Wamit2vessel2json

Requirements

→MATLAB

File	Purpose
`*.1`	Added mass and damping matrices
`*.2` **or** `*.3`	Force RAOs — Haskind (2) or Diffraction (3); one of them is used
`*.4`	Motion RAOs (Response Amplitude Operators for vessel motion)
`*.8` **or** `*.9`	Wave drift forces — Momentum (8) or Pressure (9); one of them is used
`*.frc`	Rigid-body mass properties
`*.out`	Additional properties (volume, CG, CB, gravity, etc.)
`*.pot`	Used to check the number of headings in simulation

Optional / Supporting Files:

- `.gdf` : Geometry Definition File (used by WAMIT, **not read by** `wamit2vessel`, but must exist for WAMIT runs) can view geometry with Thor Fossens

```
plot_wamitgdf('name_of_gdf_file', [0.3 0.5 0.9], 1, 1, 0);
```

- `.wam` : Input file for WAMIT (defines parameters/settings — not directly read here)

Steps

1. If anything like this:

```
WAMIT Numeric Output -- Filename  voyager.1              24-Feb-2025  22:33
```

in the first line of any of these files, *.1, *.2, *.3, *.4, *.8, *.9, remove that and make sure that numbers start at line one.

2. Run this, making sure that files are in path:

```
>> T_draught = 0.104;   % m
Lpp = 0.99;          % m
Boa = 0.308;         % m
vessel_name = wamit2vessel('name_of_files', 0.104, 0.99, 0.308, '1111');
```

3. Answer the questions that come up to the best of your knowledge. The time constant shall be easy to find in *.out or *.1 as all the data is there (written in tonn), furthermore, the damping coefficient, choose the value closest to zero, and either from the wamit plot or from the calculation done from wamit site, calculate the number in kg/s. If the number is not the biggest in sway and the smallest in yaw, then something is wrong and make a calculated guess. The plot is an assessment of any wrongdoing. Look for abnormal points.
4. Run

```
>>clear
load('vessel_name.mat')
vessel_name_json = jsonencode(vessel);  % Create JSON string

fid = fopen('vessel_name.json', 'w');  % Open file for writing
if fid == -1
    error('Could not create file. Check permissions or path.');
end

fwrite(fid, vessel_name_json, 'char');  % Write JSON as char
fclose(fid);  % Always close the file
```

5. DONE: move the file to your desired location.

6. Are you going to be using it for mcsimpy? run the change_57to55.py

```
#!/usr/bin/env python3
"""
change_57to55.py  —  v2

• Trim 57-point hydrodynamic JSON files down to 55 points (drop ω=0 and ω=
• Ensure `driftfrc["amp"]` is **6-DOF** by zero-padding if the file only
  contains 3 rows (surge, sway, yaw).


Author: Kristian Magnus Røen
Date:   2025-05-05
"""

import sys, json, shutil, datetime as _dt
from pathlib import Path
import numpy as np


def trim57_to_55(arr):
    """Return np.array with every 57-element axis sliced 1:-1"""
    arr = np.asarray(arr, dtype=np.float32)
    slicer = [slice(None) if dim != 57 else slice(1, -1)
              for dim in arr.shape]
    return arr[tuple(slicer)]


def recurse(obj):
    if isinstance(obj, list):
        try:
            arr = np.asarray(obj, dtype=np.float32)
            if 57 in arr.shape:
                return trim57_to_55(arr).tolist()
        except Exception:
            pass
        return [recurse(x) for x in obj]
```

```
    if isinstance(obj, dict):
        return {k: recurse(v) for k, v in obj.items()}
    return obj


def pad_drift_amp(drift_amp):
    """Ensure drift_amp shape is (6, 55, ...). Pad zeros on DOF-axis if needed.""
    arr = np.asarray(drift_amp, dtype=np.float32)
    if arr.shape[0] == 6:
        return arr
    if arr.shape[0] == 3:       # surge, sway, yaw
        zeros = np.zeros((3, *arr.shape[1:]), dtype=arr.dtype)
        arr6 = np.concatenate([arr, zeros], axis=0)
        # copy row 2 (heave) into row 5 (yaw) to match WaveLoad expectations
        arr6[5] = arr6[2]
        arr6[2] = 0.0
        return arr6
    raise RuntimeError(f"Unexpected drift_amp DOF axis length {arr.shape[0]} (


def main():
    if len(sys.argv) < 2:
        print("Usage: python change_57to55.py input.json [output.json]")
        sys.exit(1)
    in_path = Path(sys.argv[1]).expanduser()
    out_path = Path(sys.argv[2]).expanduser() if len(sys.argv) > 2 else in_path

    with in_path.open() as f:
        data = json.load(f)

    if "freqs" in data and len(data["freqs"]) == 57:
        data["freqs"] = data["freqs"][1:-1]

    for key in ("A", "B", "Bv"):
        if key in data:
            data[key] = recurse(data[key])

    # driftfrc / forceRAO blocks
```

```
    for block in ("forceRAO",):
        if block in data:
            for sub in ("amp", "phase"):
                if sub in data[block]:
                    data[block][sub] = recurse(data[block][sub])

    if "driftfrc" in data and "amp" in data["driftfrc"]:
        trimmed = recurse(data["driftfrc"]["amp"])
        padded = pad_drift_amp(trimmed)
        data["driftfrc"]["amp"] = padded.tolist()
        if "phase" in data["driftfrc"]:
            data["driftfrc"]["phase"] = recurse(data["driftfrc"]["phase"])

    # backup if overwriting
    if out_path == in_path:
        ts = _dt.datetime.now().strftime("%Y%m%d_%H%M%S")
        backup = in_path.with_suffix(in_path.suffix + f".bak_{ts}")
        shutil.copy2(in_path, backup)
        print("Backup saved to", backup)

    with out_path.open("w") as f:
        json.dump(data, f, indent=2)
    print("Trimmed & padded JSON written to", out_path)


if __name__ == "__main__":
    main()
```

wamit2vessel code for version 7

```
function vessel = wamit2vessel(filename,T_draught,Lpp,Boa,plot_flag)
% wamit2vessel reads data from WAMIT output files and store the data in
%   vesselname.mat using the MSS vessel struture.
