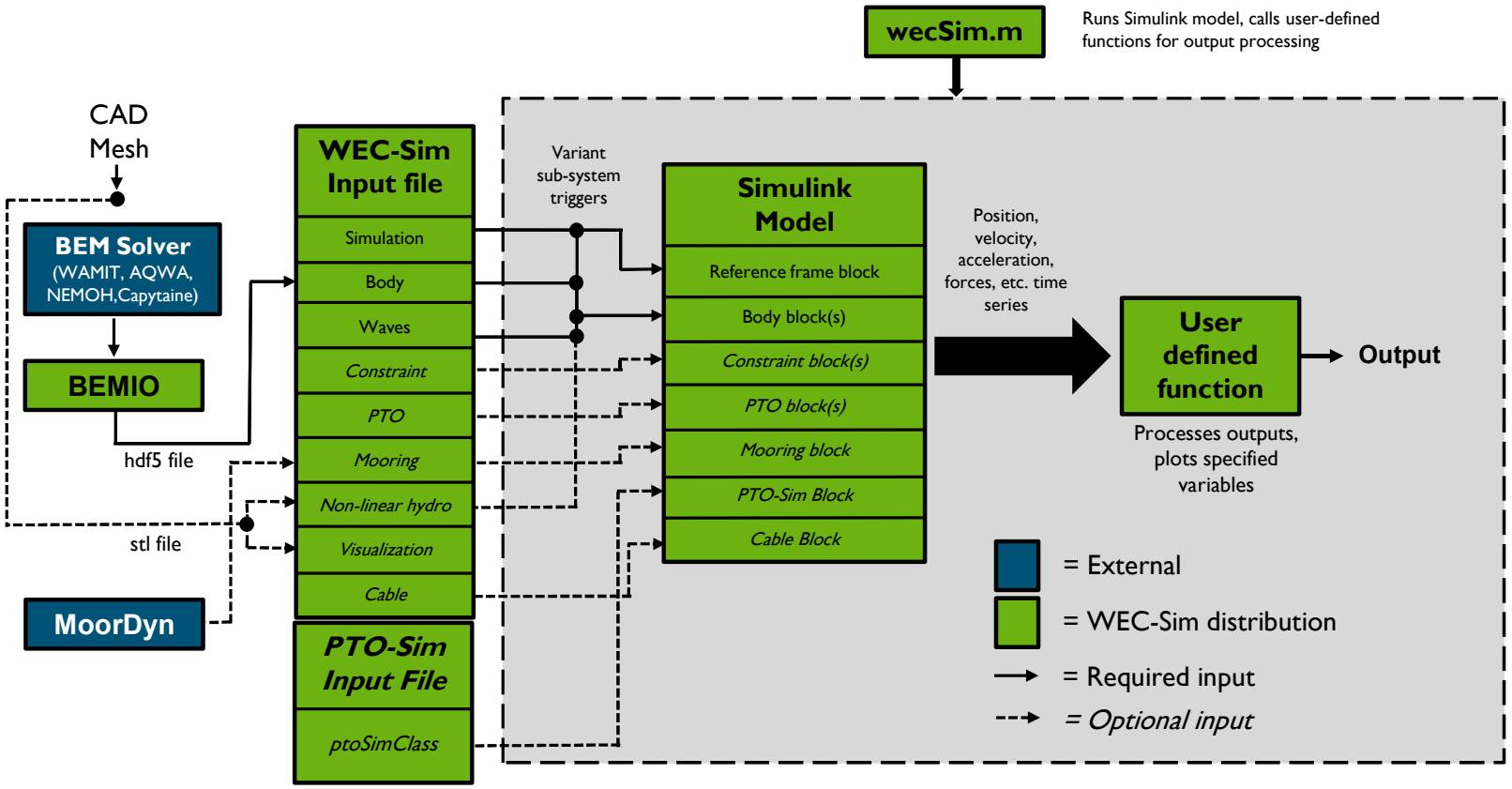


*** See Training Materials → Theory and Workflow for more information**

https://www.mathworks.com/help/simulink/examples/variant-subsystems.html

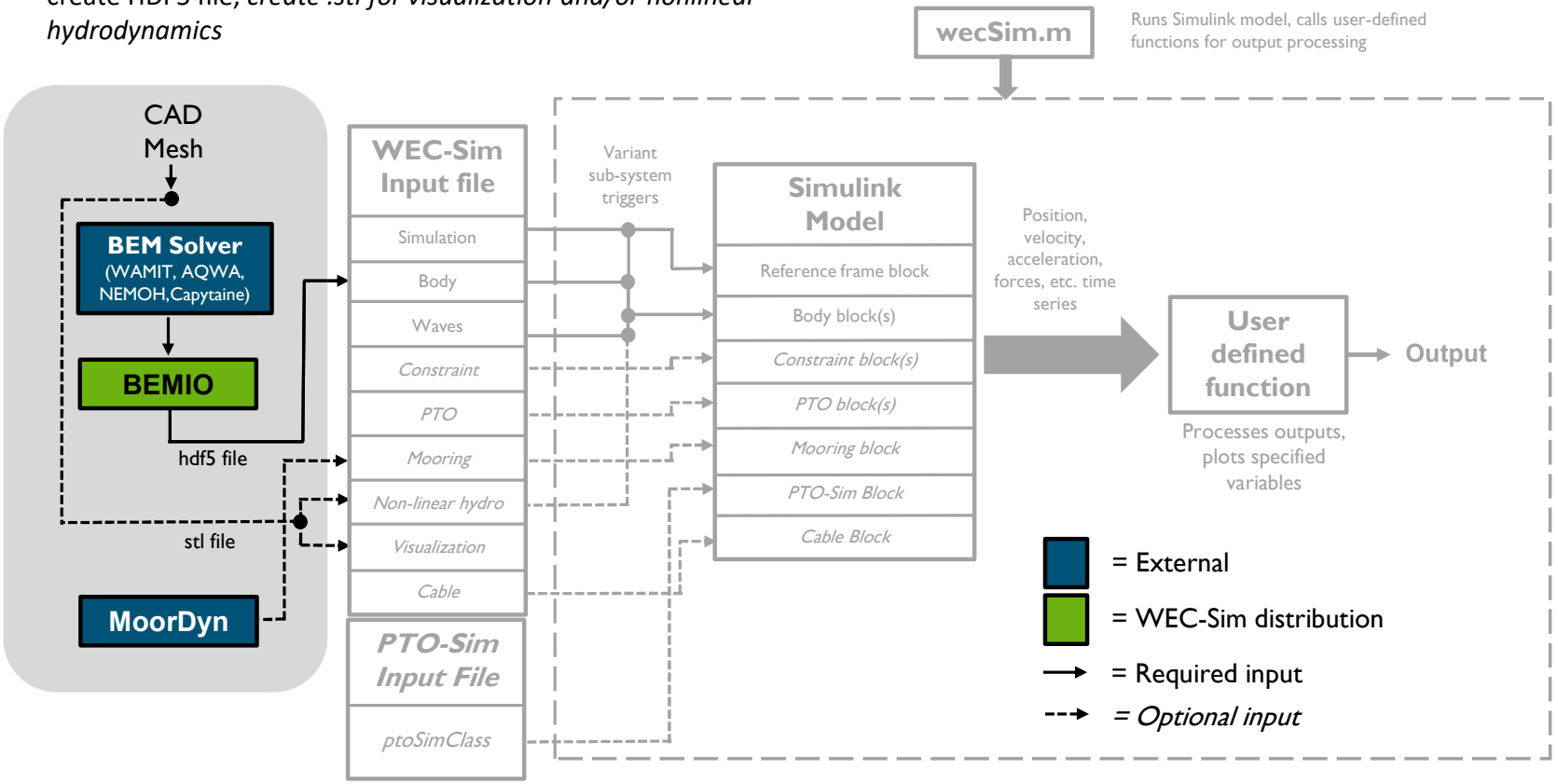
In conclusion…

WEC-Sim Workflow



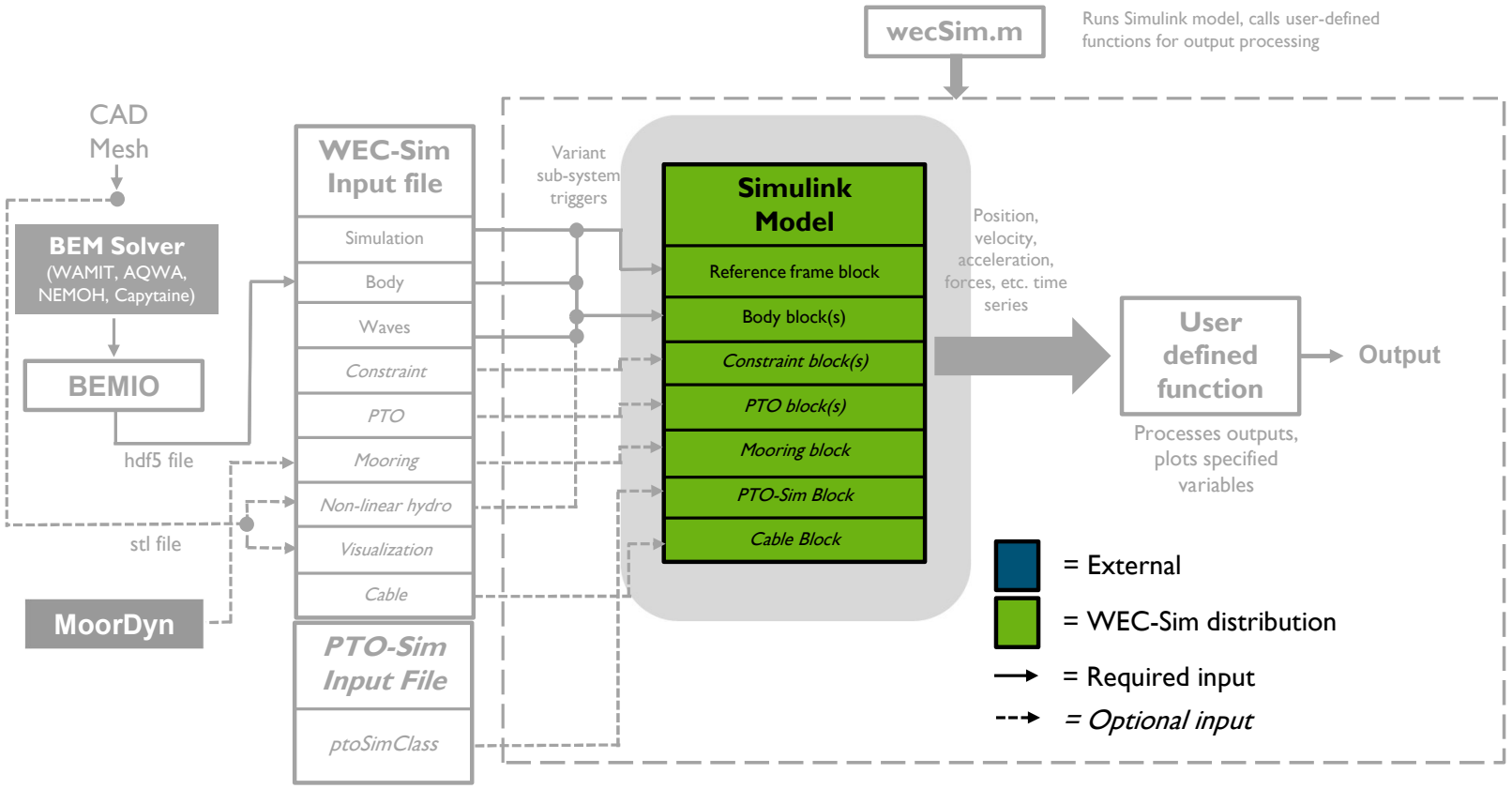
WEC-Sim Workflow

1). Generate 3-D mesh, calculate hydrodynamic coefficients, create HDF5 file, *create .stl for visualization and/or nonlinear hydrodynamics*