%
%   vessel = wamit2vessel(filename)
%
```

```
% The Wamit GDF-file must be defined in GLOBAL COORDINATES, i.e. origin
% [Lpp/2 B/2 WL], see the Wamit manual. The axes are transformed from WA
% axes to Fossen axes.
%
% Examples:
%   ../MSS/HYDRO/vessels_wamit/tanker
%       vessel = wamit2vessel('tanker')
%       vessel = wamit2vessel('tanker',10,246,46)
%       vessel = wamit2vessel('tanker',10,246,46,'1111')
%
% ../MSS/HYDRO/vessels_wamit/semisub
%       vessel = wamit2vessel('semisub',21,115,80,'1111')
%
% ../MSS/HYDRO/vessels_wamit/fpso
%       vessel = wamit2vessel('fpso',12,200,44,'1111')
%
% Inputs:
%    filename (without extension) reads and processes the following
%                WAMIT files:
%                  *.1    added mass and damping
%                  *.2    force RAOs from Haskind (vessel.forceRAO)
%                  *.3    force RAOs from diffraction (vessel.forceRAO)
%                  *.4    motion RAOs (vessel.motionRAO)
%                  *.8    wave drift data from momentum (vessel.driftfrc)
%                  *.9    wave drift data from pressure (vessel.driftfrc)
%                  *.frc  rigid body mass parameters
%                  *.out  output file
%
%   T_draught (optionally)   Draught corresponding to GDF file
%   Lpp (optionally)         Length between AP and FP corresponding to GDF file
%   Boa (optionally)         Breadth over all/Beam corresponding to GDF file
%   plot_flag (optionally):  '1000' plot A and B matrices
%                          '0100' plot force RAOs
%                            '0010' plot motion RAOs
%                            '0001' plot wave drift forces
%                            '0000' NO PLOT
%                            '1111' PLOT ALL
%
```

```
% Outputs:
% vessel contains data in the following form:
%
%   vessel.headings:                headings
%   vessel.velocities:              velocities
%   vessel.freqs:                   frequencies (A and B matrices)
%
%   vessel.A(6,6,freqno,velno):     added mass matrix
%   vessel.B(6,6,freqno,velno):     damping matrix
%   vessel.C(6,6,freqno,velno):     spring stiffness matrix
%   vessel.MRB(6,6):                rigid-body mass matrix
%
%   vessel.driftfrc.
%     amp(freqno,headno,velno):     wave drift force amplitudes
%     w(1,freqno):                  circular wave frequencies
%
%   vessel.forceRAO.
%     amp{1:6}(freqno,headno,velno):    wave excitation force amplitudes
%     phase{1:6}(freqno,headno,velno):  wave excitation force phases
%     w(1,freqno):                  circular wave frequencies
%
%   vessel.motionRAO.
%     amp{1:6}(freqno-n,headno,velno):   wave motion RAO amplitudes
%     phase{1:6}(freqno-n,headno,velno): wave motion RAO phases
%     w(1,freqno-n):                circular wave frequencies where n>=0
%                                   is chosen such that w_o > w_min
%                                   (typically w_min = 0.2 rad/s)
%   vessel.main.
%     name:  vessel name
%     m:     mass
%     rho    density of water
%     nabla: displacement volume
%     k44:   radius of inertia
%     k55:   radius of inertia
%     k66:   radius of inertia
%     m:     mass
%     CG:    centre of gravity  [LCG 0 VCG] w.r.t. Lpp/2 and Keel Line
%     CB:    centre of buoyancy [LCB 0 VCB] w.r.t. Lpp/2 and Keel Line
```

```
%       Lpp:   length between the perpendiculars
%       Lwl:   length of water line
%       T:     draught (water line)
%       B:     breadth of ship
%       C_B:  block coefficient
%       S:    area of wetted surface
%       GM_L: longitudinal metacentric height
%       GM_T: transverse metacentric height
%
% ------------------------------------------------------------------------
% Author:    Thor I. Fossen, edit: Kristian Magnus Roen
% Date:      2005-09-17 first version
% Revisions: 2008-02-15 Minor bug fixes
%            2009-09-11 Using new viscous damping viscous.m
%            2021-03-07 Minor bug fixes
%            2022-12-20 Fixed incorrect mirroring of RAOs to 180-360 deg
%            2023-04-17 Added a correction for LCF when computing GM_L
%            2025-05-05 Version 7 is now acceptable

%%
if ~exist('plot_flag')
   plot_flag = '1000';
end

disp(' ');
disp('************************* MSS Hydro ********************************
disp('vessel = wamit2vessel(*) computes the MSS vessel structure from')
disp('WAMIT (version 6.X) output files *.N (N=1,2,3,4,8,9), *.out, and *.frc')
disp('generated using wamit_inputs.m. The results are stored as *.mat.')
disp(' ');
disp('Author: Thor I. Fossen');
disp('************************************************************************')
disp(' ')

% different number of input arguments
if nargin == 1

   fileinput  = input(['Vessel name (default = ' filename ')     : '],'s');
```

```
    if strcmp(fileinput,'')
        vesselfile = filename;
    else
        vesselfile = fileinput;       % default value
    end

    T_draught = input('Draught T (m) corresponding to the GDF-file: ');
    Lpp       = input('Length between perpendiculars L_{pp}(m)    : ');
    Boa       = input('Breadth (m)                               : ');

elseif nargin == 4 || nargin == 5  % Lpp, B and T_draught are specified in call
     vesselfile = filename;
else
    error('wrong number of input arguments');
end

if T_draught<=0
  error('Error: T_draft>0 must be specified');
  return
end

if Lpp<=0
  error('Error: Lpp>0 must be specified');
  return
end

if Boa<=0
  error('Error: B>0 must be specified');
  return
end

vessel.main.name  = vesselfile;
vessel.main.T     = T_draught;
vessel.main.B     = Boa;
vessel.main.Lpp   = Lpp;
vessel.velocities = 0;
vessel.headings   = (pi/180)*(0:10:180);
Nheadings         = length(vessel.headings);
```

```
% Axes transformations for data processing:
% Wamit uses negative incoming wave angle beta compared to mss: T_data
% Wamit GDF x,y,z axes are transformed to mss axes by: T_gdf
T_gdf = [1 -1 -1];            % Wamit2Fossen axes (sign correction)
T_data = [1 -1 1 -1 1 -1];     % RAO data sign correction due to neg. wave angles

% Scaling of raw data:
% Matrix: Tscale * A * Tscale
% RAO: change amplitude with signs in T_rao (alternavively change phase wit
Tscale = diag([T_gdf T_gdf]);    % 6-DOF transformation matrix for A and B da
T_rao = [T_gdf T_gdf] .* T_data; % total force/motion RAO transformation

%--------------------------------------------------------------------------
%% Check number of WAMIT headings in *.pot file
%--------------------------------------------------------------------------
goflag = 0;

if exist([filename '.pot'])
   fid_pot = fopen(strcat(filename,'.pot'));
   while feof(fid_pot) == 0
      txt = char(fgetl(fid_pot));
      if contains(txt, '19') || contains(txt, '-19')
         goflag = 1;
      end
   end
   fclose(fid_pot);
else
   disp(['Error: the WAMIT file ' strcat(filename,'.pot') ' is missing']);
   return
end

if goflag == 0
   disp('Error: run WAMIT with 19 headings 0:10:180 deg');
   return
end

%--------------------------------------------------------------------------
```

```
%% Read rigid-body mass parameters from *.frc file
%-------------------------------------------------------------------------
if exist([filename '.frc'], 'file')
   fid_frc = fopen([filename '.frc'], 'r');
   first_line = fgetl(fid_frc);
   frewind(fid_frc);  % Reset pointer

   % Decide version 6 or 7
   if contains(first_line, 'FRC file')
      % ----- Version 7 -----
      FRC_VERSION = 7;
   else
      % ----- Version 6 -----
      FRC_VERSION = 6;
   end

   if FRC_VERSION == 6
      % ================= WAMIT v6 =================
      frcfile = [filename '.frc'];
      % Skip the first line and read the numeric data into 'frc'
      frc = dlmread(frcfile, '', 1, 0);

      % Check alt 1 or alt 2
      if frc(5,1) < 1000
         % ---------- Alt 1 ----------
         FRC_ALT = 1;

         % Example lines in alt 1:
         % (2,1) ⇒ VCG, (3,1),(4,2),(5,3) ⇒ k44,k55,k66, etc.
         % Usually no explicit density ⇒ set default 1025 unless you parse it
         VCG = frc(2,1);
         xg = 0; yg = 0; zg = VCG;

         vessel.main.k44 = frc(3,1);
         vessel.main.k55 = frc(4,2);
         vessel.main.k66 = frc(5,3);
         vessel.main.k46 = frc(3,3);  % if present
         vessel.main.rho = 1025;      % default salt water
```

```
    else
        % ---------- Alt 2 ----------
        FRC_ALT = 2;

        % parse the xg, yg, zg from line 3
        xg = frc(3,1);
        yg = frc(3,2);
        zg = frc(3,3);

        % lines 5..10 ⇒ 6×6 mass matrix
        MRB_wamit = frc(5:10, 1:6);

        % Convert from WAMIT to Fossen axes
        MRB = Tscale * MRB_wamit * Tscale;
        vessel.MRB = MRB;

        vessel.main.m   = frc(5,1);   % mass
        vessel.main.rho = frc(2,1);   % density

        % Gyration radii
        vessel.main.k44 = sqrt(MRB(4,4) / vessel.main.m);
        vessel.main.k55 = sqrt(MRB(5,5) / vessel.main.m);
        vessel.main.k66 = sqrt(MRB(6,6) / vessel.main.m);
    end

    fclose(fid_frc);

elseif FRC_VERSION == 7
    % ================= WAMIT v7 =================
    C = textscan(fid_frc, '%s', 'Delimiter', '\n', 'Whitespace', '');
    lines = C{1};
    fclose(fid_frc);

    % lines{1}: "FRC file"
    % lines{2}: "1 0 1 1 0 0 0 1 0"
    % lines{3}: "1000       RHO"
    % lines{4}: "-0.0597 0. 0.016    VCG"
```

```
% lines{5}: "1          IMASS"
% lines{6..11}: 6 lines for MRB
% line 12: "0" → IDAMP, etc.