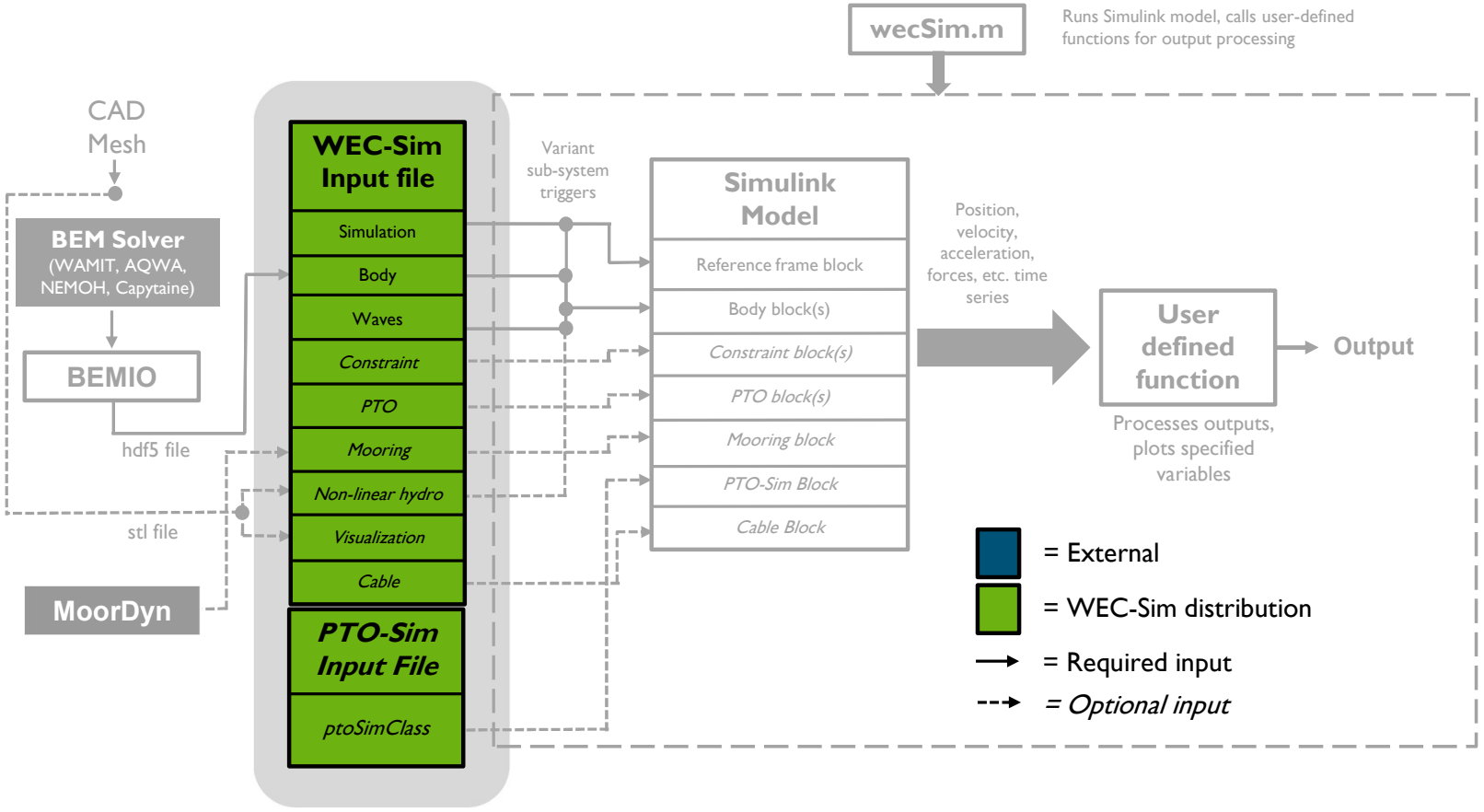
WEC-Sim Workflow

2). Build WEC-Sim model in Simulink



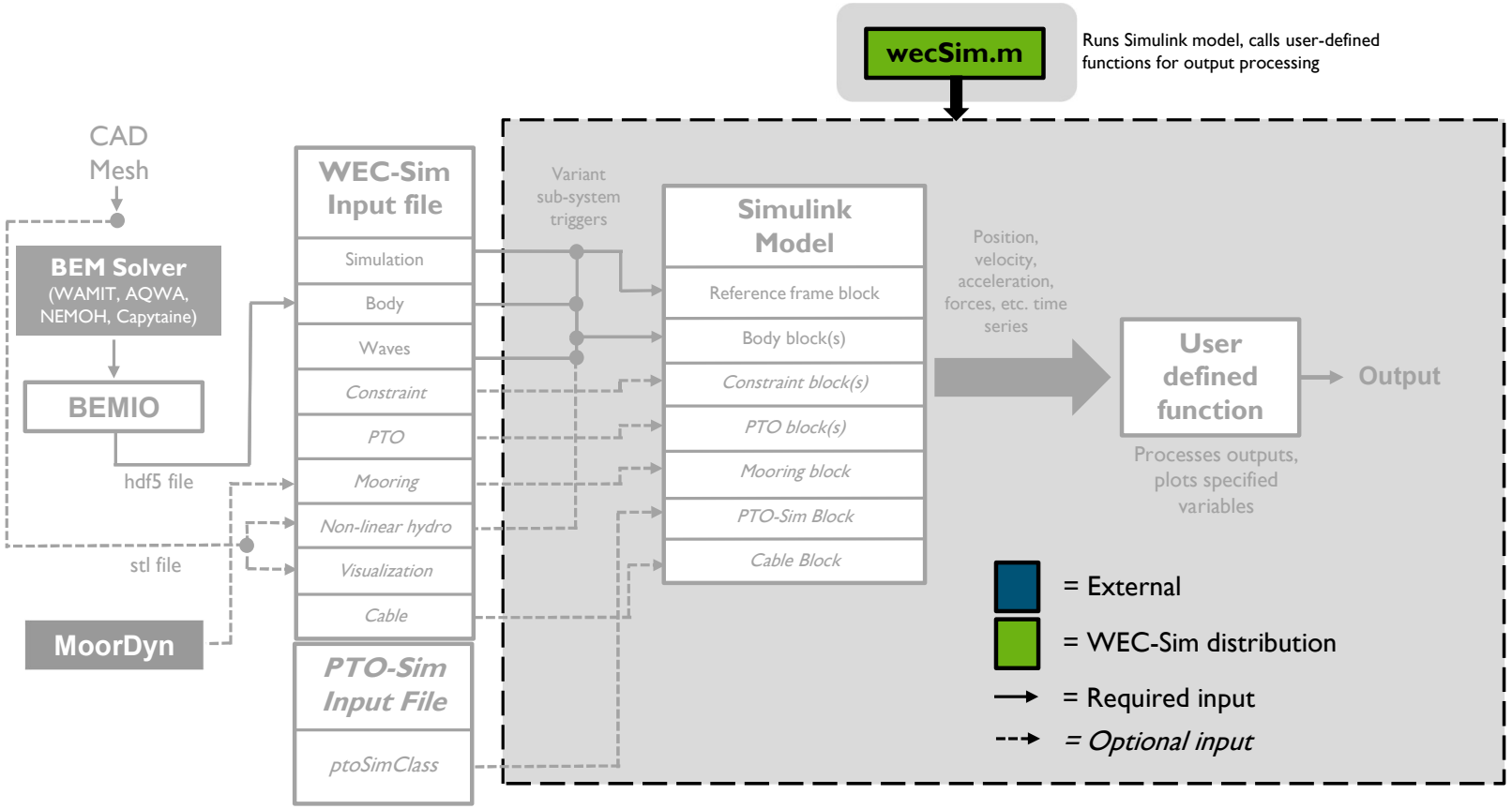
WEC-Sim Workflow

3). Write WEC-Sim input file



WEC-Sim Workflow

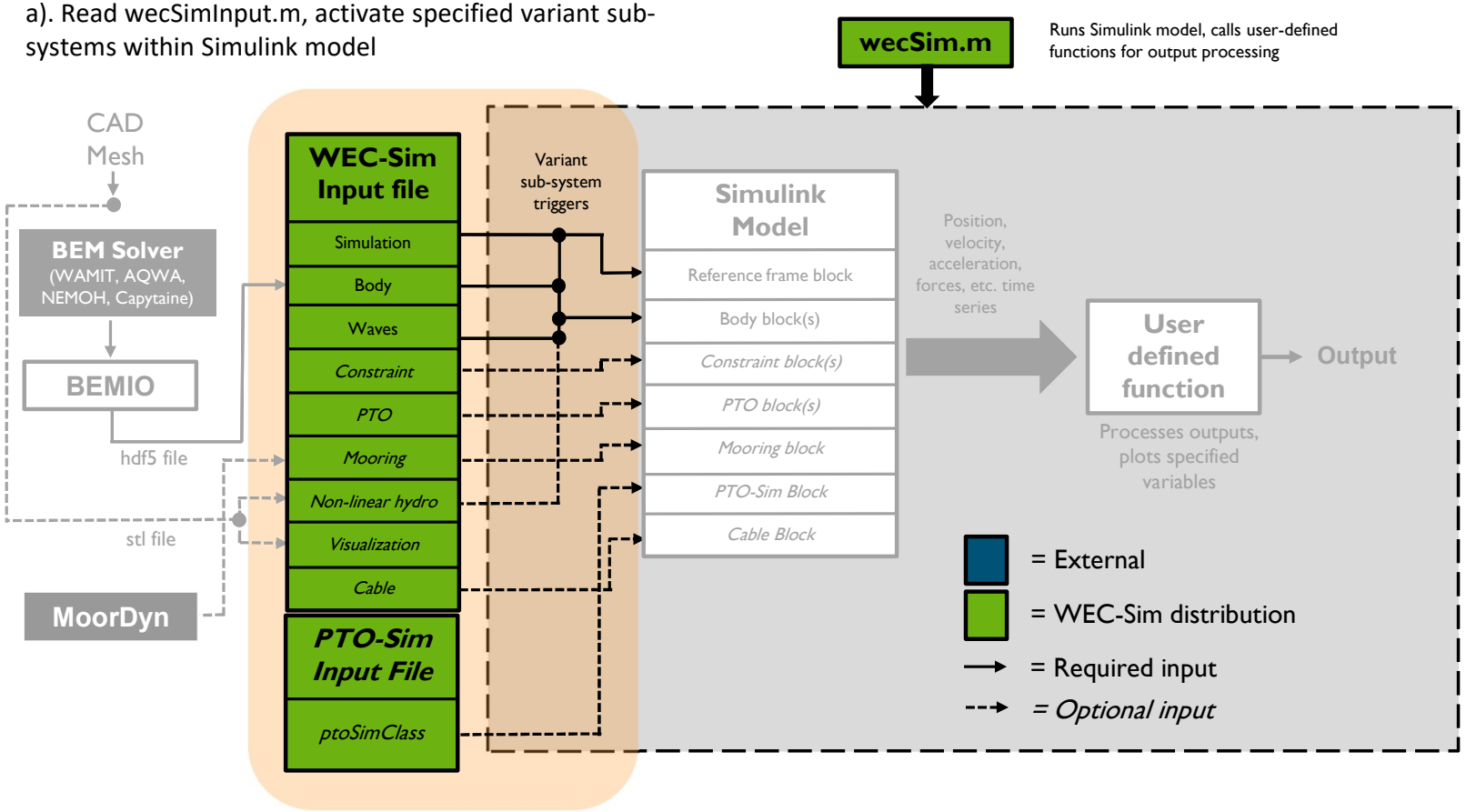
4). Execute wecSim.m



WEC-Sim Workflow

4). Execute wecSim.m

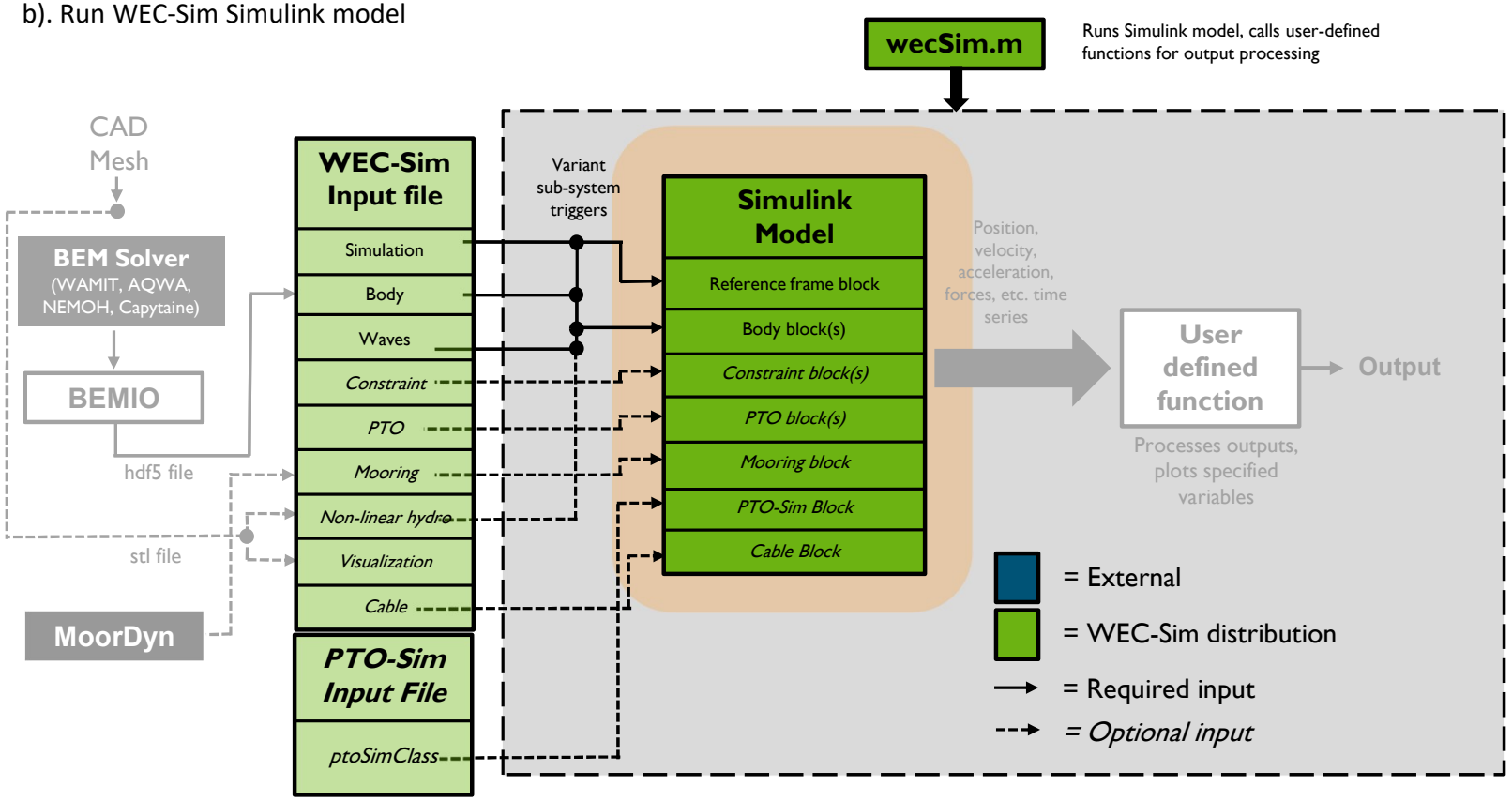
a). Read wecSimInput.m, activate specified variant sub-systems within Simulink model



WEC-Sim Workflow

4). Execute wecSim.m

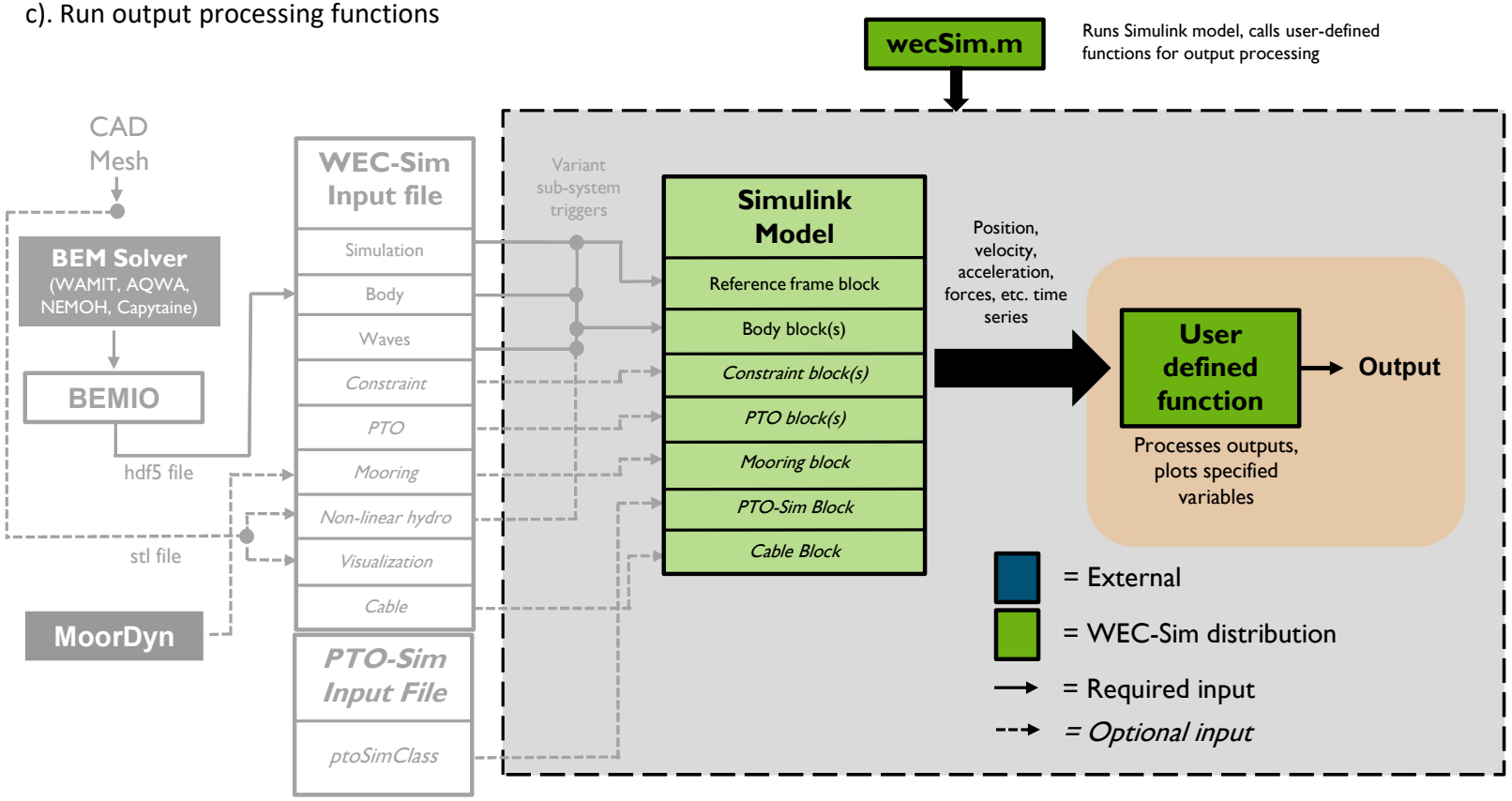
b). Run WEC-Sim Simulink model



WEC-Sim Workflow

4). Execute wecSim.m

c). Run output processing functions



BEMIO

What is BEMIO

Workflow: BEM → BEMIO → WEC-Sim

- The BEMIO (**B**oundary **E**lement **M**ethod **I**nput/**O**utput) functions are used to preprocess the BEM hydrodynamic data prior to running WEC-Sim.

Purpose

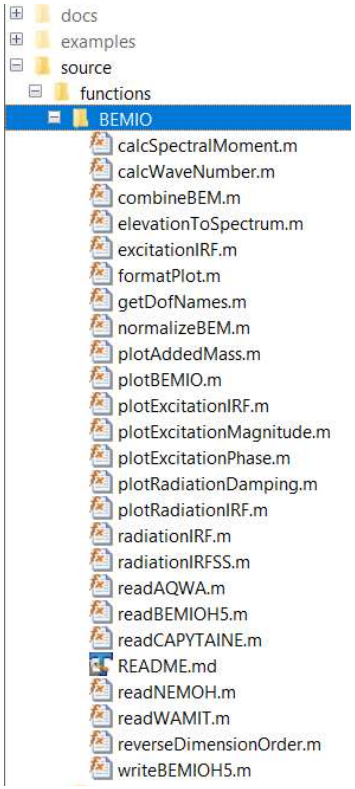
- Read BEM results from WAMIT, NEMOH, Capytaine, or AQWA.
- Calculate the radiation and excitation impulse response functions (IRFs).
- Calculate state space realization coefficients for the radiation IRF.
- Save the resulting data in Hierarchical Data Format 5 (HDF5).
- Plot typical hydrodynamic data for user verification.

Implementation

- Includes .h5 file, MATLAB (***hydro***) data structure

Locations

- Functions: ***\\WEC-Sim\source\functions\BEMIO***
- Documentation: https://wec-sim.github.io/WEC-Sim/master/user/advanced_features.html#bemio



readWAMIT Reads data from a WAMIT output file (*.out)

functions.BEMIO.readWAMIT(*hydro, filename, exCoeff*)

Reads data from a WAMIT output file.

If generalized body modes are used, the output directory must also include the **.cfg*, **.mmx*, and **.hst* files. If simu.nonlinearHydro = 3 will be used, the output directory must also include the **.3fk* and **.3sc* files.

See `WEC-Sim/examples/BEMIO/WAMIT` for examples of usage.

Parameters:

- **hydro** (`struct`) – Structure of hydro data that WAMIT input data will be appended to
- **filename** (`string`) – Path to the WAMIT output file
- **exCoeff** (`integer`) – Flag indicating the type of excitation force coefficients to read, 'diffraction' (default), 'haskind', or 'rao'

Returns: **hydro** – Structure of hydro data with WAMIT data appended

Return type: struct

bemio.m

```
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[]);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```

readNEMOH Reads data from a NEMOH working folder

functions.BEMIO.readNEMOH(*hydro, filedir*)

Reads data from a NEMOH working folder.

See `WEC-Sim\examples\BEMIO\NEMOH` for examples of usage.

Parameters:

- **hydro** (`struct`) – Structure of hydro data that NEMOH input data will be appended to
- **filename** (`string`) –
  Path to the NEMOH working folder, must include:
  - `Nemoh.cal`
  - `Mesh/Hydrostatics.dat` (or `Hydrostatiscs_0.dat` , `Hydrostatics_1.dat` , etc. for multiple bodies)
  - `Mesh/KH.dat (or ``KH_0.dat` , `KH_1.dat` , etc. for multiple bodies)
  - `Results/RadiationCoefficients.tec`
  - `Results/ExcitationForce.tec`
  - `Results/DiffractionForce.tec` - If simu.nonlinearHydro = 3 will be used
  - `Results/FKForce.tec` - If simu.nonlinearHydro = 3 will be used

Returns: **hydro** – Structure of hydro data with NEMOH data appended

Return type: struct

bemio.m

```
1  hydro = struct();
2
3  hydro = readNEMOH(hydro,'../RM3/');
4  % hydro = readWAMIT(hydro,'../../WAMIT/RM3/rm3.out',[]);
5  % hydro = combineBEM(hydro); % Compare WAMIT
6  hydro = radiationIRF(hydro,60,[],[],[],1.9);
7  hydro = radiationIRFSS(hydro,[],[]);
8  hydro = excitationIRF(hydro,157,[],[],[],1.9);
9  writeBEMIOH5(hydro)
10 plotBEMIO(hydro)
11
```

readAQWA Reads data from AQWA output files

functions.BEMIO.readAQWA(*hydro, ah1Filename, lisFilename*)

Reads data from AQWA output files.

See `WEC-Sim\examples\BEMIO\AQWA` for examples of usage.

Parameters:
- **hydro** (`struct`) – Structure of hydro data that Aqwa input data will be appended to
- **ah1Filename** (`string`) – .AH1 AQWA output file
- **lisFilename** (`string`) – .LIS AQWA output file

Returns: **hydro** – Structure of hydro data with Aqwa data appended

Return type: struct

bemio.m

```
1  hydro = struct();
2
3  hydro = readAQWA(hydro, 'RM3.AH1', 'RM3.LIS');
4  hydro = radiationIRF(hydro,150,[],[],[],1.8);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,150,[],[],[],1.8);
7  writeBEMIOH5(hydro)
8
```

readCAPYTAINE Reads data from a Capytaine netcdf file

functions.BEMIO.readCAPYTAINE(*hydro, filename*)

Reads data from a Capytaine netcdf file

See `WEC-Sim\examples\BEMIO\CAPYTAINE` for examples of usage.

bemio.m

```
1  hydro = struct();
2
3  hydro = readCAPYTAINE(hydro,'rm3_full.nc');
4  hydro = radiationIRF(hydro,60,[],[],[],1.9);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,157,[],[],[],1.9);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```

Parameters:

- **hydro** (`struct`) – Structure of hydro data that Capytaine input data will be appended to
- **filename** (`string`) – Capytaine .nc output file

Returns: **hydro** – Structure of hydro data with Capytaine data appended

Return type: struct

combineBEM Combines multiple BEM outputs into one hydrodynamic ‘system’

functions.BEMIO.combineBEM(*hydro*)

Combines multiple BEM outputs into one hydrodynamic ‘system.’ This function requires that all BEM outputs have the same water depth, wave frequencies, and wave headings. This function would be implemented following multiple readWAMIT, readNEMOH, readCAPYTAINE, or readAQWA and before radiationIRF, radiationIRFSS, excitationIRF, writeBEMIOH5, or plotBEMIO function calls.

See `WEC-Sim\examples\BEMIO\NEMOH` for examples of usage.

Parameters: **hydro** (`[1 x n] struct`) – Structures of hydro data that will be combined into a single structure

Returns: **hydro** – Combined structure.

Return type: [1 x 1] struct

bemio.m

```
1   hydro = struct();
2
3   hydro = readNEMOH(hydro,'../RM3/');
4   hydro = readWAMIT(hydro,'../../WAMIT/RM3/rm3.out',[]);
5   hydro = combineBEM(hydro); % Compare WAMIT
6   hydro = radiationIRF(hydro,60,[],[],[],1.9);
7   hydro = radiationIRFSS(hydro,[],[]);
8   hydro = excitationIRF(hydro,157,[],[],[],1.9);
9   writeBEMIOH5(hydro)
10  plotBEMIO(hydro)
11
```

radiationIRF Calculates the normalized radiation impulse response function

functions.BEMIO.radiationIRF(*hydro, tEnd, nDt, nDw, wMin, wMax*)

Calculates the normalized radiation impulse response function. This is equivalent to the radiation IRF in the theory section normalized by ρ:

$$\overline{K}_{r,i,j}(t) = \frac{2}{\pi}\int_0^\infty \frac{B_{i,j}(\omega)}{\rho}\cos(\omega t)d\omega$$

Default parameters can be used by inputting []. See `WEC-Sim\examples\BEMIO` for examples of usage.