% Helper function to extract the first float from a line with trailing text
cleanNumber = @(strLine) ...
   str2double( regexp( regexprep(strLine, '^\xEF\xBB\xBF',''), ...
                  '[+-]?(\d+(\.\d*)?|\.\d+)([eE][+-]?\d+)?', 'match','once') );

% 1) RHO ⇒ line 3
raw_rho = lines{3};
vessel.main.rho = cleanNumber(raw_rho);
disp(['DEBUG: Cleaned RHO line ⇒ "' raw_rho '" , numeric = ' num2str(ve

% 2) VCG ⇒ line 4
raw_vcg = lines{4};
% parse 3 floats from line 4
vcgVals = sscanf(raw_vcg, '%f');
if length(vcgVals) >= 1
   zg = vcgVals(1);
else
   zg = 0;
end
xg = 0; yg = 0;  % if not used

% 3) IMASS ⇒ line 5
raw_imass = lines{5};
imass_val = cleanNumber(raw_imass);
disp(['DEBUG: Cleaned IMASS line ⇒ "' raw_imass '" , numeric = ' num2s

% 4) MRB ⇒ lines 6..11
MRB_data = zeros(6,6);
for row_idx = 1:6
   row_str = strtrim(lines{5 + row_idx});  % lines{6..11}
   row_vals = sscanf(row_str, '%f')';
   if length(row_vals) < 6
      error(['Error reading MRB row ', num2str(row_idx), ...
         ': need 6 floats, got ', num2str(length(row_vals))]);
```

```
            end
            MRB_data(row_idx,:) = row_vals(1:6);
        end

        % Convert from WAMIT to Fossen axes
        MRB = Tscale * MRB_data * Tscale;
        vessel.MRB = MRB;

        % mass is MRB(1,1)
        vessel.main.m = MRB(1,1);
        disp(['DEBUG: Computed MRB mass (v7) = ', num2str(vessel.main.m)]);

        % Gyration
        vessel.main.k44 = sqrt(MRB(4,4) / vessel.main.m);
        vessel.main.k55 = sqrt(MRB(5,5) / vessel.main.m);
        vessel.main.k66 = sqrt(MRB(6,6) / vessel.main.m);

        % so .out file sees we have a full MRB
        FRC_ALT = 2;
    end

else
    error('Error: No WAMIT *.frc file found.')
end
%--------------------------------------------------------------------
%   Reading *.out for version-7 WAMIT, matching the old style code
%--------------------------------------------------------------------
if exist([filename '.out'])
    fid1 = fopen([filename '.out']);

    % Detect WAMIT version from header
    is_v7 = false;
    while ~feof(fid1)
        line = fgetl(fid1);
        if contains(line, 'WAMIT  Version 7')
            is_v7 = true;
            break;
        end
```

```
end
frewind(fid1);

% We want to track how many lines pass to define Nfreqs
count = 0;
countstart = 0;

while ~feof(fid1)
   count = count + 1;
   txt = fgetl(fid1);

   %--------------------------------
   % 1) POTEN run date ⇒ define countstart so we can do: Nfreqs = count -
   %--------------------------------
   if contains(txt,'POTEN run date and starting time:')
      countstart = count;
   end

   %--------------------------------
   % 2) Parse Gravity as a single numeric
   %--------------------------------
   if contains(txt,'Gravity:')
      tokensG = regexp(txt,'Gravity:\s*([\d\.Ee+\-]+)','tokens');
      if ~isempty(tokensG)
         vessel.main.g = str2double(tokensG{1}{1});  % e.g. 9.81
         disp(['DEBUG: gravity = ' num2str(vessel.main.g)]);
      end
      % Attempt to define Nfreqs right after we found gravity:
      % Usually the line after "POTEN run date" minus some offset,
      % but let's keep the "Nfreqs = ..." the same as v6:
      Nfreqs = count - countstart - 2;
   end

   % 3) Parse "Length scale:"  (grab the LAST float on the line)
   if contains(txt,'Length scale:')
      nums  = regexp(txt,'[+\-]?\d+(\.\d*)?([eE][+\-]?\d+)?','match');
      ULEN  = str2double(nums{end});         % e.g. 0.99  (model length)
      disp(['DEBUG: ULEN = ' num2str(ULEN)]);
```

```
end


%-------------------------------
% 4) Parse Volumes
%-------------------------------
if contains(txt, 'Volumes (VOLX,VOLY,VOLZ):')
    repline = strrep(txt, 'Volumes (VOLX,VOLY,VOLZ):','');
    repline = strrep(repline, char(160), ' ');
    repline = strrep(repline, char(65279), ' ');
    repline = strtrim(repline);
    nums = regexp(repline, '[0-9eE+.\-]+', 'match');
    disp(['DEBUG: raw numeric tokens → ', strjoin(nums, ', ')])
    nums = str2double(nums);
    if length(nums) < 3 || any(isnan(nums))
        error('Error parsing volume data from .out file (found <3 or NaNs).');
    end
    volumes = nums(end-2:end);
    disp(['Parsed VOLX, VOLY, VOLZ: ', num2str(volumes)]);
    vessel.main.nabla = mean(volumes);
    mass = vessel.main.rho * vessel.main.nabla;

    if FRC_ALT == 2
        disp(' ');
        disp('Processing WAMIT data files....');
        disp(' ');
        disp('---------------- Mass property check (*.frc file)-----------------
        disp(['Mass computed from displaced fluid is: ' num2str(round(mass
        disp(['Mass input to Wamit *.frc file is    : ' num2str(round(vessel.mai
        if abs(mass - vessel.main.m) > 500e3
            disp('It is recommended to rerun WAMIT with correct mass param
        end
        disp('>>Return to continue, <ctrl C> to Abort');
        pause
        disp('-------------------------------------------------------------
    else
        % FRC_ALT == 1
        vessel.main.m = mass;
```

```
        MRB = zeros(6,6);
        MRB(1,1) = mass;
        MRB(2,2) = mass;
        MRB(3,3) = mass;
        MRB(1,5) = mass * zg;
        MRB(5,1) = mass * zg;
        MRB(2,4) = -mass * zg;
        MRB(4,2) = -mass * zg;

        MRB(4,4) = mass * vessel.main.k44^2;
        MRB(5,5) = mass * vessel.main.k55^2;
        MRB(6,6) = mass * vessel.main.k66^2;
        MRB(4,6) = mass * vessel.main.k46^2;
        MRB(6,4) = mass * vessel.main.k46^2;

        MRB = Tscale * MRB * Tscale;
        vessel.MRB = MRB;
    end
end

%-------------------------------
% 5) Parse " C(3,3),C(3,4),C(3,5):"
%-------------------------------
if contains(txt,' C(3,3),C(3,4),C(3,5):')
    C3 = str2num(txt(23:end));
    txt = fgetl(fid1);
    C4 = str2num(txt(23:end));
    txt = fgetl(fid1);
    C5 = str2num(txt(23:end));

    rho_g = vessel.main.rho * vessel.main.g;
    C3 = C3 .* rho_g .* [ULEN^2 ULEN^3 ULEN^3];
    C4 = C4 .* rho_g .* [ULEN^4 ULEN^4 ULEN^4];
    C5 = C5 .* rho_g .* [ULEN^4 ULEN^4];

    % Build the "C_wamit" matrix
    C_wamit = zeros(6,6);
    C_wamit(3:6,3:6) = [C3 0
```

```
                    C3(2) C4
                    C3(3) C4(2) C5
                    0     0      0   0];
    % transform to Fossen axes
    C_wamit = Tscale * C_wamit * Tscale;

    % store in vessel.C(:,:,i) for i=1..Nfreqs
    % we rely on Nfreqs from above
    for iC = 1:Nfreqs
       vessel.C(:,:,iC) = C_wamit;
    end

    vessel.main.GM_T = C_wamit(4,4) / (vessel.main.m * vessel.main.g);

    % approximate GM_L:
    if (vessel.main.Lpp >= 100)
       Awp = 0.8 * vessel.main.Lpp * vessel.main.B;
    elseif (vessel.main.Lpp > 50)
       Awp = 0.65 * vessel.main.Lpp * vessel.main.B;
    else
       Awp = 0.5 * vessel.main.Lpp * vessel.main.B;
    end

    LCF = -0.1 * vessel.main.Lpp;
    vessel.main.GM_L = ( C_wamit(5,5) - ...
      vessel.main.rho*vessel.main.g *Awp* (LCF^2 ) ) / ...
      (vessel.main.m * vessel.main.g);

    if vessel.main.GM_T < 0
       error(['Error: GM_T = ' num2str(vessel.main.GM_T) ' < 0']);
    end
    if vessel.main.GM_L < 0
       error(['Error: GM_L = ' num2str(vessel.main.GM_L) ' < 0']);
    end
end

%--------------------------------
% 6) Center of Buoyancy
```

```
        %-------------------------------
        if contains(txt, 'Center of Buoyancy')
            temp = str2num(txt(33:end));
            if ~isempty(temp)
                C_B = T_gdf .* temp;
                vessel.main.CB = [C_B(1) C_B(2) T_draught - C_B(3)];
            end
        end

        %-------------------------------
        % 7) Center of Gravity
        %-------------------------------
        if contains(txt, 'Center of Gravity')
            temp = str2num(txt(33:end));
            if ~isempty(temp)
                if FRC_ALT == 2
                    C_G = T_gdf .* temp;
                    vessel.main.CG = [C_G(1), C_G(2), T_draught - C_G(3)];
                else
                    vessel.main.CG = [0, 0, T_draught + VCG];
                end
            end
        end

    end

    fclose(fid1);

else
    error('Error: No WAMIT *.out file')
end



%-------------------------------------------------------------------------
%% Read added mass and damping from *.1 file
%-------------------------------------------------------------------------
if exist([filename '.1'])
```

```
 disp('Processing WAMIT added mass and damping data...')