Parameters:

- **hydro** (`struct`) – Structure of hydro data
- **tEnd** (`float`) – Calculation range for the IRF, where the IRF is calculated from t = 0 to tEnd, and the default is 100 s
- **nDt** (`float`) – Number of time steps in the IRF, the default is 1001
- **nDw** (`float`) – Number of frequency steps used in the IRF calculation (hydrodynamic coefficients are interpolated to correspond), the default is 1001
- **wMin** (`float`) – Minimum frequency to use in the IRF calculation, the default is the minimum frequency from the BEM data
- **wMax** (`float`) – Maximum frequency to use in the IRF calculation, the default is the maximum frequency from the BEM data

Returns: **hydro** – Structure of hydro data with radiation IRF

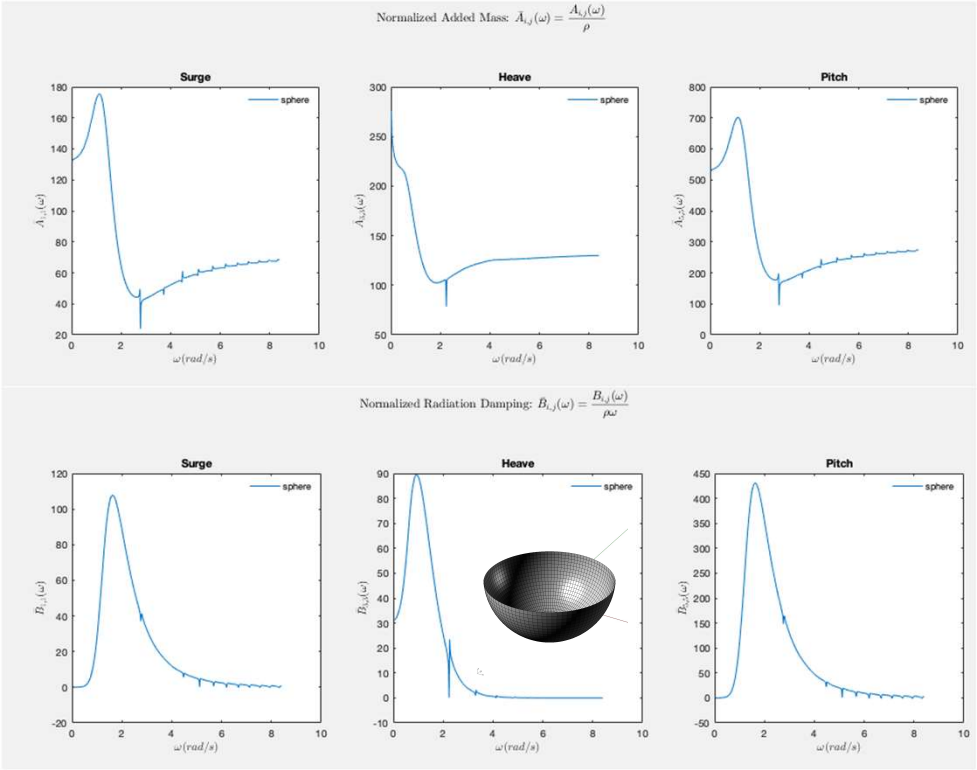
Return type: struct

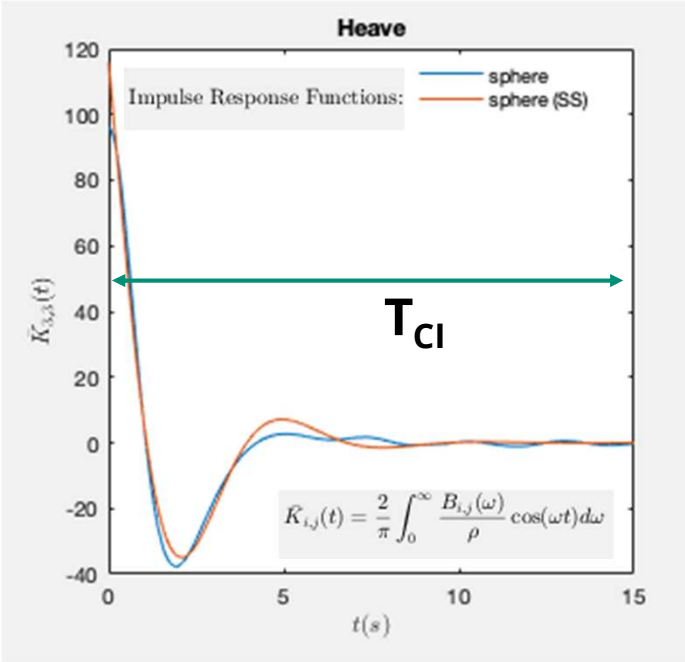
bemio.m

```
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[]);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```

$$\overline{K}_{i,j}(t) = \frac{2}{\pi}\int_0^\infty \frac{B_{i,j}(\omega)}{\rho}\cos(\omega t)\,d\omega$$

radiationIRF Calculates the normalized radiation impulse response function





NOTE: Make sure *simu.cicEndTime* <= T_{CI}

radiationIRFSS Calculates the state space (SS) realization of the radiation IRF

functions.BEMIO.radiationIRFSS(*hydro*, *Omax*, *R2t*)

Calculates the state space (SS) realization of the normalized radiation IRF. If this function is used, it must be implemented after the radiationIRF function.

Default parameters can be used by inputting []. See `WEC-Sim\examples\BEMIO` for examples of usage.

Parameters:
- **hydro** (`struct`) – Structure of hydro data
- **Omax** (`integer`) – Maximum order of the SS realization, the default is 10
- **R2t** (`float`) – R^2 threshold (coefficient of determination) for the SS realization, where R^2 may range from 0 to 1, and the default is 0.95

Returns: **hydro** – Structure of hydro data with radiation IRF state space coefficients

Return type: struct

bemio.m

```
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[]);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```

excitationIRF Calculates the excitation impulse response function

functions.BEMIO.excitationIRF(*hydro, tEnd, nDt, nDw, wMin, wMax*)

Calculates the normalized excitation impulse response function:

$$\overline{K}_{e,i,\theta}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{X_i(\omega,\theta)e^{i\omega t}}{\rho g}d\omega$$

Default parameters can be used by inputting []. See `WEC-Sim\examples\BEMIO` for examples of usage.

Parameters:

- **hydro** (`struct`) – Structure of hydro data
- **tEnd** (`float`) – Calculation range for the IRF, where the IRF is calculated from t = 0 to tEnd, and the default is 100 s
- **nDt** (`float`) – Number of time steps in the IRF, the default is 1001
- **nDw** (`float`) – Number of frequency steps used in the IRF calculation (hydrodynamic coefficients are interpolated to correspond), the default is 1001
- **wMin** (`float`) – Minimum frequency to use in the IRF calculation, the default is the minimum frequency from the BEM data
- **wMax** (`float`) – Maximum frequency to use in the IRF calculation, the default is the maximum frequency from the BEM data

Returns: **hydro** – Structure of hydro data with excitation IRF

Return type: struct

bemio.m

```
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[]);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```

$$\overline{K}_i(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{X_i(\omega,\beta)}{\rho g}e^{i\omega t}d\omega$$

writeBEMIOH5 Writes the hydro data structure to a .h5 file.

functions.BEMIO.writeBEMIOH5(*hydro*)

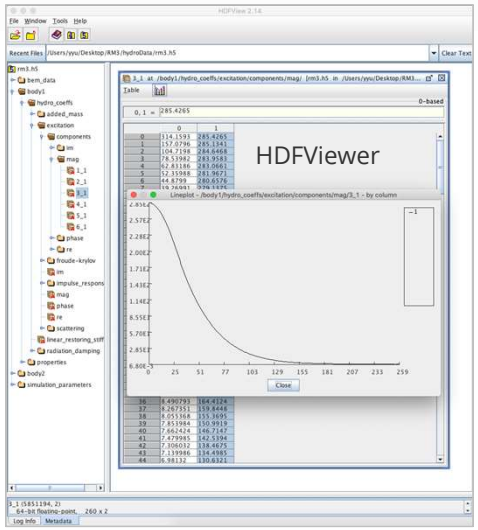
Writes the hydro data structure to a .h5 file.

See `WEC-Sim\tutorials\BEMIO` for examples of usage.

bemio.m

```
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[]);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```

Parameters: **hydro** (`[1 x 1] struct`) – Structure of hydro data that is written to `hydro.file`



plotBEMIO Plots the hydrodynamic data

functions.BEMIO.plotBEMIO(*varargin*)

Plots the added mass, radiation damping, radiation IRF, excitation force magnitude, excitation force phase, and excitation IRF for each body in the heave, surge and pitch degrees of freedom.

Usage: `plotBEMIO(hydro, hydro2, hydro3, ...)`

See `WEC-Sim\examples\BEMIO` for additional examples.

bemio.m

```
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[]);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```

Parameters: **varargin** (`struct(s)`) – The hydroData structure(s) created by the other BEMIO functions. One or more may be input.

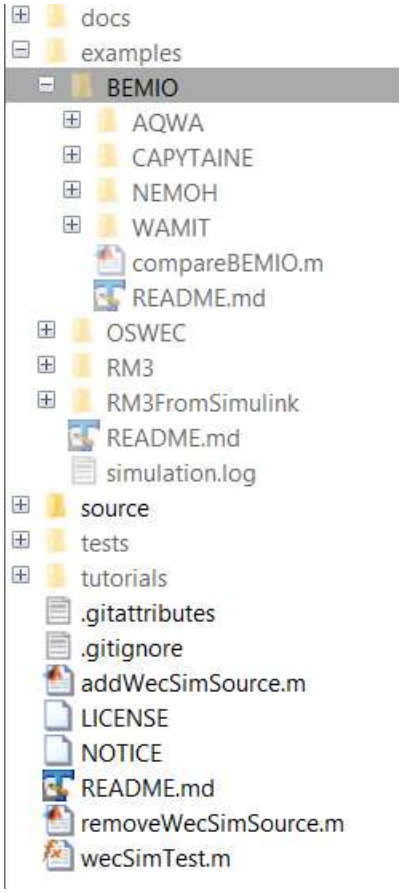
Examples and Usage

BEMIO tutorials in \WEC-Sim\examples\BEMIO

- WAMIT
- NEMOH
- Aqwa
- Capytaine
- compareBEMIO

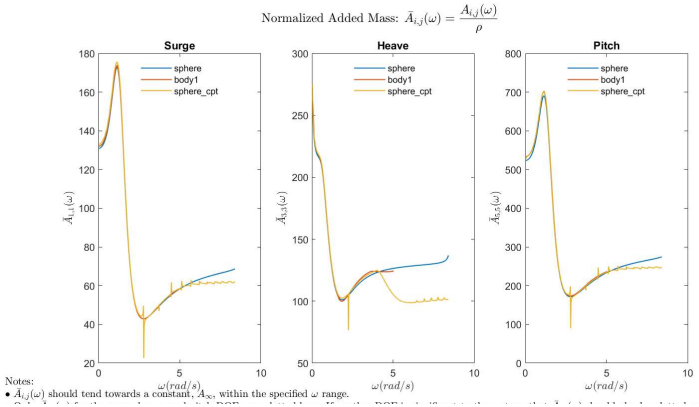
Data structures

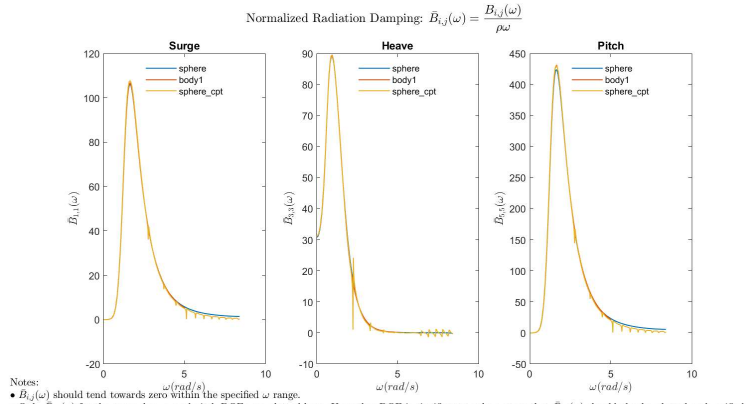
- BEMIO
- https://wec-sim.github.io/WEC-Sim/advanced_features.html#bemio
- .h5
- HDFVIEW: https://support.hdfgroup.org/products/java/hdfview/

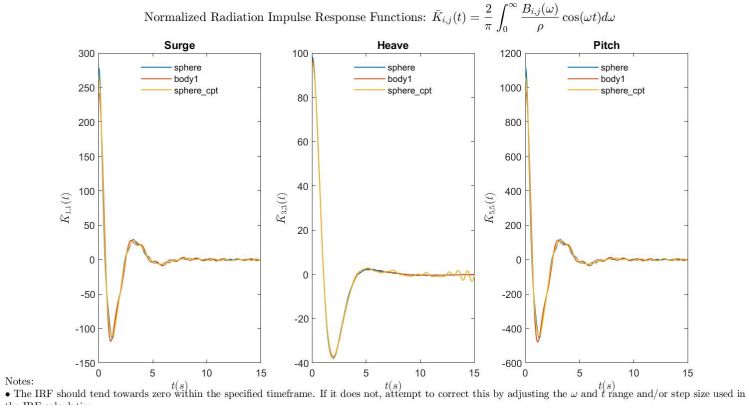


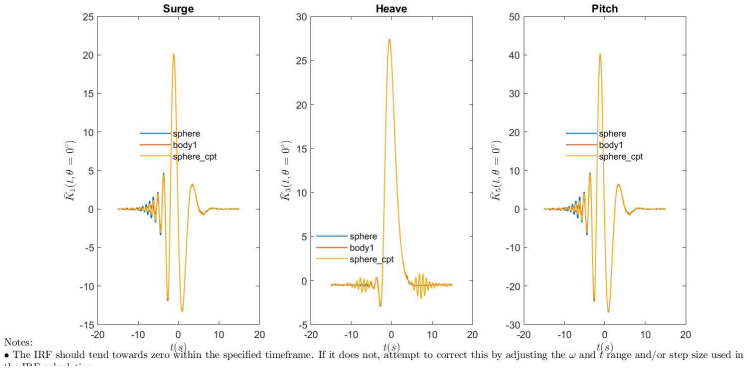
compareBEMIO

Sphere comparison available in: ***WEC-Sim\examples\BEMIO***









BEMIO Tutorial Practice

You can go to this folder in WEC-Sim to follow along:

\\WEC-Sim\examples\BEMIO\WAMIT\RM3

wecSimInputFile.m | bemio.m