 ABCfile = [filename '.1'];
 format1 = '%n %n %n %n %n';
 [periods,i,j,A,B] = textread(ABCfile,format1);
 Nperiods = length(periods);

 unique_periods = unique(periods);

% check to see if data contain the zero and INF frequencies
if isempty(find(unique_periods==-1))
   error('Run WAMIT with periods: -1 0 p1 p2 p3...pn where pi are postive pe
end

if isempty(find(unique_periods==0))
   error('Run WAMIT with periods: -1 0 p1 p2 p3...pn where pi are postive pe
end

 % frequencies (inf is chosen as 40 rad/s)
 freqs = [0 40 (2*pi./unique_periods(3:length(unique_periods)))'];

 % extract added mass and damping
 for p = 1:Nperiods
    idx = find(unique_periods == periods(p));
    Aij(i(p),j(p),idx) = A(p);
    Bij(i(p),j(p),idx) = B(p);
 end

 % sort with respect to frequency
 [freqs_sorted, freq_idx] = sort(freqs);
 Aij_sorted = Aij(:,:,freq_idx);
 Bij_sorted = Bij(:,:,freq_idx);

 vessel.freqs   = freqs_sorted;
 Nfreqs         = length(vessel.freqs);

 % Scaling of added mass and damping matrices (Wamit manual p. 4-3)
```

```
    % Aij = Aij' * rho * ULEN^k
    % Bij = Bij' * rho * w * ULEN^k
    % where k=3 for i,j=1,2,3, k=5 for i,j=1,2,3, k = 4 otherwise.
    scaleA  = [ ones(3)*3 ones(3)*4
                ones(3)*4 ones(3)*5 ];

    for w = 1:Nfreqs
        % scale Wamit data to SI system (Wamit axes)
        A_dim = Aij_sorted (:,:,w)*vessel.main.rho .* (ULEN .^ scaleA);
        B_dim = Bij_sorted (:,:,w)*vessel.main.rho .* vessel.freqs(w) ...
            .* (ULEN .^ scaleA);
        % transform to Fossen axes
        vessel.A(:,:,w) = Tscale * A_dim * Tscale;
        vessel.B(:,:,w) = Tscale * B_dim * Tscale;
    end

else
    error('No WAMIT *.1 file')
end

%-------------------------------------------------------------------------
%% motion RAOs in Global Coordinates
%-------------------------------------------------------------------------
if exist([filename '.4'])
    motionRAOfile = [filename '.4'];
    disp('WAMIT motion RAOs --> vessel.motionRAO')

    format_string = '%n %n %n %n %n %n %n';   % Wamit manual page 4-8
    [per,beta,DOF,mod,phase,realpart,impart] = textread(motionRAOfile,format_

    N = length(per);

    % Exstract all unique periods and wave directions
    periods = unique(per);
    ang     = unique(beta);

    for i=1:6
        Motionamp{i}   = zeros(length(periods),length(ang));
```

```
    Motionphase{i} = zeros(length(periods),length(ang));
  end

  for w = 1:N
    perindex = find(periods == per(w));
    angindex = find(ang == beta(w));
    Motionamp{DOF(w)}(perindex,angindex)   = mod(w);      % amplitude
    Motionphase{DOF(w)}(perindex,angindex) = phase(w);    % phase
  end

  % sort with respect to frequency
  freqMotion = 2*pi./periods;
  [freqs_sorted, freq_idx] = sort(freqMotion);

  for i=1:6
    Motionamp_sorted{i}   = Motionamp{i}(freq_idx,:);
    Motionphase_sorted{i} = Motionphase{i}(freq_idx,:);
  end

  vessel.motionRAO.w  = freqs_sorted';

  % scale Wamit Motion-data to SI (Wamit axes)
  vessel.motionRAO.amp{1}(:,:,1) = Motionamp_sorted{1}(:,:);
  vessel.motionRAO.amp{2}(:,:,1) = Motionamp_sorted{2}(:,:);
  vessel.motionRAO.amp{3}(:,:,1) = Motionamp_sorted{3}(:,:);
  vessel.motionRAO.amp{4}(:,:,1) = Motionamp_sorted{4}(:,:)*ULEN;
  vessel.motionRAO.amp{5}(:,:,1) = Motionamp_sorted{5}(:,:)*ULEN;
  vessel.motionRAO.amp{6}(:,:,1) = Motionamp_sorted{6}(:,:)*ULEN;

  % phase in rad: add pi for DOF with negative T_rao values (Fossen axes)
  vessel.motionRAO.phase{1}(:,:,1) = Motionphase_sorted{1}(:,:)*pi/180 - min((
  vessel.motionRAO.phase{2}(:,:,1) = Motionphase_sorted{2}(:,:)*pi/180 - min(
  vessel.motionRAO.phase{3}(:,:,1) = Motionphase_sorted{3}(:,:)*pi/180 - min(
  vessel.motionRAO.phase{4}(:,:,1) = Motionphase_sorted{4}(:,:)*pi/180 - min
  vessel.motionRAO.phase{5}(:,:,1) = Motionphase_sorted{5}(:,:)*pi/180 - min(
  vessel.motionRAO.phase{6}(:,:,1) = Motionphase_sorted{6}(:,:)*pi/180 - min(

end
```

```
%-------------------------------------------------------------------------
%% check if force RAOs are computed using the diffraction or Haskind option
%-------------------------------------------------------------------------
if exist([filename '.2']) && exist([filename '.3'])

   forceRAOinput = input('WAMIT force RAOs from  DIFFRACTION (0) or HASK

   if forceRAOinput == 0
      forceRAOfile = [filename '.3'];
      disp('WAMIT force RAOs from HASKIND  → vessel.forceRAO)')
   else
      forceRAOfile = [filename '.2'];
      disp('WAMIT force RAOs from HASKIND  → vessel.forceRAO')
   end

elseif exist([filename '.3'])
   disp('WAMIT force RAOs from DIFFRACTION  → vessel.forceRAO')
   forceRAOfile = [filename '.3'];
elseif exist([filename '.2'])
   disp('WAMIT force RAOs from HASKIND      → vessel.forceRAO')
   forceRAOfile = [filename '.2'];
end



format_string = '%n %n %n %n %n %n %n';   % Wamit manual page 4-8
[per,beta,DOF,mod,phase,realpart,impart] = textread(forceRAOfile,format_strin

N = length(per);

% Exstract all unique periods and wave directions
periods = unique(per);
ang     = unique(beta);

for i=1:6
   FKamp{i}   = zeros(length(periods),length(ang));
   FKphase{i} = zeros(length(periods),length(ang));
end
```

```
for w = 1:N
   perindex = find(periods == per(w));
   angindex = find(ang == beta(w));
   FKamp{DOF(w)}(perindex,angindex)   = mod(w);      % amplitude
   FKphase{DOF(w)}(perindex,angindex) = phase(w);    % phase
end

% sort with respect to frequency
freqFK = 2*pi./periods;
[freqs_sorted, freq_idx] = sort(freqFK);

for i=1:6
   FKamp_sorted{i}   = FKamp{i}(freq_idx,:);
   FKphase_sorted{i} = FKphase{i}(freq_idx,:);
end

vessel.forceRAO.w  = freqs_sorted';

% scale Wamit FK-data to SI (Wamit axes)
Fx = FKamp_sorted{1}(:,:) * rho_g*ULEN^2;
Fy = FKamp_sorted{2}(:,:) * rho_g*ULEN^2;
Fz = FKamp_sorted{3}(:,:) * rho_g*ULEN^2;
Mx = FKamp_sorted{4}(:,:) * rho_g*ULEN^3;
My = FKamp_sorted{5}(:,:) * rho_g*ULEN^3;
Mz = FKamp_sorted{6}(:,:) * rho_g*ULEN^3;

% Wamit axes
vessel.forceRAO.amp{1}(:,:,1) = Fx;
vessel.forceRAO.amp{2}(:,:,1) = Fy;
vessel.forceRAO.amp{3}(:,:,1) = Fz;
vessel.forceRAO.amp{4}(:,:,1) = Mx;
vessel.forceRAO.amp{5}(:,:,1) = My;
vessel.forceRAO.amp{6}(:,:,1) = Mz;

% phase in rad: add pi for DOF with negative T_rao values (Fossen axes)
vessel.forceRAO.phase{1}(:,:,1) = FKphase_sorted{1}(:,:)*pi/180 - min(0,T_rao(1
vessel.forceRAO.phase{2}(:,:,1) = FKphase_sorted{2}(:,:)*pi/180 - min(0,T_rao(
```

```
vessel.forceRAO.phase{3}(:,:,1) = FKphase_sorted{3}(:,:)*pi/180 - min(0,T_rao(
vessel.forceRAO.phase{4}(:,:,1) = FKphase_sorted{4}(:,:)*pi/180 - min(0,T_rao(
vessel.forceRAO.phase{5}(:,:,1) = FKphase_sorted{5}(:,:)*pi/180 - min(0,T_rao(
vessel.forceRAO.phase{6}(:,:,1) = FKphase_sorted{6}(:,:)*pi/180 - min(0,T_rao(

%--------------------------------------------------------------------------
%% check if wave drift data from the momentum or pressure option
%--------------------------------------------------------------------------
if exist([filename '.8']) && exist([filename '.9'])

   WDinput = input('WAMIT wave drift from MOMENTUM (0) or PRESSURE (1)'

   if WDinput == 0
      WDfile = [filename '.8'];
      disp('WAMIT wave drift data from MOMENTUM --> vessel.driftfrc')
   else
      WDfile = [filename '.9'];
      disp('WAMIT wave drift data from PRESSURE --> vessel.driftfrc)')
   end

elseif exist([filename '.8'])
   disp('WAMIT wave drift data from MOMENTUM --> vessel.driftfrc')
   WDfile = [filename '.8'];
   flag = 8;
elseif exist([filename '.9'])
   disp('WAMIT wave drift data from PRESSURE --> vessel.driftfrc')
   WDfile = [filename '.9'];
   flag = 9;
else
   error('No WAMIT *.8 or *.9 file')
end

% read WD data from *.8 or *.9 file
format_string = '%n %n %n %n %n %n %n %n';   % Wamit manual page 4-8
[per,beta1,beta2,DOF,mod,phase,realpart,impart] = textread(WDfile,format_str

N = length(per);
```

```
% Exstract all unique periods and directions with computed data
periods = unique(per);
ang     = unique(beta1);

% assumes that beta1 = beta2 (Wamit: IOPTN(9) = 1)
if abs(beta1(1:5)-beta2(1:5)) > 0.0001
   disp('Error: Run Wamit with IOPTN(8)=1 or IOPTN(9) = 1 such that beta1 = b