```
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[2]);
5  hydro = radiationIRFSS(hydro,[],[]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
```

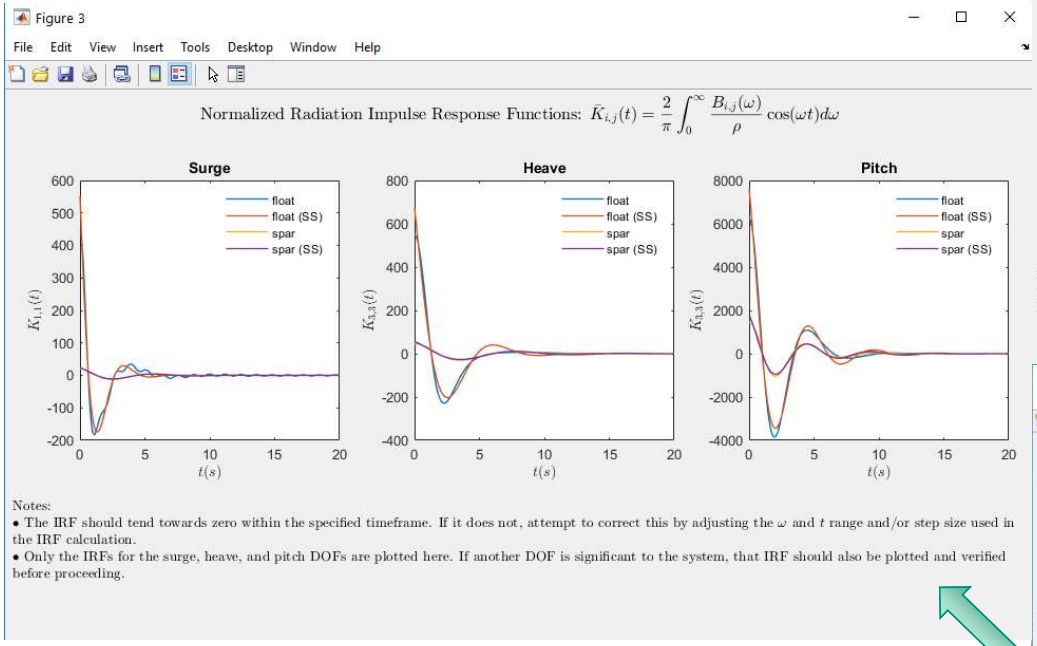
State Space Representation of IRF

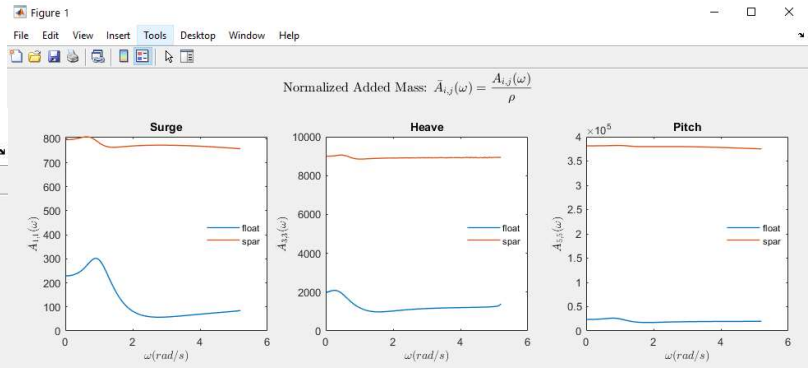
It is desirable to represent the radiation convolution integral in state space form. This has been shown to dramatically increase computational speeds and allow utilization of conventional control methods that rely on linear state space models.

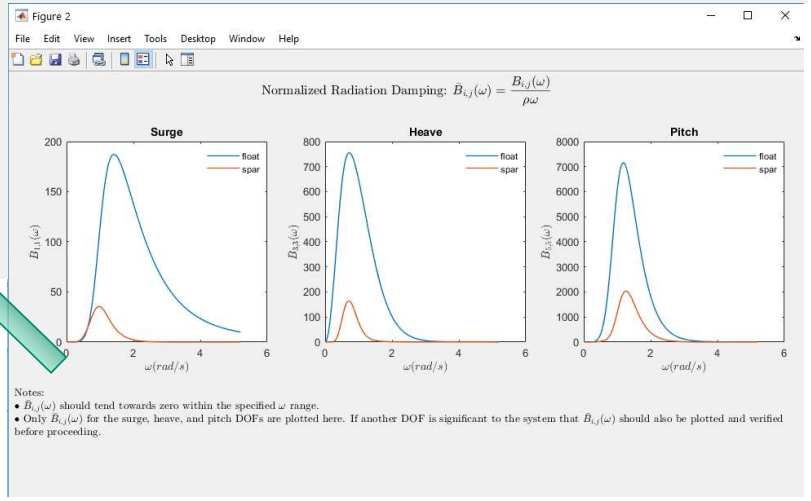
$$\int_0^t K_r(t-\tau)u(\tau)d\tau \quad \approx \quad \begin{array}{c} \dot{X}_r(t) = A_r X_r(t) + B_r u(t) \\ C_r X_r(t) + D_r u(t);\ X_r(0) = 0 \end{array}$$

An approximation will be made as K_r is solved from a set of partial differential equations where as a linear state space is constructed from a set of ordinary differential equations.

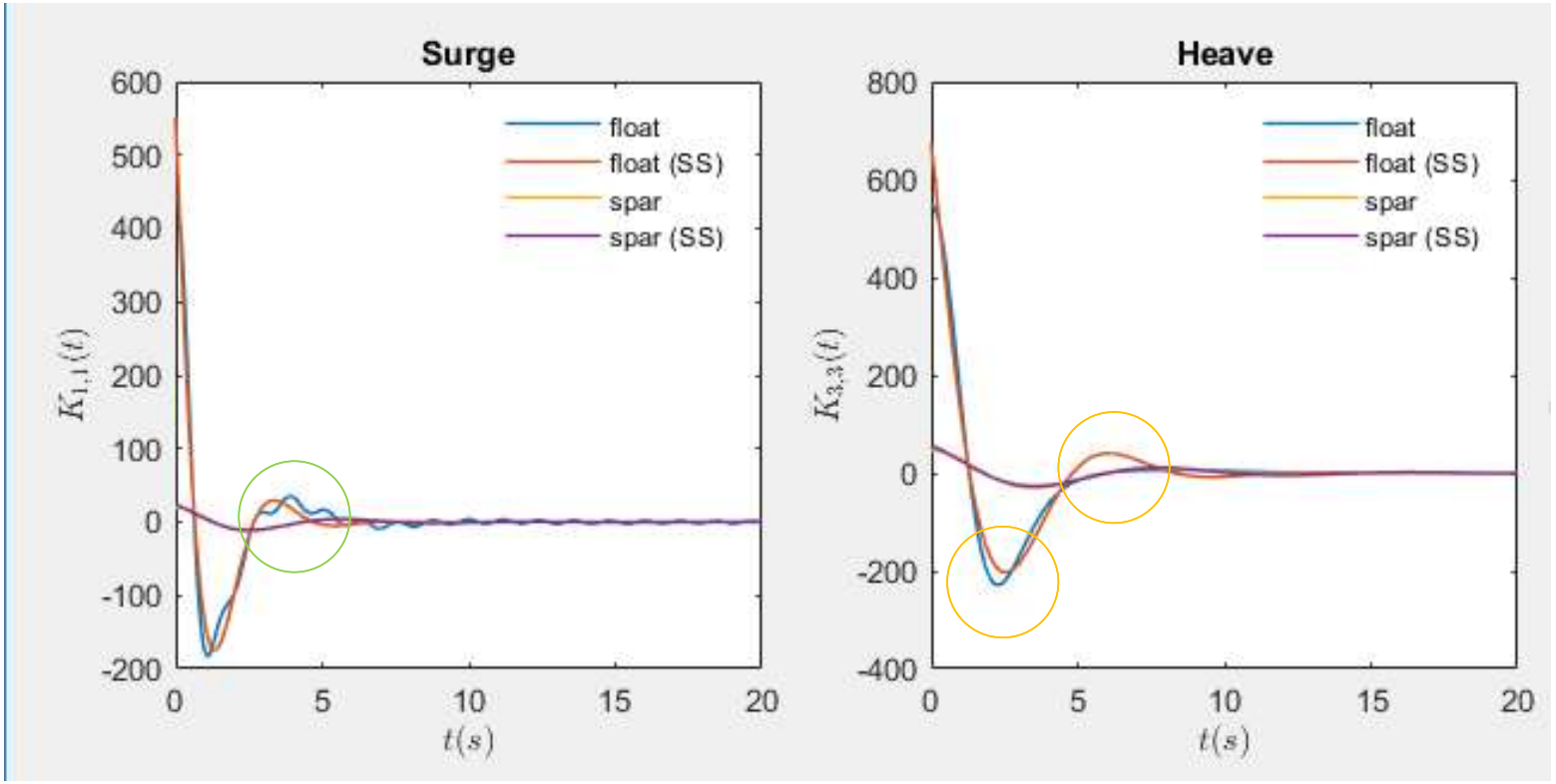
BEMIO Tutorial Practice







BEMIO Tutorial Practice

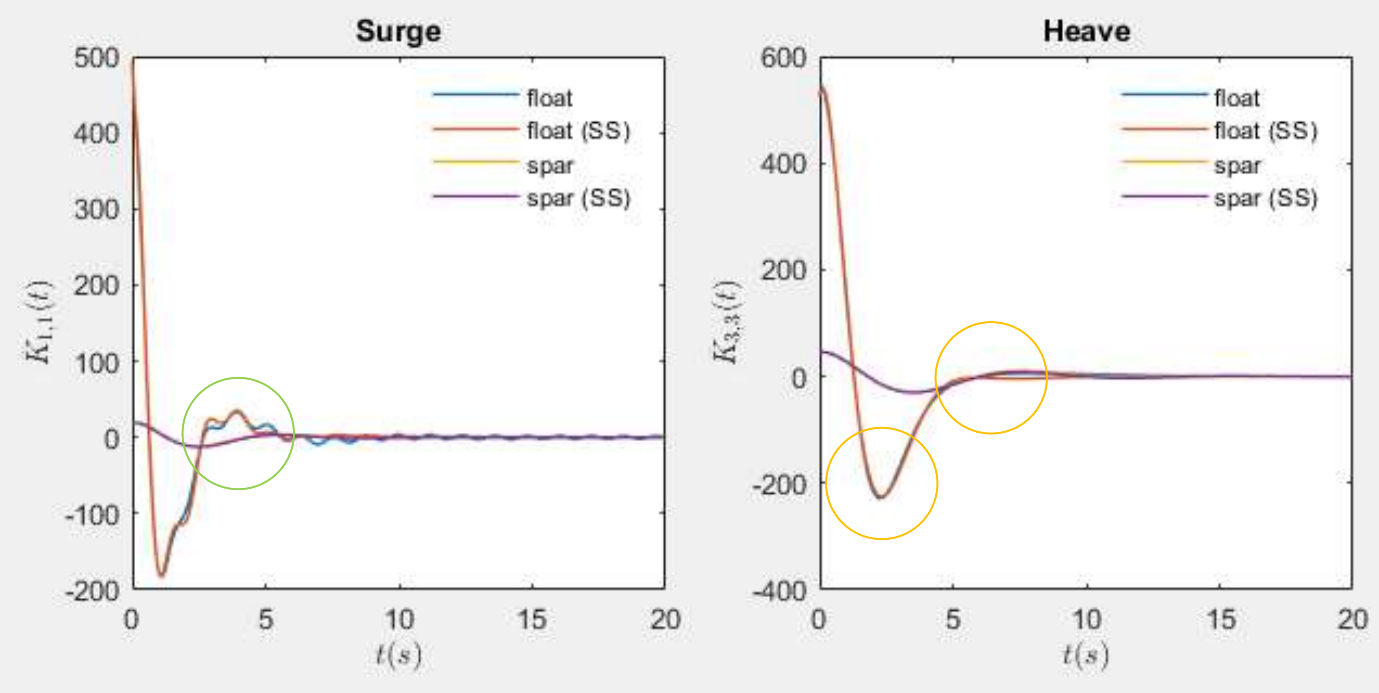


BEMIO Tutorial Practice

Let's increase the R2 threshold to 0.99
hydro = radiationIRFSS(hydro, [ ], [0.99])
Default is 0.95

bemio.m

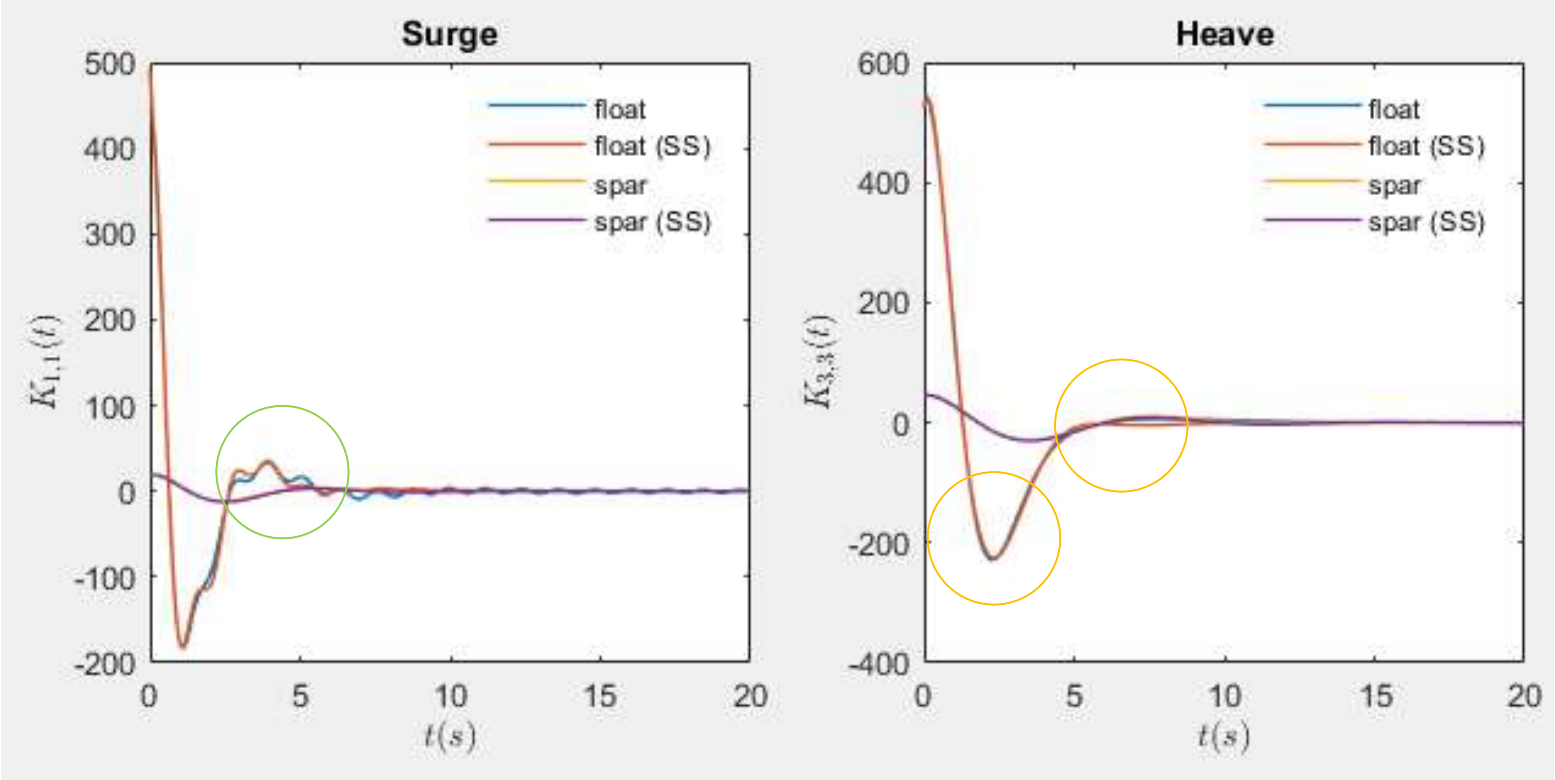
```
hydro = struct();

hydro = readWAMIT(hydro,'rm3.out',[]);
hydro = radiationIRF(hydro,20,[],[],[],[]);
hydro = radiationIRFSS(hydro,[],[0.99]);
hydro = excitationIRF(hydro,20,[],[],[],[]);
writeBEMIOH5(hydro)
plotBEMIO(hydro)
```



BEMIO Tutorial Practice

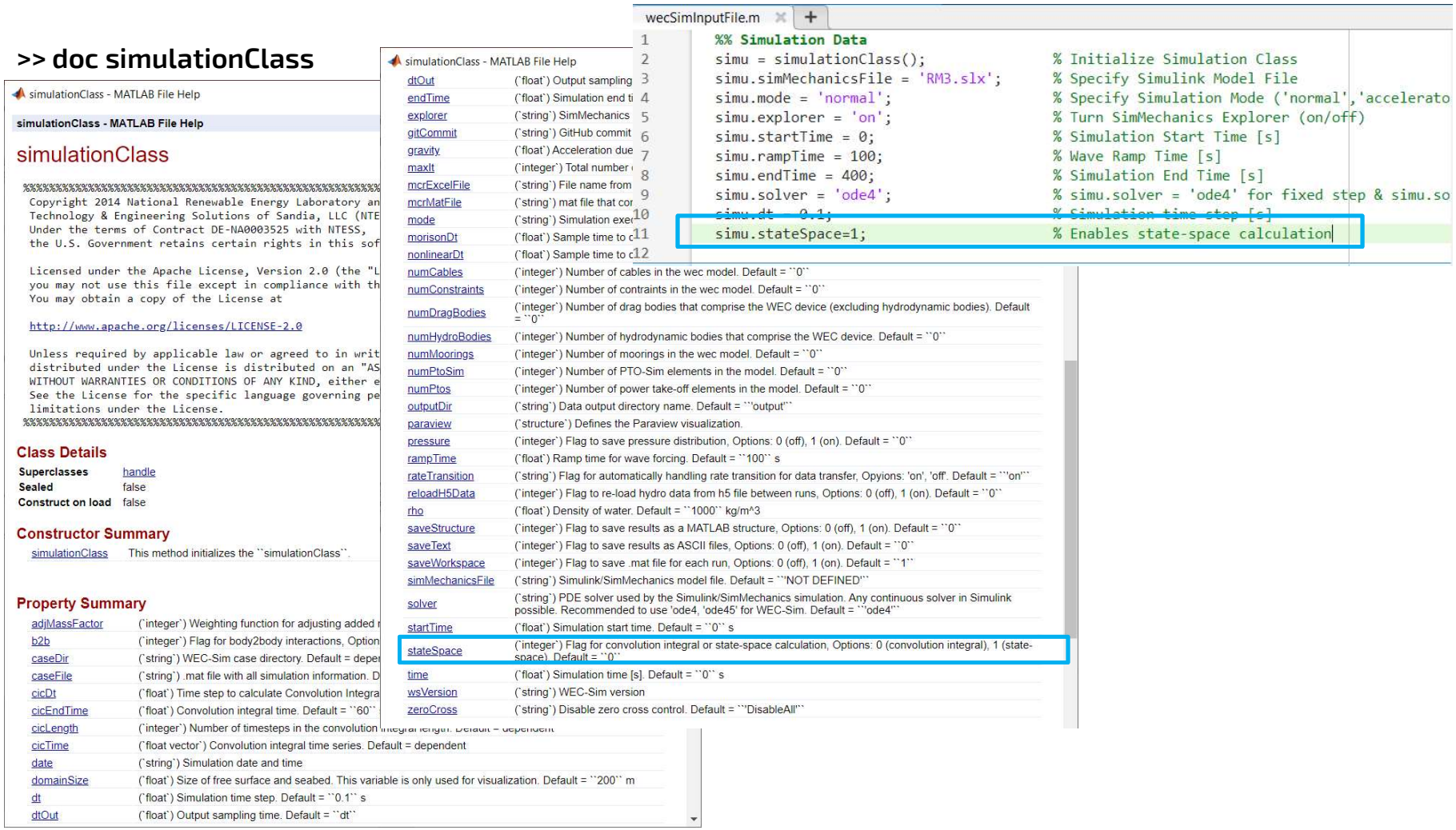
hydro = radiationIRFSS(hydro, [ ], [0.999])

```
bemio.m
1  hydro = struct();
2
3  hydro = readWAMIT(hydro,'rm3.out',[]);
4  hydro = radiationIRF(hydro,20,[],[],[],[]);
5  hydro = radiationIRFSS(hydro,[],[0.99]);
6  hydro = excitationIRF(hydro,20,[],[],[],[]);
7  writeBEMIOH5(hydro)
8  plotBEMIO(hydro)
9
```



BEMIO Tutorial Practice

>> doc simulationClass



simulationClass - MATLAB File Help

simulationClass - MATLAB File Help

simulationClass

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Copyright 2014 National Renewable Energy Laboratory an
Technology & Engineering Solutions of Sandia, LLC (NTE
Under the terms of Contract DE-NA0003525 with NTESS,
the U.S. Government retains certain rights in this sof

Licensed under the Apache License, Version 2.0 (the "L
you may not use this file except in compliance with th
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writ
distributed under the License is distributed on an "AS
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either e
See the License for the specific language governing pe
limitations under the License.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

simulationClass	This method initializes the ``simulationClass``.

Property Summary

adjMassFactor	(`integer`) Weighting function for adjusting added r
b2b	(`integer`) Flag for body2body interactions, Option
caseDir	(`string`) WEC-Sim case directory. Default = depe
caseFile	(`string`) .mat file with all simulation information. D
cicDt	(`float`) Time step to calculate Convolution Integra
cicEndTime	(`float`) Convolution integral time. Default = ``60``
cicLength	(`integer`) Number of timesteps in the convolution integral length. Default = dependent
cicTime	(`float vector`) Convolution integral time series. Default = dependent
date	(`string`) Simulation date and time
domainSize	(`float`) Size of free surface and seabed. This variable is only used for visualization. Default = ``200`` m
dt	(`float`) Simulation time step. Default = ``0.1`` s
dtOut	(`float`) Output sampling time. Default = ``dt``

simulationClass - MATLAB File Help

dtOut	(`float`) Output sampling
endTime	(`float`) Simulation end ti
explorer	(`string`) SimMechanics
gitCommit	(`string`) GitHub commit
gravity	(`float`) Acceleration due
maxIt	(`integer`) Total number
mcrExcelFile	(`string`) File name from
mcrMatFile	(`string`) mat file that cor
mode	(`string`) Simulation exe
morisonDt	(`float`) Sample time to c
nonlinearDt	(`float`) Sample time to c
numCables	(`integer`) Number of cables in the wec model. Default = ``0``
numConstraints	(`integer`) Number of contraints in the wec model. Default = ``0``
numDragBodies	(`integer`) Number of drag bodies that comprise the WEC device (excluding hydrodynamic bodies). Default = ``0``
numHydroBodies	(`integer`) Number of hydrodynamic bodies that comprise the WEC device. Default = ``0``
numMoorings	(`integer`) Number of moorings in the wec model. Default = ``0``
numPtoSim	(`integer`) Number of PTO-Sim elements in the model. Default = ``0``
numPtos	(`integer`) Number of power take-off elements in the model. Default = ``0``
outputDir	(`string`) Data output directory name. Default = ```output```
paraview	(`structure`) Defines the Paraview visualization.
pressure	(`integer`) Flag to save pressure distribution, Options: 0 (off), 1 (on). Default = ``0``
rampTime	(`float`) Ramp time for wave forcing. Default = ``100`` s
rateTransition	(`string`) Flag for automatically handling rate transition for data transfer, Opyions: 'on', 'off'. Default = ```on```
reloadH5Data	(`integer`) Flag to re-load hydro data from h5 file between runs, Options: 0 (off), 1 (on). Default = ``0``
rho	(`float`) Density of water. Default = ``1000`` kg/m^3
saveStructure	(`integer`) Flag to save results as a MATLAB structure, Options: 0 (off), 1 (on). Default = ``0``
saveText	(`integer`) Flag to save results as ASCII files, Options: 0 (off), 1 (on). Default = ``0``
saveWorkspace	(`integer`) Flag to save .mat file for each run, Options: 0 (off), 1 (on). Default = ``1``
simMechanicsFile	(`string`) Simulink/SimMechanics model file. Default = ```NOT DEFINED```
solver	(`string`) PDE solver used by the Simulink/SimMechanics simulation. Any continuous solver in Simulink possible. Recommended to use 'ode4, 'ode45' for WEC-Sim. Default = ```ode4```
startTime	(`float`) Simulation start time. Default = ``0`` s
stateSpace	(`integer`) Flag for convolution integral or state-space calculation, Options: 0 (convolution integral), 1 (state-space). Default = ``0``
time	(`float`) Simulation time [s]. Default = ``0`` s
wsVersion	(`string`) WEC-Sim version
zeroCross	(`string`) Disable zero cross control. Default = ```DisableAll```

wecSimInputFile.m

```
%% Simulation Data
simu = simulationClass();              % Initialize Simulation Class
simu.simMechanicsFile = 'RM3.slx';     % Specify Simulink Model File
simu.mode = 'normal';                  % Specify Simulation Mode ('normal','accelerato
simu.explorer = 'on';                  % Turn SimMechanics Explorer (on/off)
simu.startTime = 0;                    % Simulation Start Time [s]
simu.rampTime = 100;                   % Wave Ramp Time [s]
simu.endTime = 400;                    % Simulation End Time [s]
simu.solver = 'ode4';                  % simu.solver = 'ode4' for fixed step & simu.so
simu.dt = 0.1;                         % Simulation time step [s]
simu.stateSpace=1;                     % Enables state-space calculation
```

BEMIO Tutorial Practice

How does changing the upper wave frequency limit on the IRF?

Perhaps the BEM hydrodynamic data is poor and needs to be cut off.

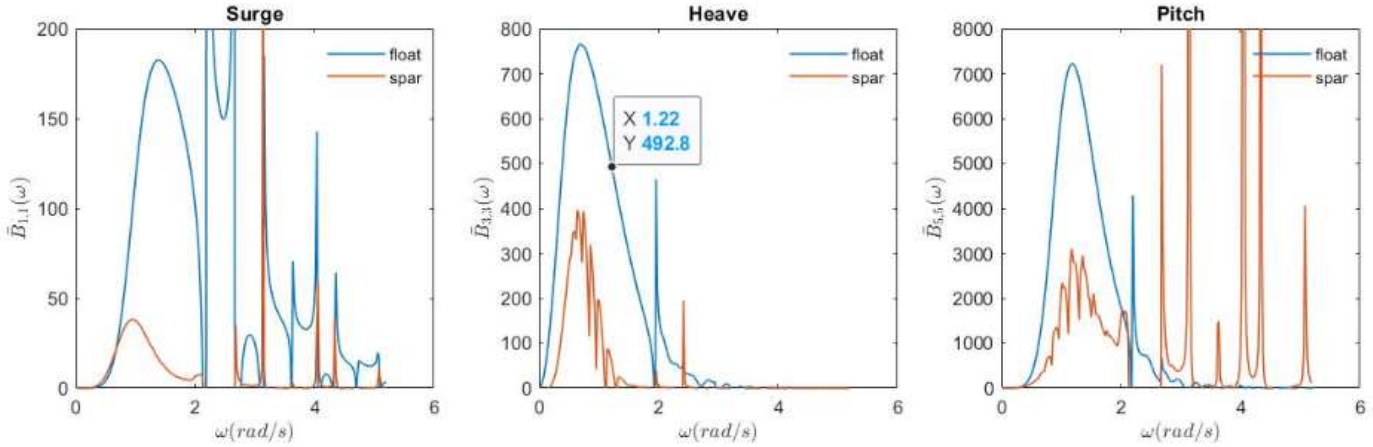
wecSimInputFile.m | bemio.m

```
1   hydro = struct();
2
3   hydro = readWAMIT(hydro,'rm3.out',[]);
4   hydro = radiationIRF(hydro,20,[],[],[],[2]);
5   hydro = radiationIRFSS(hydro,[],[]);
6   hydro = excitationIRF(hydro,20,[],[],[],[]);
7   writeBEMIOH5(hydro)
8   plotBEMIO(hydro)
```

BEMIO Tutorial Practice

Depending on the BEM solver, mesh quality, and size of your device the hydrodynamic coefficients can be reported with noise and nonphysical solutions.

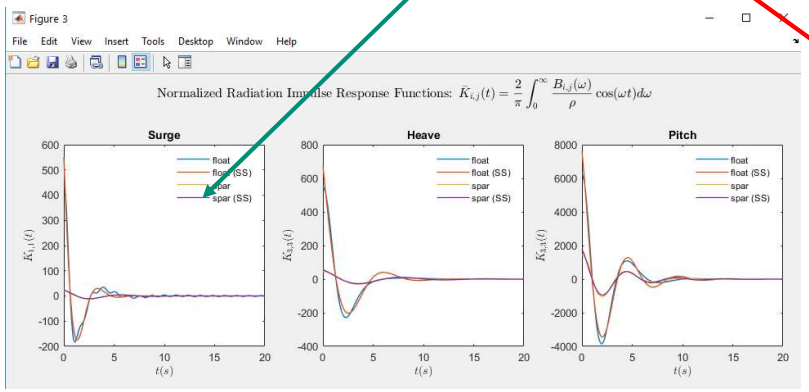
Normalized Radiation Damping: $\bar{B}_{i,j}(\omega) = \frac{B_{i,j}(\omega)}{\rho\omega}$

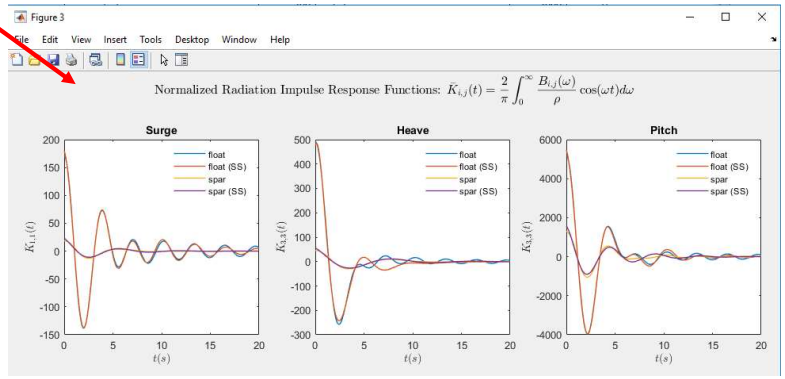


Notes:
- $\bar{B}_{i,j}(\omega)$ should tend towards zero within the specified ω range.
- Only $\bar{B}_{i,j}(\omega)$ for the surge, heave, and pitch DOFs are plotted here. If another DOF is significant to the system that $\bar{B}_{i,j}(\omega)$ should also be plotted and verified before proceeding.

Since the BEM solution defines the WEC response, poor BEM data can lead to unstable WEC-Sim simulations.

BEMIO Tutorial Practice

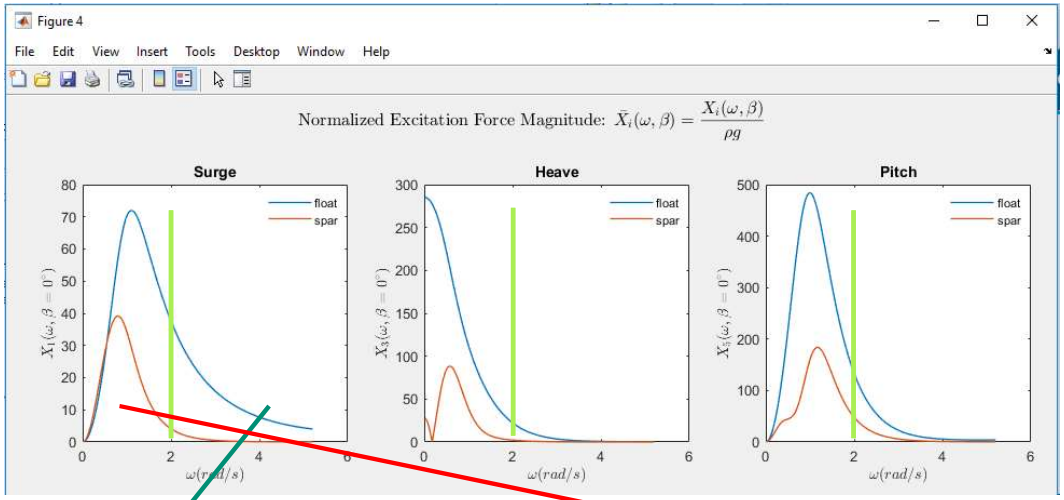


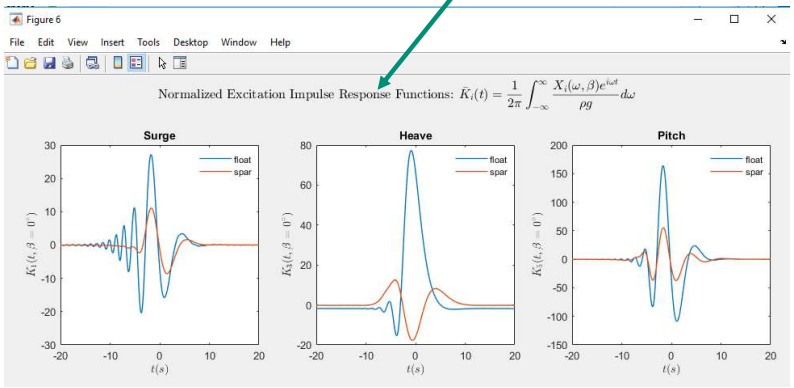


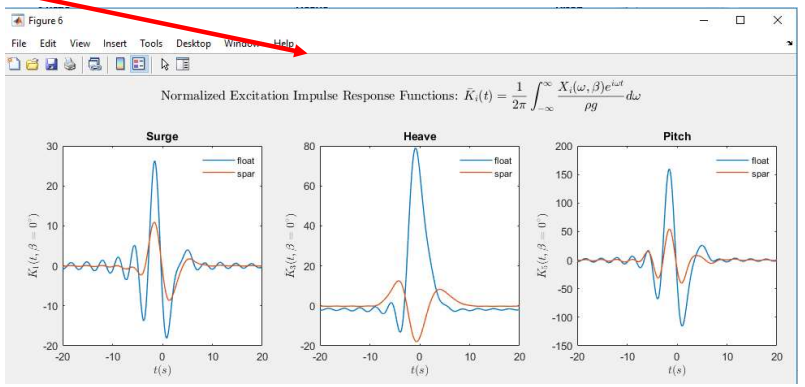
Cut-off frequency is sufficiently high

Cut-off frequency is too low

BEMIO Tutorial Practice





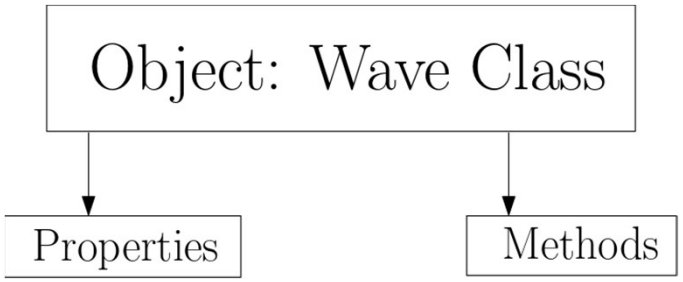


Cut-off frequency is sufficiently high

Cut-off frequency is too low

Wave Class

Wave Class: Overview



Choice of wave inputs to the system

```
%% Wave Information
% % noWaveCIC, no waves with radiation CIC
% waves = waveClass('noWaveCIC');       % Initialize Wave Class and Specify Type
```
Still Water

```
% Regular Waves
waves = waveClass('regular');          % Initialize Wave Class and Specify Type
waves.height = 2.5;                    % Wave Height [m]
waves.period = 8;                      % Wave Period [s]
```
Regular Waves

```
% % Regular Waves with CIC
% waves = waveClass('regularCIC');          % Initialize Wave Class and Specify Type
% waves.height = 2.5;                       % Wave Height [m]
% waves.period = 8;                         % Wave Period [s]
```
Regular Waves with Radiation Force Convolution

```
% % Irregular Waves using PM Spectrum
% waves = waveClass('irregular');           % Initialize Wave Class and Specify Type
% waves.height = 2.5;                       % Significant Wave Height [m]
% waves.period = 8;                         % Peak Period [s]
% waves.spectrumType = 'PM';                % Specify Wave Spectrum Type
```
Pierson-Moskowitz

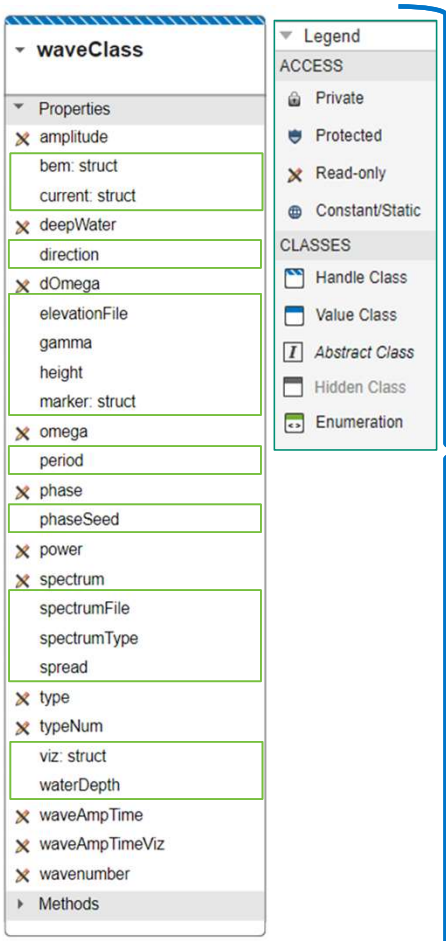
```
% % Irregular Waves using JS Spectrum with Equal Energy and Seeded Phase
% waves = waveClass('irregular');           % Initialize Wave Class and Specify Type
% waves.height = 2.5;                       % Significant Wave Height [m]
% waves.period = 8;                         % Peak Period [s]
% waves.spectrumType = 'JS';                % Specify Wave Spectrum Type
% waves.bem.option = 'EqualEnergy';         % Uses 'EqualEnergy' bins (default)
% waves.phaseSeed = 1;                      % Phase is seeded so eta is the same
```
JONSWAP

```
% % Irregular Waves using PM Spectrum with Traditional and State Space
% waves = waveClass('irregular');           % Initialize Wave Class and Specify Type
% waves.height = 2.5;                       % Significant Wave Height [m]
% waves.period = 8;                         % Peak Period [s]
% waves.spectrumType = 'PM';                % Specify Wave Spectrum Type
% simu.stateSpace = 1;                      % Turn on State Space
% waves.bem.option = 'Traditional';         % Uses 1000 frequencies
```

```
% % Irregular Waves with imported spectrum
% waves = waveClass('spectrumImport');      % Create the Wave Variable and Specify Type
% waves.spectrumFile = 'spectrumData.mat';  % Name of User-Defined Spectrum File [:,2] = [f, Sf]

% % Waves with imported wave elevation time-history
% waves = waveClass('elevationImport');          % Create the Wave Variable and Specify Type
% waves.elevationFile = 'elevationData.mat';     % Name of User-Defined Time-Series File [:,2] = [time, eta]
```
User Import

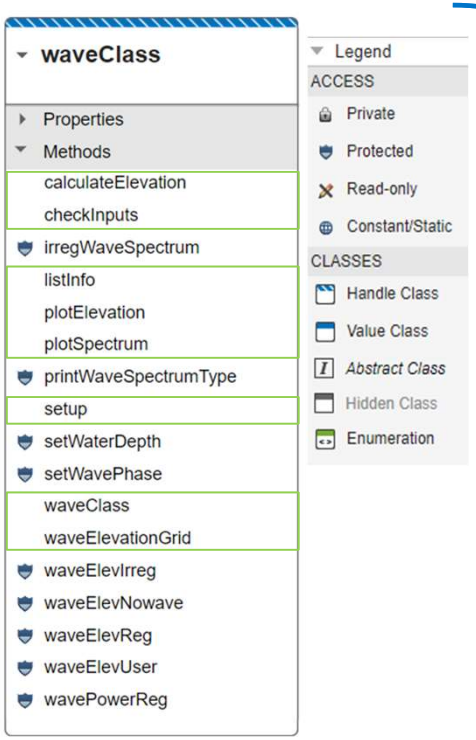
Wave Class: Properties

Wave Type	Required Properties
`noWave`	`waves.period`
`noWaveCIC`	
`regular`	`waves.height` , `waves.period`
`regularCIC`	`waves.height` , `waves.period`
`irregular`	`waves.height` , `waves.period` , `waves.spectrumType`
`spectrumImport`	`waves.spectrumFile`
`elevationImport`	`waves.elevationFile`



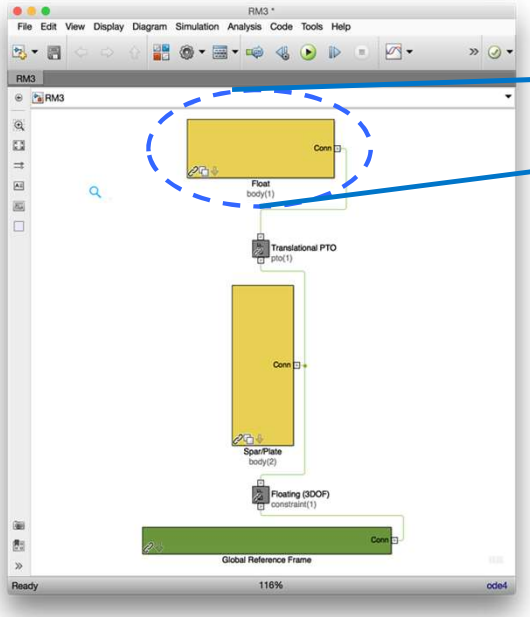

```
waves =

  waveClass with properties:

              bem: [1×1 struct]
          current: [1×1 struct]
        direction: 0
    elevationFile: 'NOT DEFINED'
            gamma: []
           height: 2.5000
           marker: [1×1 struct]
           period: 8
        phaseSeed: 0
     spectrumFile: 'NOT DEFINED'
     spectrumType: 'NOT DEFINED'
              viz: [1×1 struct]
       waterDepth: []
           spread: 1
        amplitude: []
        deepWater: []
           dOmega: 0
            omega: []
            phase: 0
            power: []
         spectrum: []
             type: 'regular'
          typeNum: 10
      waveAmpTime: []
   waveAmpTimeViz: []
       wavenumber: []
```

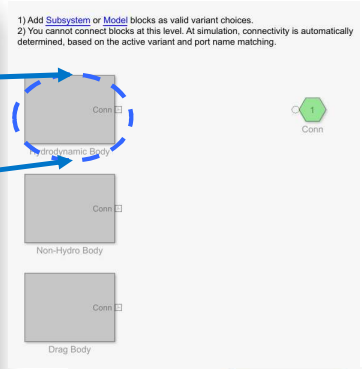
Wave Class: Methods




```
Methods for class waveClass:

calculateElevation  listInfo            plotSpectrum        waveClass
checkInputs         plotElevation       setup               waveElevationGrid