end

% extract wave drift amplitudes using the REAL part (IMAG part is zero)

for i=1:6
   WDamp{i} = zeros(length(periods),length(ang));
end

for w = 1:N
   perindex = find(periods == per(w));
   angindex = find(ang == beta1(w));
    % b-frame data DOF = {1,2,3,-4,-5,-6}, see Fig. 12.2 in the Wamit manual
    % h-frame data w.r.t. GLOBAL COORDINATES for DOF = {1,2,3,4,5,6}
   if DOF(w)==1 || DOF(w)==2 || DOF(w)==3 || DOF(w)==4 || DOF(w)==5 || DO
      WDamp{abs(DOF(w))}(perindex,angindex) = realpart(w);
      % real part is equal to mod corrected for phase/sign
   end
end

% sort with respect to frequency
freqWD = 2*pi./periods;
[freqs_sorted, freq_idx] = sort(freqWD);

for i=1:6
   WDamp_sorted{i} = WDamp{i}(freq_idx,:);
end

vessel.driftfrc.w  = freqs_sorted';

% scale Wamit WD-data to SI (Wamit axes) and change signs
Fx = T_rao(1) * WDamp_sorted{1}(:,:) * rho_g*ULEN;
```

```
Fy = T_rao(2) * WDamp_sorted{2}(:,:) * rho_g*ULEN;
Fz = T_rao(3) * WDamp_sorted{3}(:,:) * rho_g*ULEN;
Mx = T_rao(4) * WDamp_sorted{4}(:,:) * rho_g*ULEN^2;
My = T_rao(5) * WDamp_sorted{5}(:,:) * rho_g*ULEN^2;
Mz = T_rao(6) * WDamp_sorted{6}(:,:) * rho_g*ULEN^2;

% Wamit axes
vessel.driftfrc.amp{1}(:,:,1) = Fx;
vessel.driftfrc.amp{2}(:,:,1) = Fy;
%vessel.driftfrc.amp{4}(:,:,1) = Fz;
%vessel.driftfrc.amp{5}(:,:,1) = Mx;
%vessel.driftfrc.amp{6}(:,:,1) = My;
vessel.driftfrc.amp{3}(:,:,1) = Mz;

%-------------------------------------------------------------------------%
%% mirror data from 0-180 deg to 180-360 deg (symmetry about x-axes)
%-------------------------------------------------------------------------
vessel.headings = (pi/180)*(0:10:350);

% correct sign of flipped motions RAOs by modyfying the phases in sway,
% roll and yaw (2, 4, 6) by pi
rao_sign = [0 1 0 1 0 1];
for i = 1:6

   vessel.motionRAO.amp{i}(:,20:36,1) = vessel.motionRAO.amp{i}(:,flip(2:18),1
   vessel.motionRAO.phase{i}(:,20:36,1) = rao_sign(i) * pi + ...
      vessel.motionRAO.phase{i}(:,flip(2:18),1);

   vessel.forceRAO.amp{i}(:,20:36,1) = vessel.forceRAO.amp{i}(:,flip(2:18),1);
   vessel.forceRAO.phase{i}(:,20:36,1) = rao_sign(i) * pi + ...
      vessel.forceRAO.phase{i}(:,flip(2:18),1);
end

% correct sign of flipped motion RAOs amplitudes in surge, sway and yaw (1, 
rao_sign_WD = [1 -1 -1];
for i = 1:3
  vessel.driftfrc.amp{i}(:,20:36,1) = rao_sign_WD(i) * vessel.driftfrc.amp{i}(:,2:1
end
```

```
%------------------------------------------------------------------------
%% add zeros for velocity indexes 2-10, only one velocity U=0 for WAMT data
%------------------------------------------------------------------------
for i = 1:6
   vessel.motionRAO.amp{i}(:,:,2:10)   = zeros(length(vessel.motionRAO.w),36,
   vessel.motionRAO.phase{i}(:,:,2:10) = zeros(length(vessel.motionRAO.w),36
   vessel.forceRAO.amp{i}(:,:,2:10)    = zeros(length(vessel.forceRAO.w),36,9);
   vessel.forceRAO.phase{i}(:,:,2:10)  = zeros(length(vessel.forceRAO.w),36,9)
end

for i = 1:3
   vessel.driftfrc.amp{i}(:,:,2:10)   = zeros(length(vessel.driftfrc.w),36,9);
end

%------------------------------------------------------------------------
%% viscous damping
%------------------------------------------------------------------------
Bv = viscous(vessel);
vessel.Bv = Bv;

%------------------------------------------------------------------------
%% plots
%------------------------------------------------------------------------
if strcmp(plot_flag(1),'1')
   plotABC(vessel,'A');
   plotABC(vessel,'B');
end
if strcmp(plot_flag(2),'1')
   plotTF(vessel,'force','rads',1)
end
if strcmp(plot_flag(3),'1')
   plotTF(vessel,'motion','rads',1)
end
if strcmp(plot_flag(4),'1')
   plotWD(vessel,'rads',1)
end
```

```
%--------------------------------------------------------------------------
%% save data to vessel.mat
%--------------------------------------------------------------------------
save(vesselfile,'vessel');
disp(['Simulink structure <vessel> has been saved to: ' vesselfile '.mat']);
disp('-------------------------------------------------------------------')
```