Methods of waveClass inherited from handle.
```

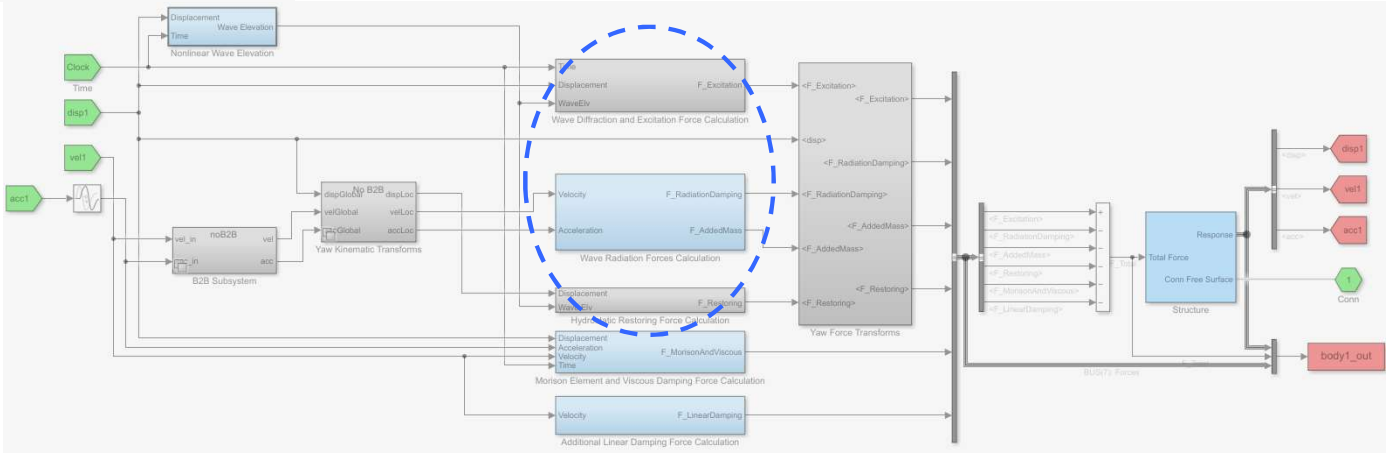
Wave Class Simulink





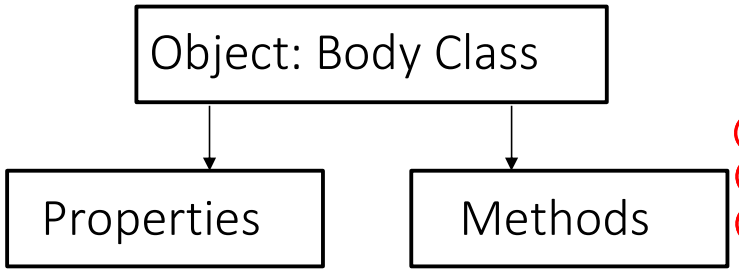
Simulink Applies (based on waveClass inputs):

- Wave Diffraction and Excitation Force
- Wave Radiation Force
  - Constant Coefficient
  - Convolution Integral
  - State Space
- Hydrostatic Restoring Force



Body Class

Body Class: Overview



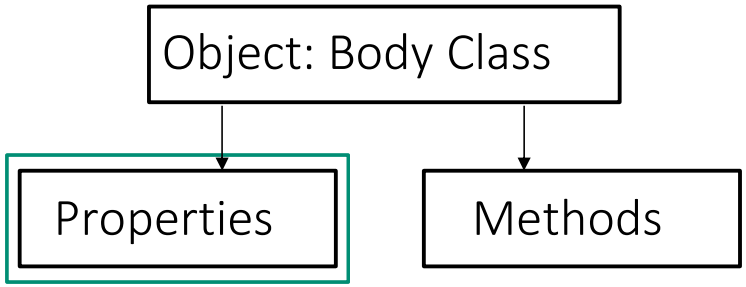
OSWEC Example

```
%% Body Data
% Flap
body(1) = bodyClass('hydroData/oswec.h5');      % Initialize bodyClass for Flap
body(1).geometryFile = 'geometry/flap.stl';     % Geometry File
body(1).mass = 127000;                          % User-Defined mass [kg]
body(1).inertia = [1.85e6 1.85e6 1.85e6];       % Moment of Inertia [kg-m^2]
body(1).initial.displacement= [0, 0, 0.1];      % Initial displacement [x, y, z] m
body(1).initial.axis        = [0, 1, 0];        % Axis of initial angle disp.
body(1).initial.angle       = 1;                % Initial ang. disp. in rad.
% Base
body(2) = bodyClass('hydroData/oswec.h5');      % Initialize bodyClass for Base
body(2).geometryFile = 'geometry/base.stl';     % Geometry File
body(2).mass = 999;                             % Placeholder mass for a fixed body
body(2).inertia = [999 999 999];                % Placeholder inertia for a fixed body
```

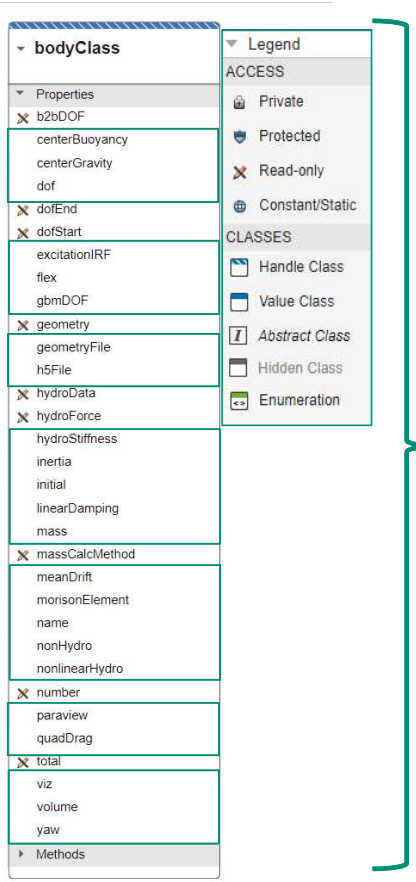
1. Generate Body Object
2. Identify geometry file in .stl format
3. Specify mass and moment of inertia properties,
4. Specify initial position initial cartesian position of the center of the center of gravity and initial angular orientation if different from the geometry file.

The definition of linear and quadratic damping parameters for the heave mode in the *wecSimInputFile.m* for the OSWEC example.

Body Class: Properties

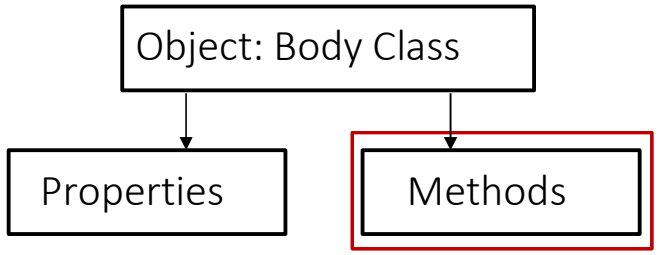


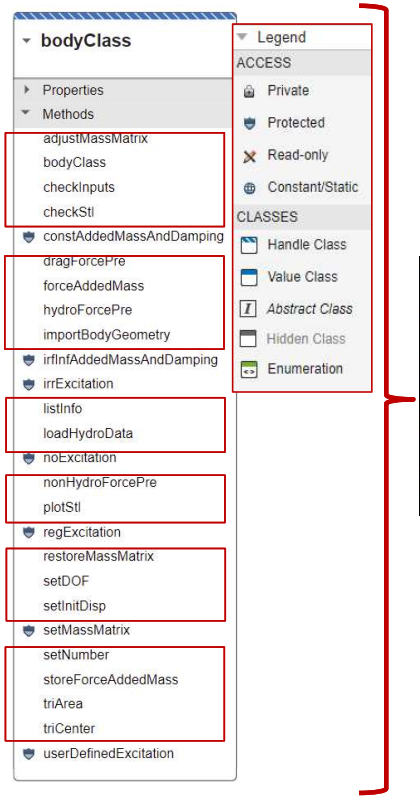
The properties inside green rectangles can be accessed by the user




```
>> body
body =
  1×2 bodyClass array with properties:

    centerBuoyancy
    centerGravity
    dof
    excitationIRF
    flex
    gbmDOF
    geometryFile
    h5File
    hydroStiffness
    inertia
    initial
    largeXYDisplacement
    linearDamping
    mass
    meanDrift
    morisonElement
    name
    nonHydro
    nonlinearHydro
    quadDrag
    paraview
    viz
    volume
    yaw
    dofEnd
    dofStart
    hydroData
    b2bDOF
    hydroForce
    massCalcMethod
    number
    total
    geometry
```

Body Class: Methods

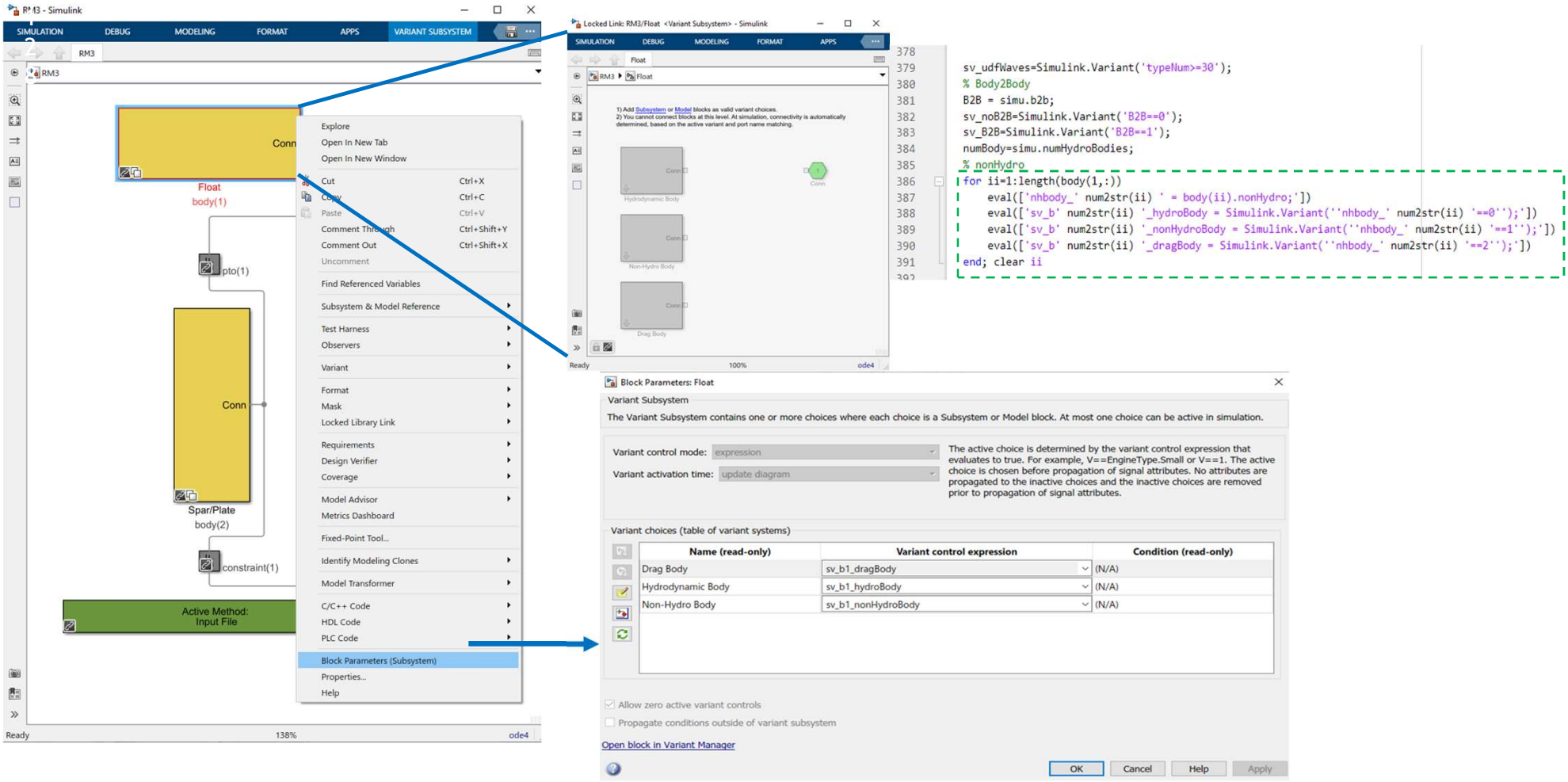




The methods inside red rectangles can be accessed by the user

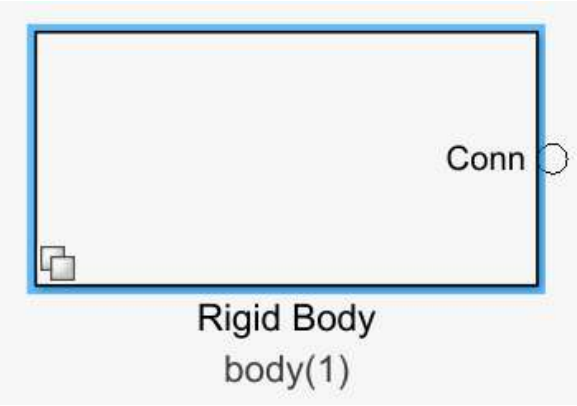
```
>> methods(body)

Methods for class bodyClass:

adjustMassMatrix     checkStl            hydroForcePre       loadHydroData       restoreMassMatrix    setNumber            triCenter
bodyClass            dragForcePre        importBodyGeometry  nonHydroForcePre    setDOF               storeForceAddedMass
checkInputs          forceAddedMass      listInfo            plotStl             setInitDisp          triArea

Methods of bodyClass inherited from handle.
```

Variant Subsystems



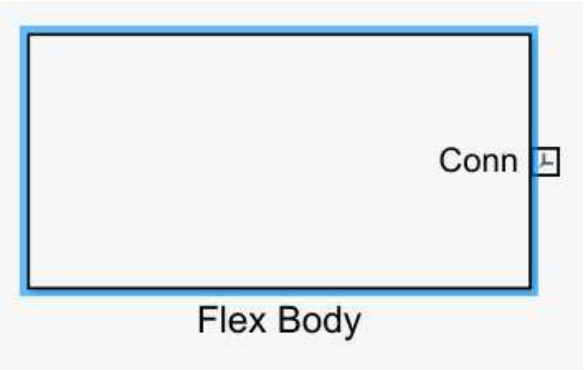
Body Blocks

Rigid body block



Rigid bodies that move in 6 DOF (surge, sway, heave, roll, pitch, yaw)

Flexible body block

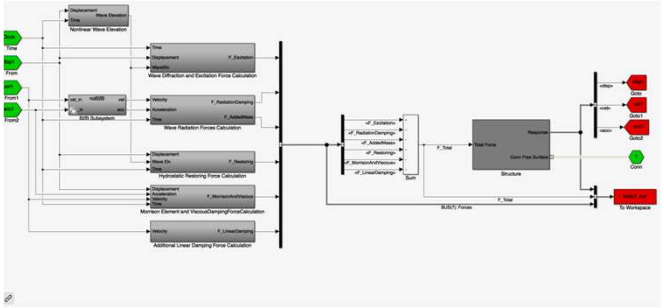


Flexible bodies with additional movement modes defined in BEM

*** See Advanced Features → Generalized Body Modes for more information on the flexible body block.**

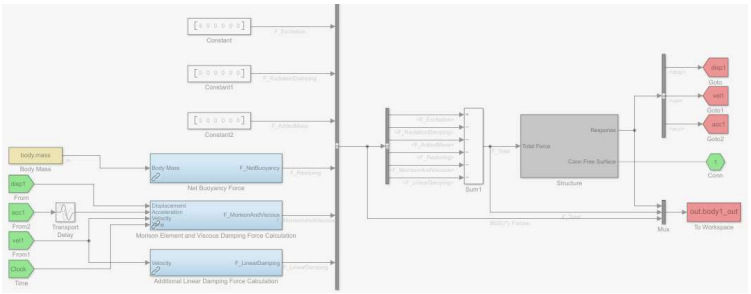
Rigid Body Block

Hydrodynamic body block



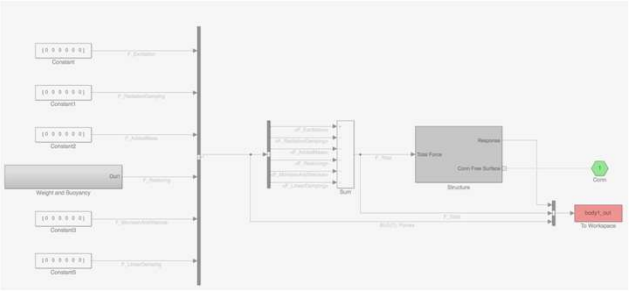
Includes blocks for calculating all the different forcing terms

Drag body block



Wave exerted forces are zero, but weight, buoyancy, and drag forces calculated

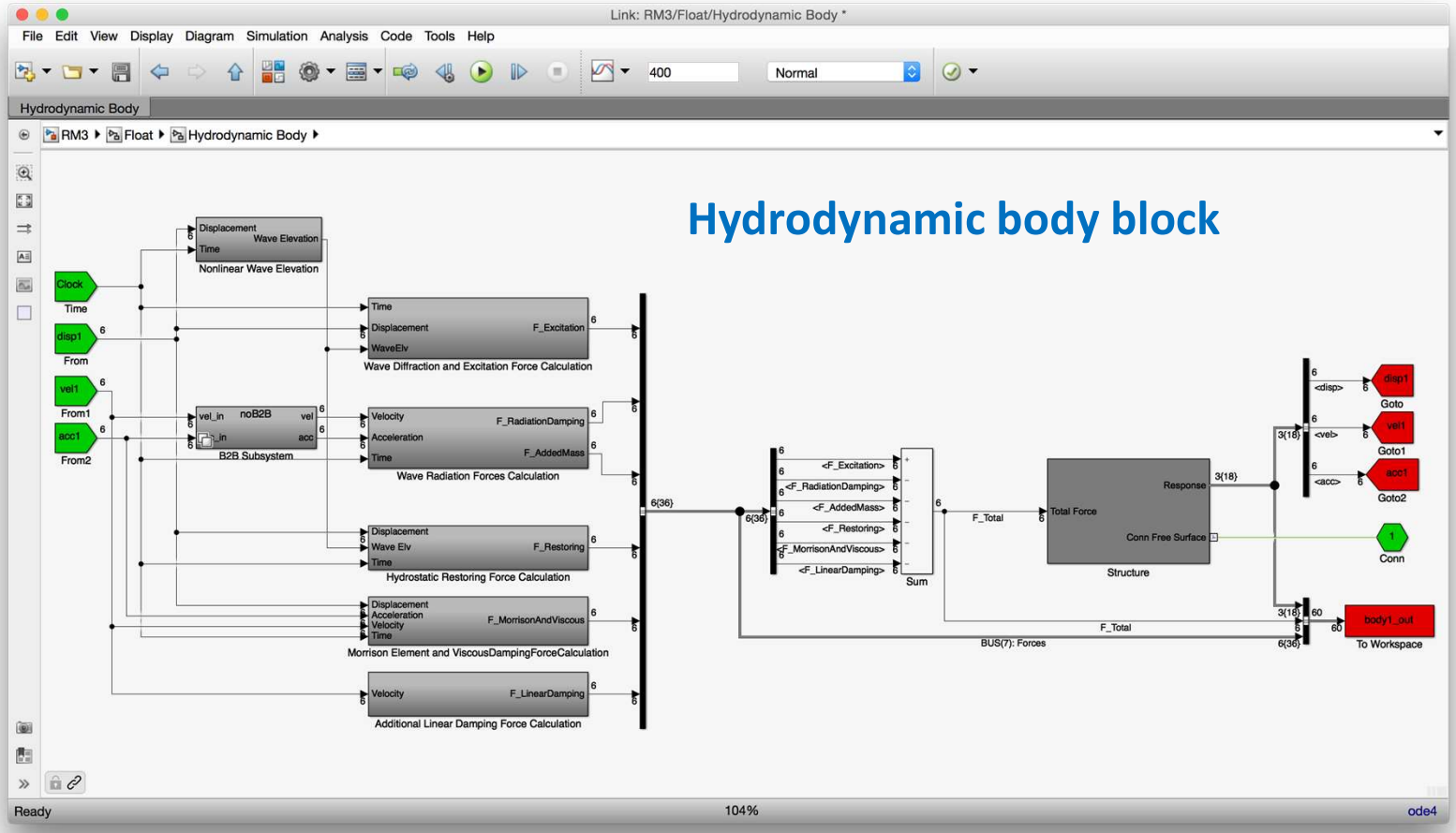
Non- hydrodynamic body block



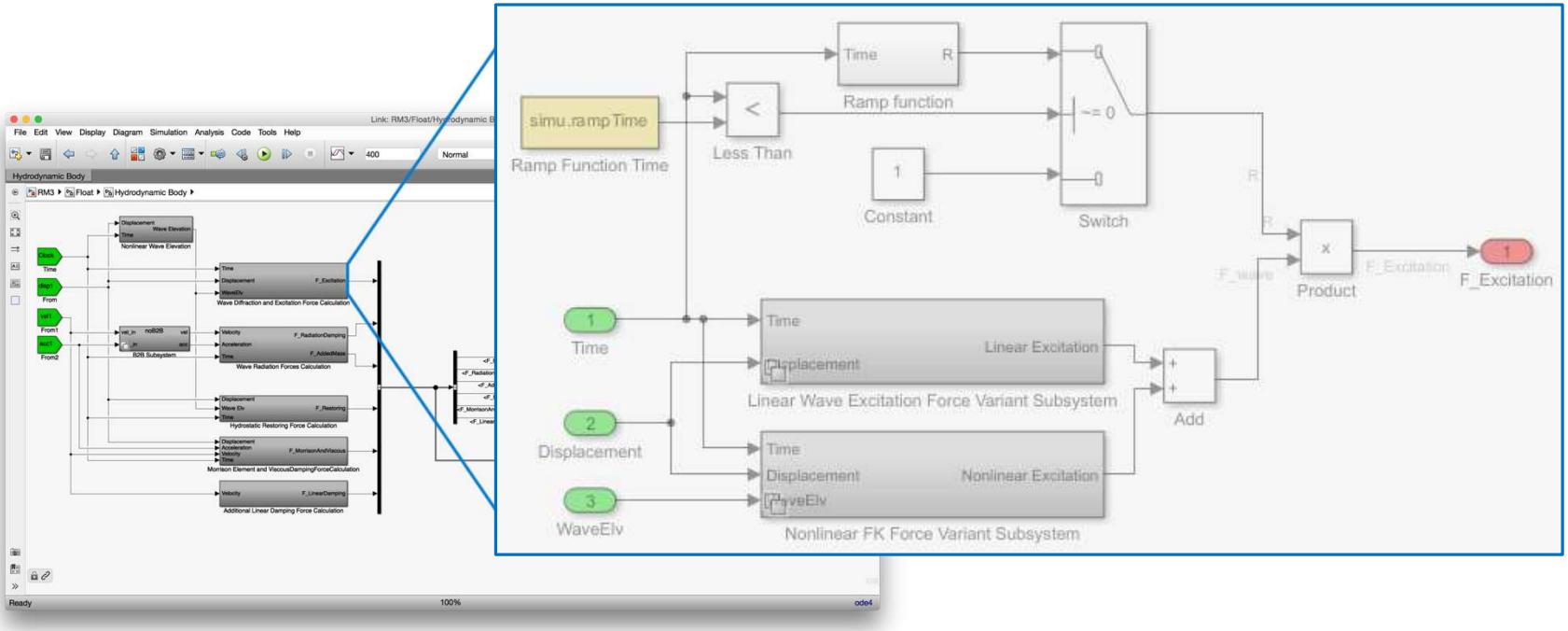
Everything is zero except for weight and buoyancy

Note: Connection forces between multiple bodies from the joint/PTO are handled by **Simscape Multibody**

Body Force Dimension Display



Body Class: Excitation Force

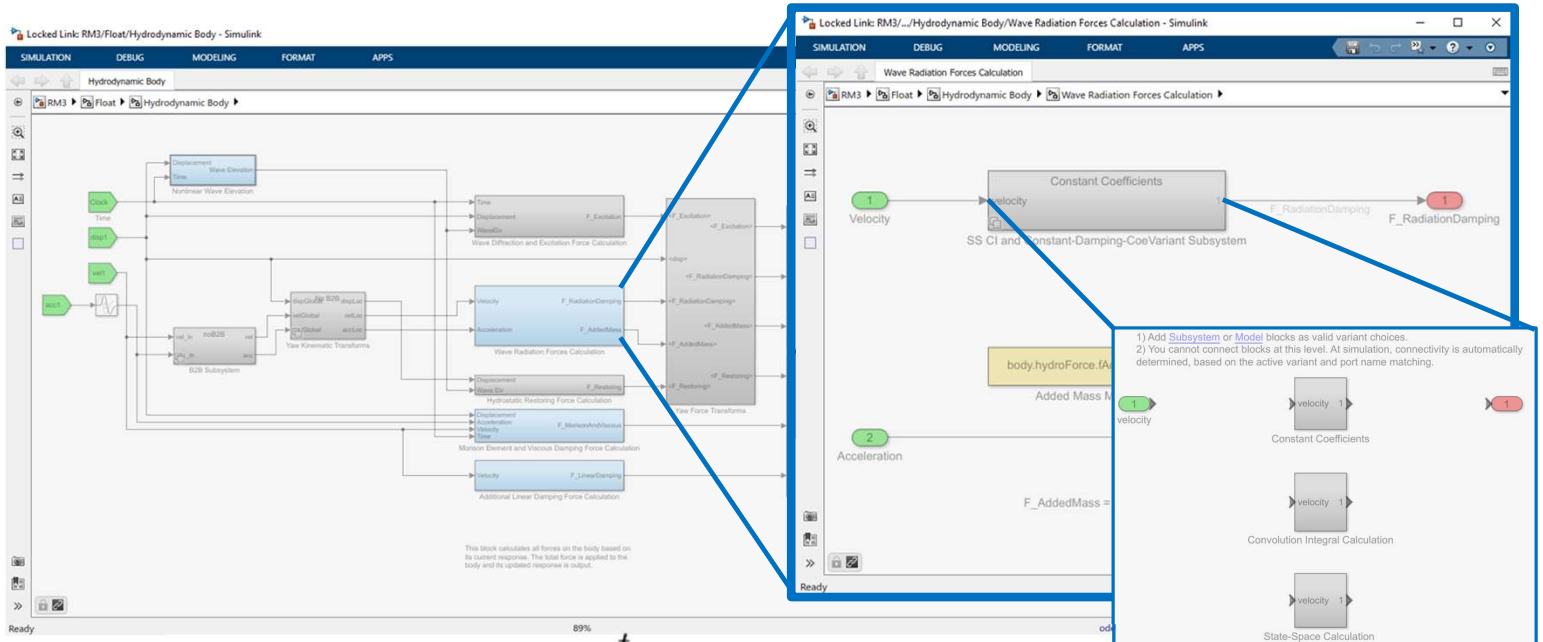


$$f_{ex}(t) = Re\left[R_f F_x(\omega_r) e^{i(\omega_r t+\phi)} \int_0^{\infty} \sqrt{2S(\omega_r)d\omega_r}\right]$$

$$f_{ex}(t) = \int_0^{t-\tau} \eta(\tau) h_f(t-\tau) d\tau$$

Hydrodynamic Body Block:

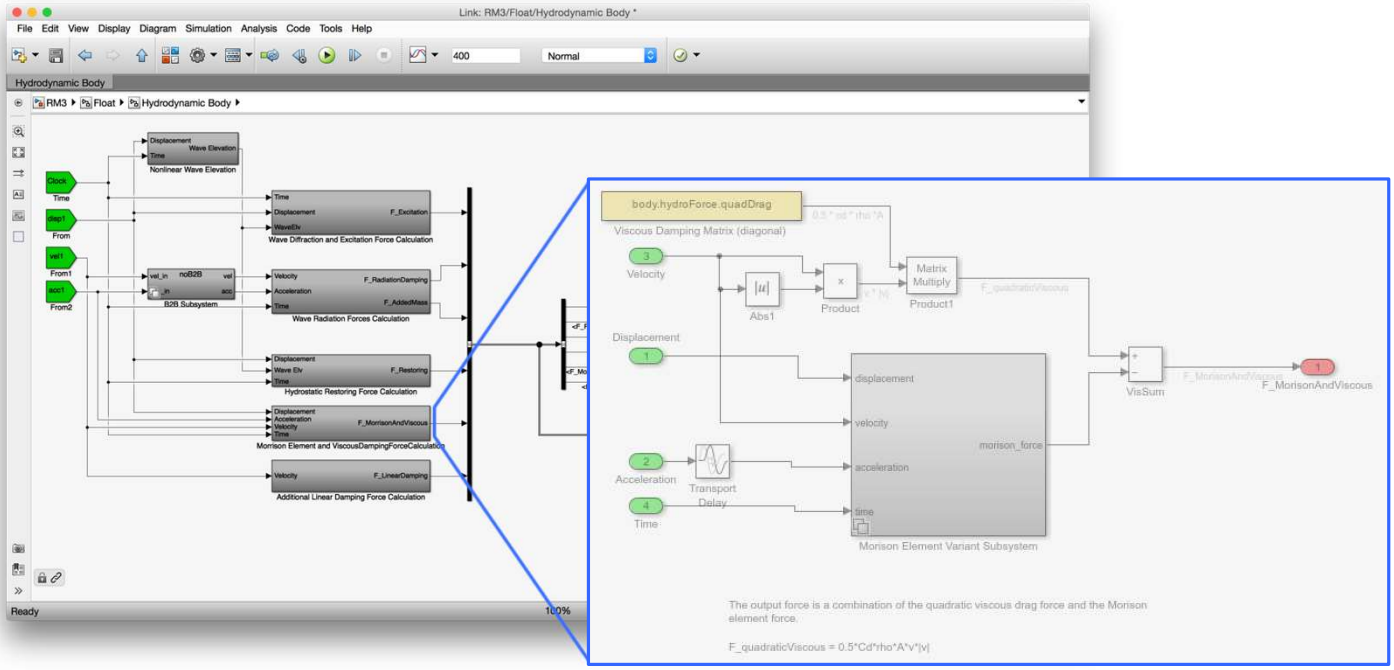
Wave Radiation Force Calculation Block



$$F_{rad}(t) = -A_\infty \ddot{X} - \int_0^t K_r(t-\tau)\dot{X}(\tau)d\tau$$

$$K_r(t) = \frac{2}{\pi}\int_0^\infty B(\omega)cos(\omega t)d\omega$$

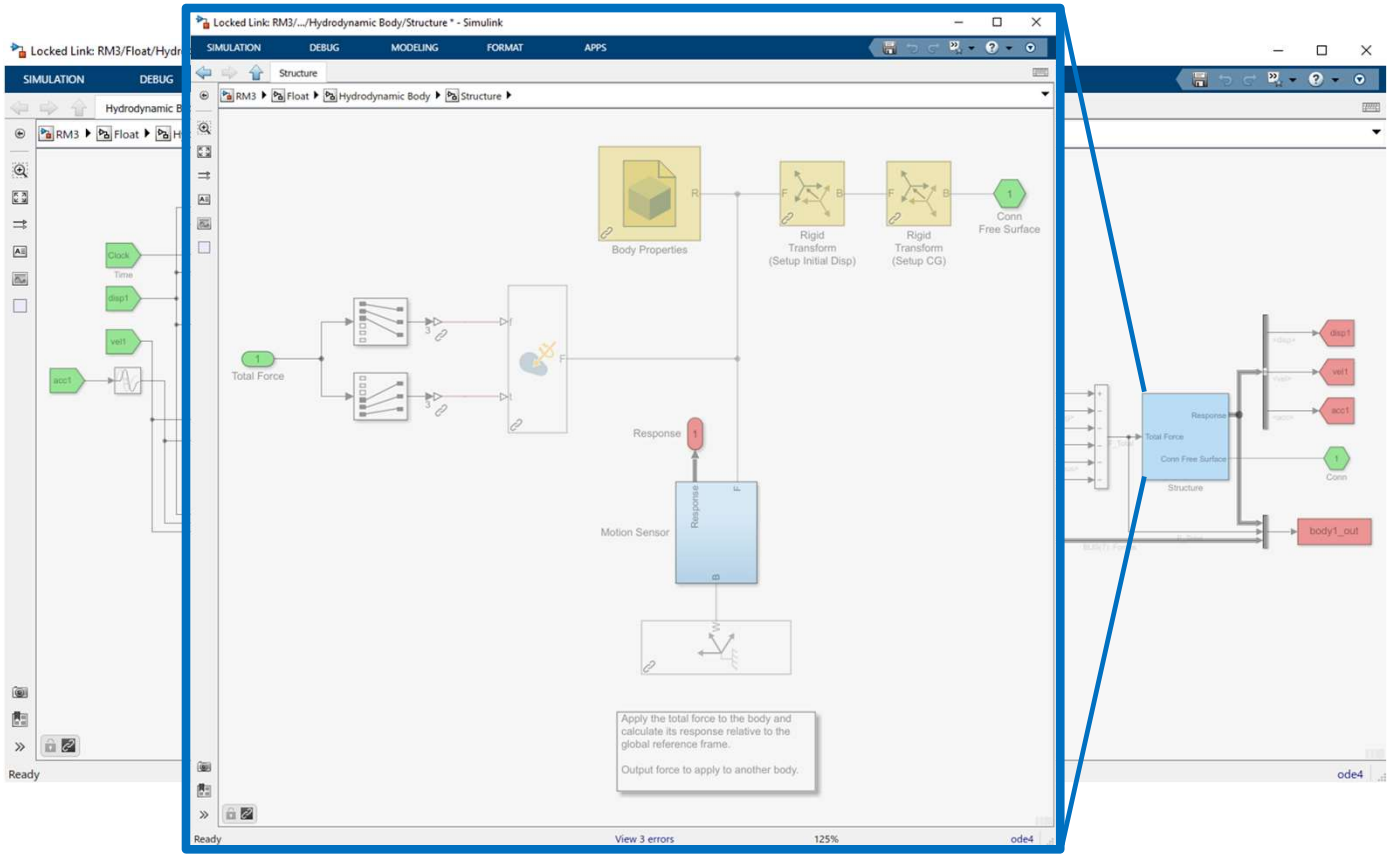
Body Class: Morrison and Viscous Damping Force



$$F_v = -C_v\dot{X} - \frac{C_d\rho A_d}{2}\dot{X}|\dot{X}| = -C_v\dot{X} - C_D\dot{X}|\dot{X}|$$

$$F_{me} = \rho\forall\dot{v} + \rho\forall C_a(\dot{v} - \ddot{X}) + \frac{C_d\rho A_d}{2}(v - \dot{X})|v - \dot{X}|$$

Simscape Multibody

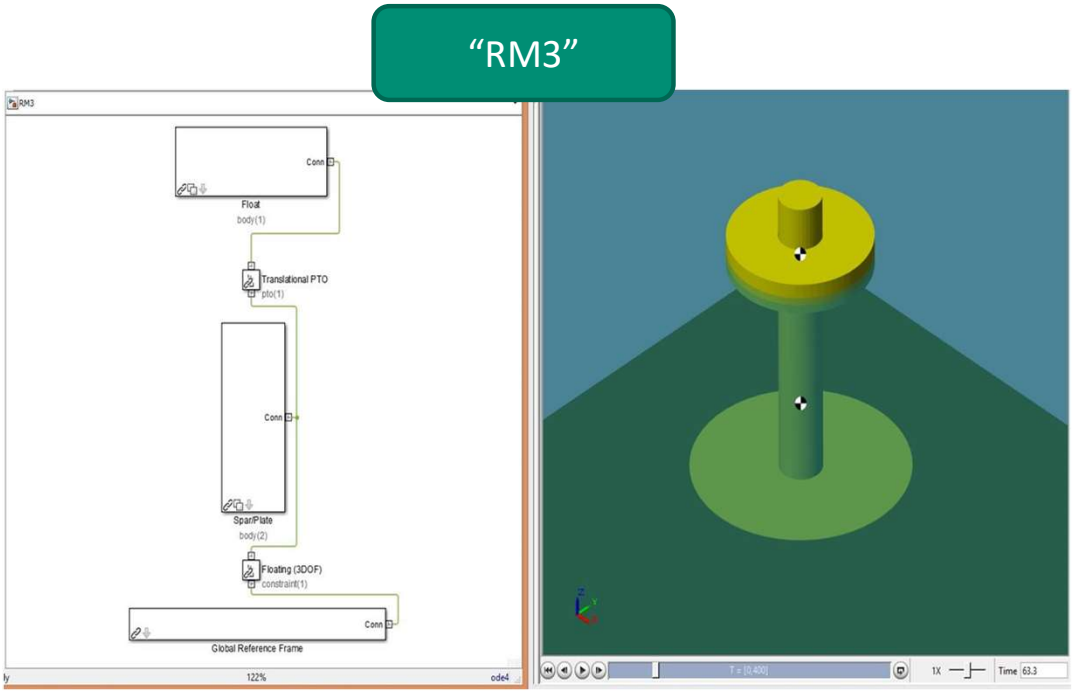


Tutorial

Get started with WEC-Sim

This presentation will give a demonstration of how to run the reference model 3 (RM3) WEC in WEC-Sim

1. **What is WEC-Sim? What is RM3?**
2. **Check WEC-Sim is installed**
3. **Run BEMIO**
4. **Build Simulink Model**
5. **Write wecSimInputFile.m**
6. **Run WEC-Sim**
7. **Visualize Outputs**



What is WEC-Sim?

WEC-Sim (Wave Energy Converter Simulator)

- Simulates WECs in operational sea states, based on linear potential flow theory
- Takes frequency-domain hydrodynamic coefficients (e.g. from BEM codes such as WAMIT or Capytaine) as an input
- Uses Cummins' equation to convert these frequency-domain coefficients into time-domain forces – enabling us to model nonlinear constraints, subsystems, moorings, etc.
- Developed in MATLAB/Simulink, using Simscape Multibody
  - Available at https://github.com/WEC-Sim/WEC-Sim
- First Release: v1.0 in June 2014
- Current Release: v5.0.1 in Sept 2022

U.S. DEPARTMENT OF ENERGY

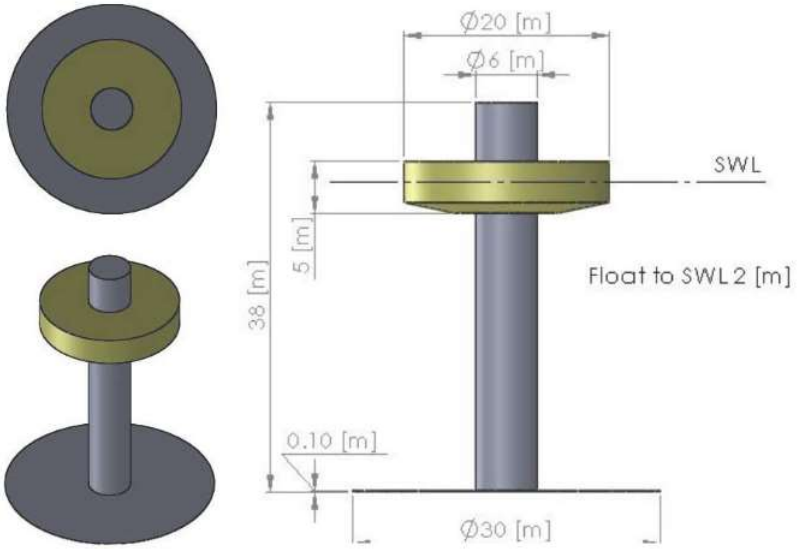
Sandia National Laboratories

NREL
NATIONAL RENEWABLE ENERGY LABORATORY

Reference Model 3 (RM3)

Device Geometry

The Reference Model 3 (RM3) is a two-body point absorber consisting of a float and a reaction plate.



WEC-Sim Model Files

Listed below are the files required to run the RM3 simulation in WEC-Sim. Optionally, users may supply a userDefinedFunctions.m file for post-processing

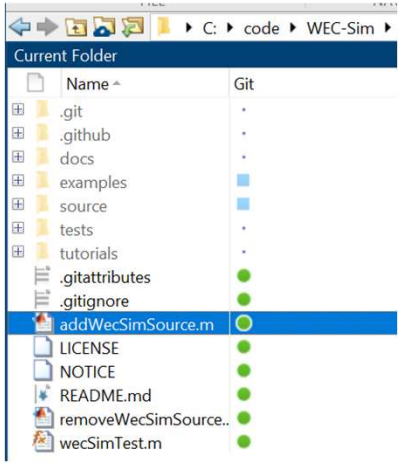
File Type	File Name	Directory
Input File	`wecSimInputFile.m`	`$WECSIM/tutorials/rm3/`
Simulink Model	`rm3.slx`	`$WECSIM/tutorials/rm3/`
Hydrodynamic Data	`rm3.h5`	`$WECSIM/tutorials/rm3/hydroData/`
Geometry Files	`float.stl` & `plate.stl`	`$WECSIM/tutorials/rm3/geometry/`

Recommendation: copy the tutorials/rm3 folder from the WEC-Sim repo into a new location

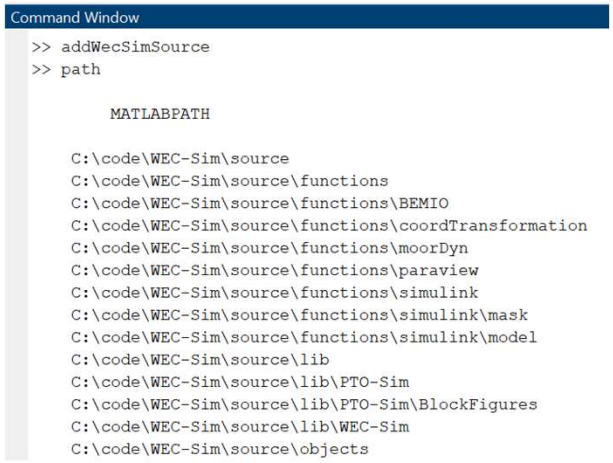
https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

Check that WEC-Sim is installed

1. Navigate to your WEC-Sim folder and run the script 'addWecSimSource.m':



2. Then check that WEC-Sim has been added to your MATLAB path:



RM3 Tutorial: Step 1. Run BEMIO

Run BEMIO

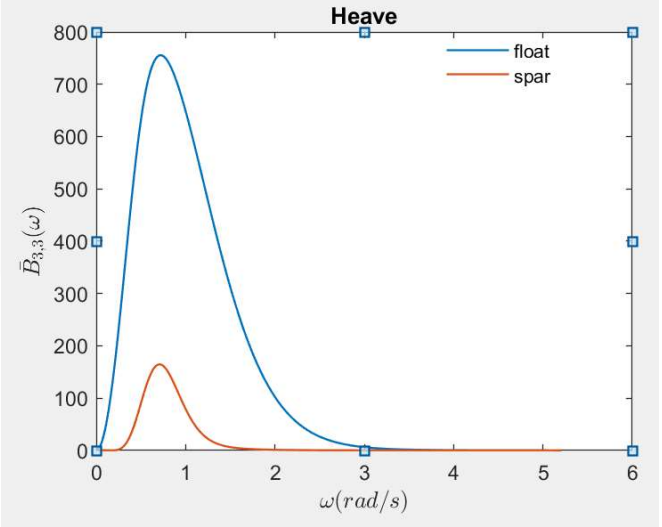
Hydrodynamic data for each RM3 body must be parsed into a HDF5 file using BEMIO. BEMIO converts hydrodynamic data from WAMIT, NEMOH or AQWA into a HDF5 file, `*.h5` that is then read by WEC-Sim. The RM3 tutorial includes data from a WAMIT run, `rm3.out`, of the RM3 geometry in the `$WECSIM/tutorials/rm3/hydroData/` directory. The RM3 WAMIT `rm3.out` file and the BEMIO `bemio.m` script are then used to generate the `rm3.h5` file.

This is done by navigating to the `$WECSIM/tutorials/rm3/hydroData/` directory, and typing``bemio`` in the MATLAB Command Window:

```
>> bemio
```

bemio.m

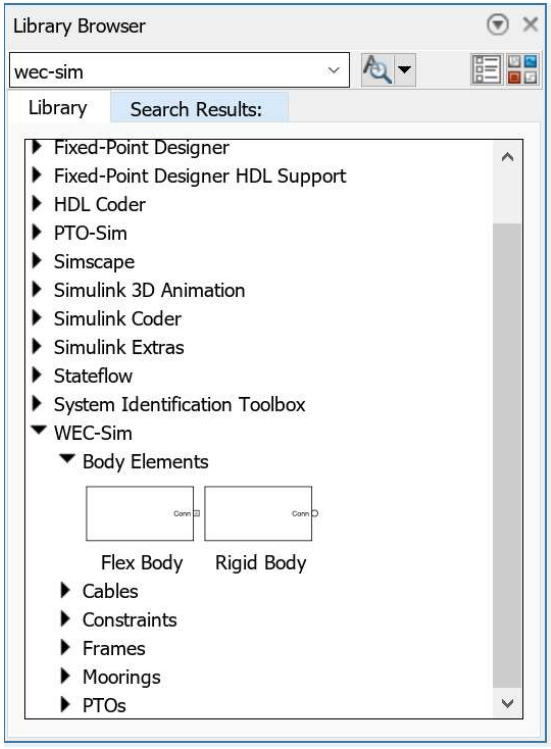
```
hydro = struct();

hydro = readWAMIT(hydro,'rm3.out',[]);
hydro = radiationIRF(hydro,60,[],[],[],[]);
hydro = radiationIRFSS(hydro,[],[]);
hydro = excitationIRF(hydro,157,[],[],[],[]);
writeBEMIOH5(hydro)
plotBEMIO(hydro)
```

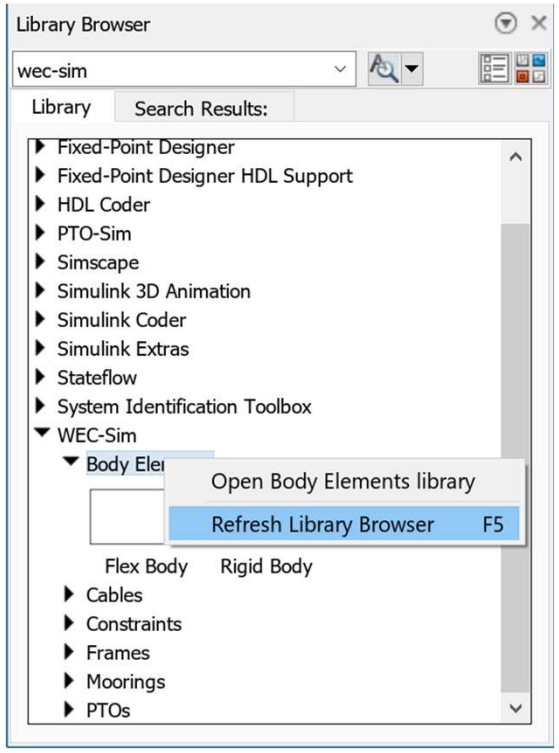


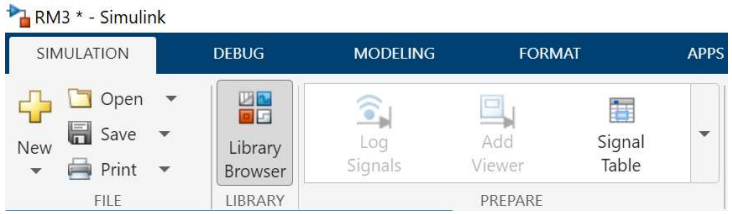
https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

RM3 Tutorial: Step 2. Build Simulink Model

i. Open RM3.slx (it should be blank)
ii. Open the Library Browser







The Library Brower may need to be refreshed to show blocks within the WEC-Sim library.

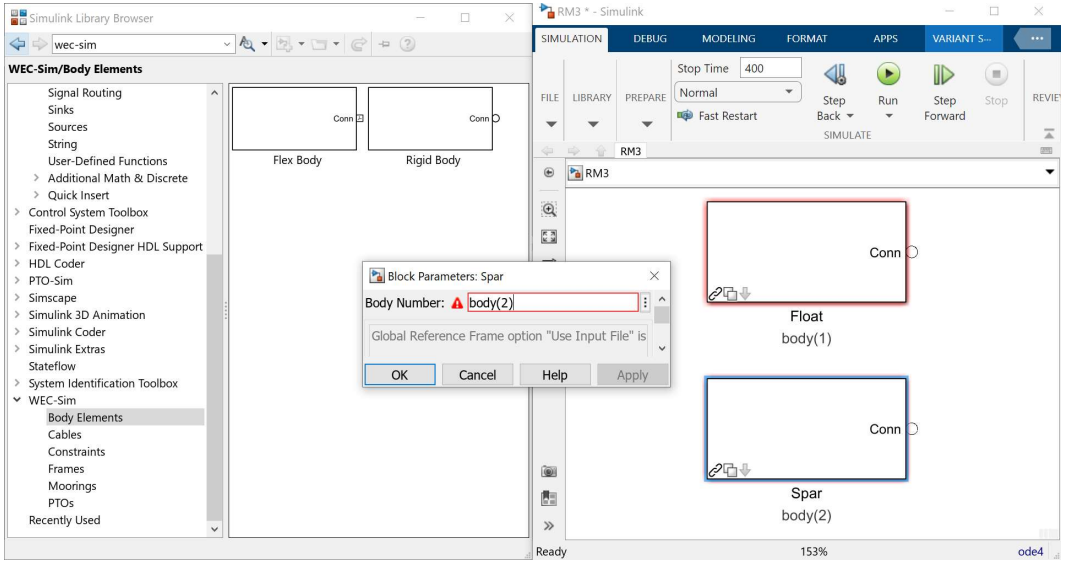
https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

RM3 Tutorial: Step 2. Build Simulink Model

ii. Add Rigid Body Blocks

- Place two **Rigid Body** blocks from *Body Elements* in *WEC-Sim Library* in the Simulink model file, one for each RM3 rigid body.
- Double click on the **Rigid Body** block, and rename each instance of the body. The first body must be called `body(1)`, and the second body should be called `body(2)`.

"Variable 'body' does not exist" warning is okay (for now)

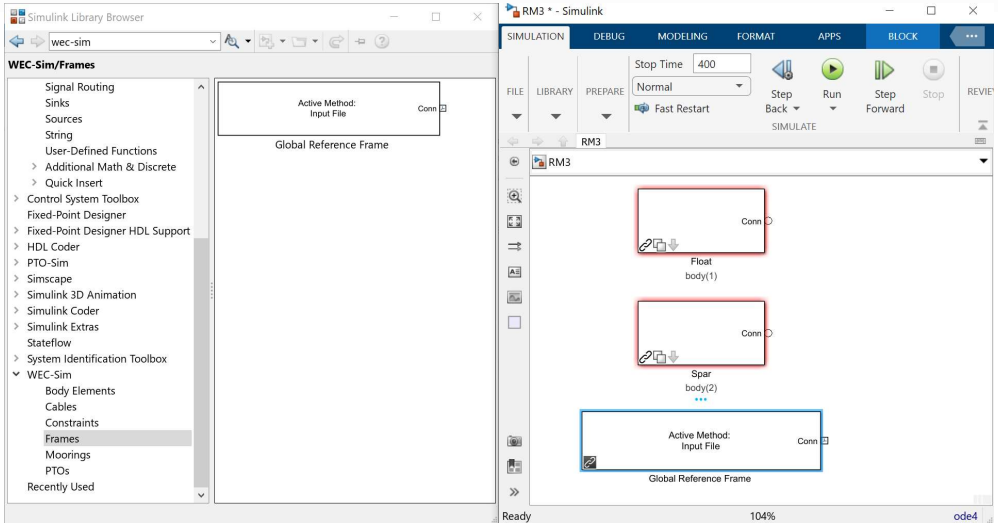


https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

RM3 Tutorial: Step 2. Build Simulink Model

iii. Add Global Reference Frame

- Place the **Global Reference Frame** from *Frames* in the *WEC-Sim Library* in the Simulink model file. The global reference frame acts as the seabed.



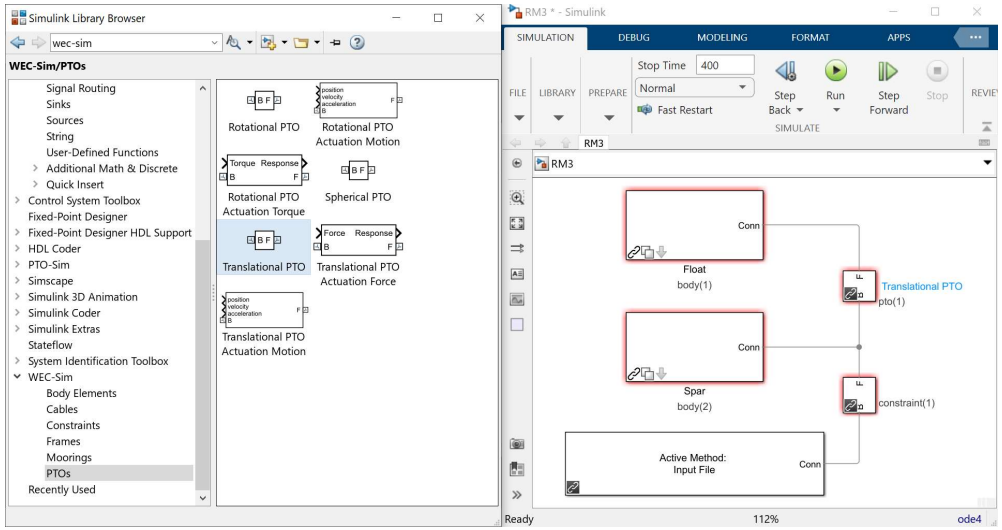
"Variable 'body' does not exist" warning is okay (for now)

https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

RM3 Tutorial: Step 2. Build Simulink Model

iv. Add 3DOF Constraint and Translational PTO

- Place the **Floating (3DOF)** block from *Constrains* to connect the plate to the seabed. This constrains the plate to move in 3DOF relative to the **Global Reference Frame**.
- Place the **Translational PTO** block from *PTOs* to connect the float to the spar. This constrains the float to move in heave relative to the spar, and allows definition of PTO damping.



- **B stands for Base**
- **F stands for Follower**
- **It is very important to use the correct base and follower**
- **Base port towards the Global Reference Frame; Follower port away**

"Variable 'pto' does not exist" warning is okay (for now)

https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

RM3 Tutorial: Step 3. Write *wecSimInputFile.m*

- Input file must be named: ***wecSimInputFile.m***
- Input file must be located in the case directory:
  $WEC-Sim/tutorials/RM3
- Input file template provided in:
  $WEC-Sim/tutorials/RM3
- Complete input file located in:
  $WEC-Sim/examples/RM3

- **Update the following:**

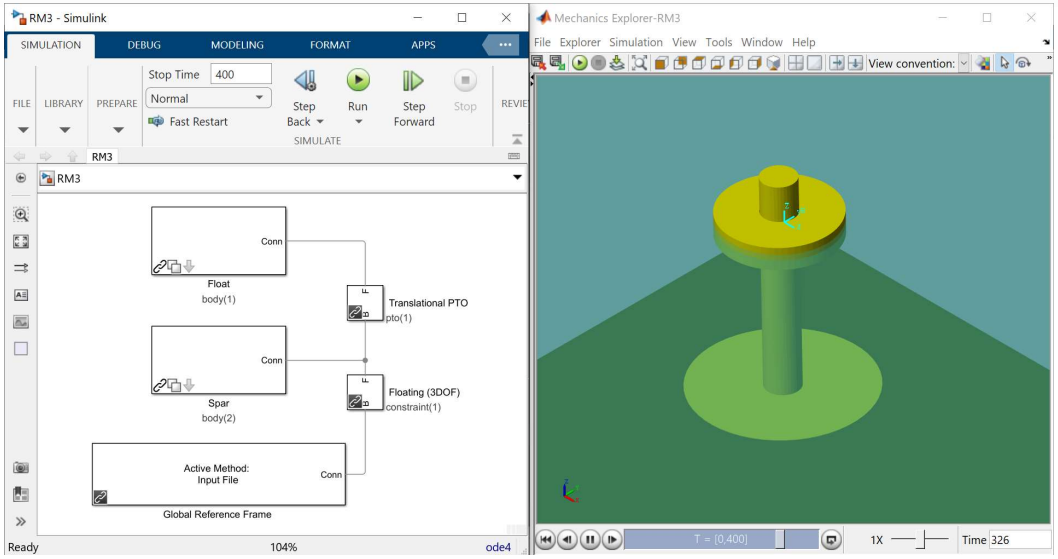
```
simu.simMechanicsFile = 'RM3.slx';
waves = waveClass('regular');
waves.period = 8;
waves.height = 2.5;
body(1) = bodyClass('hydroData/rm3.h5');
body(2) = bodyClass('hydroData/rm3.h5');
pto(1).stiffness = 0;
pto(1).damping = 1200000;
```

https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

RM3 Tutorial: Step 4. Run WEC-Sim

Run wecSim.m

To execute the WEC-Sim code for the RM3 tutorial, type `wecSim` into the MATLAB Command Window. Below is a figure showing the final RM3 Simulink model and the WEC-Sim GUI during the simulation. For more information on using WEC-Sim to model the RM3 device, refer to [A1].



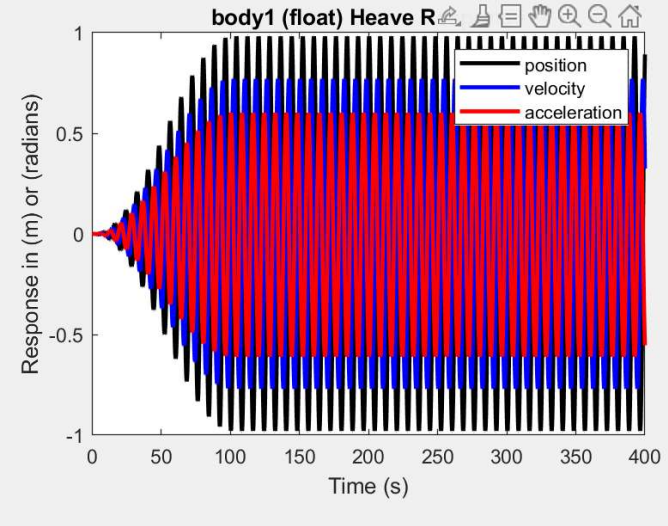
- Must execute WEC-Sim from the case directory: ***$WEC-Sim/tutorials/rm3***
- Type into Command Window: ***wecSim.m***
- Mechanics Explorer should open up with RM3 simulation

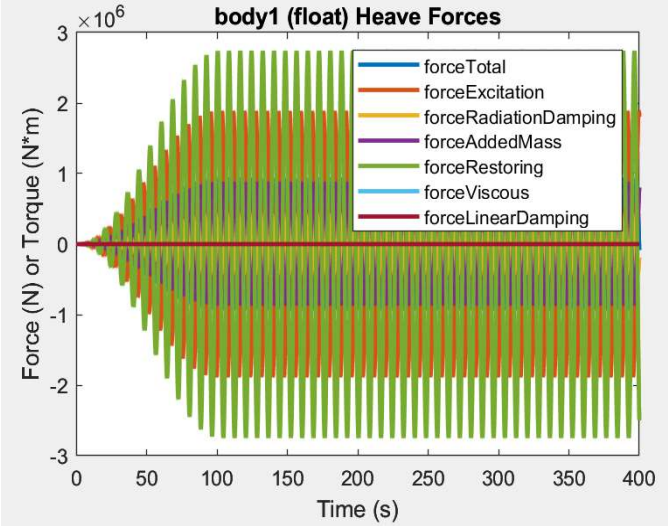
https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

RM3 Tutorial: Step 5. Post-Processing

Post-processing

The RM3 tutorial includes a `userDefinedFunctions.m` which plots RM3 forces and responses. This file can be modified by users for post-processing. Additionally, once the WEC-Sim run is complete, the WEC-Sim results are saved to the `output` variable in the MATLAB workspace.





https://wec-sim.github.io/WEC-Sim/master/user/tutorials.html#two-body-point-absorber-rm3

Thank you

For more information please visit the WEC-Sim website:

http://wec-sim.github.io/WEC-Sim

If you have questions on this presentation please reach out to any of the WEC-Sim Developers on GitHub:

https://github.com/WEC-Sim/WEC-Sim

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308.

Funding provided by the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Water Power Technologies Office. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.