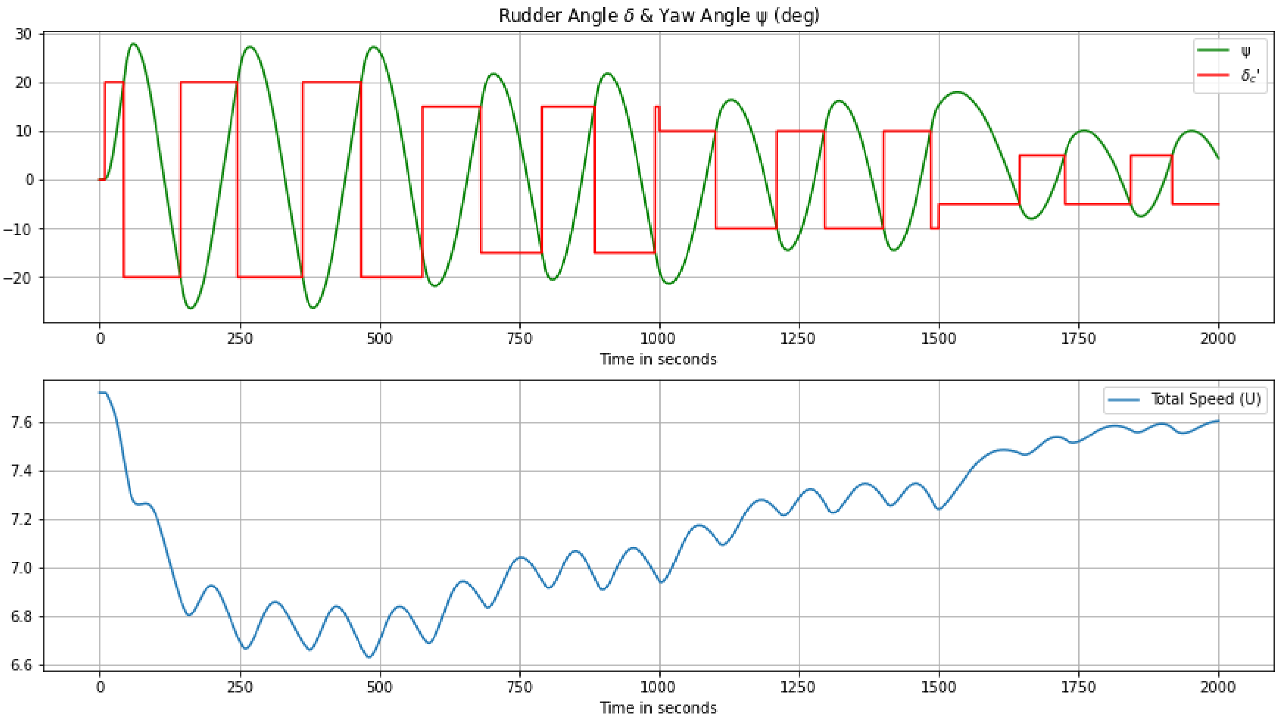
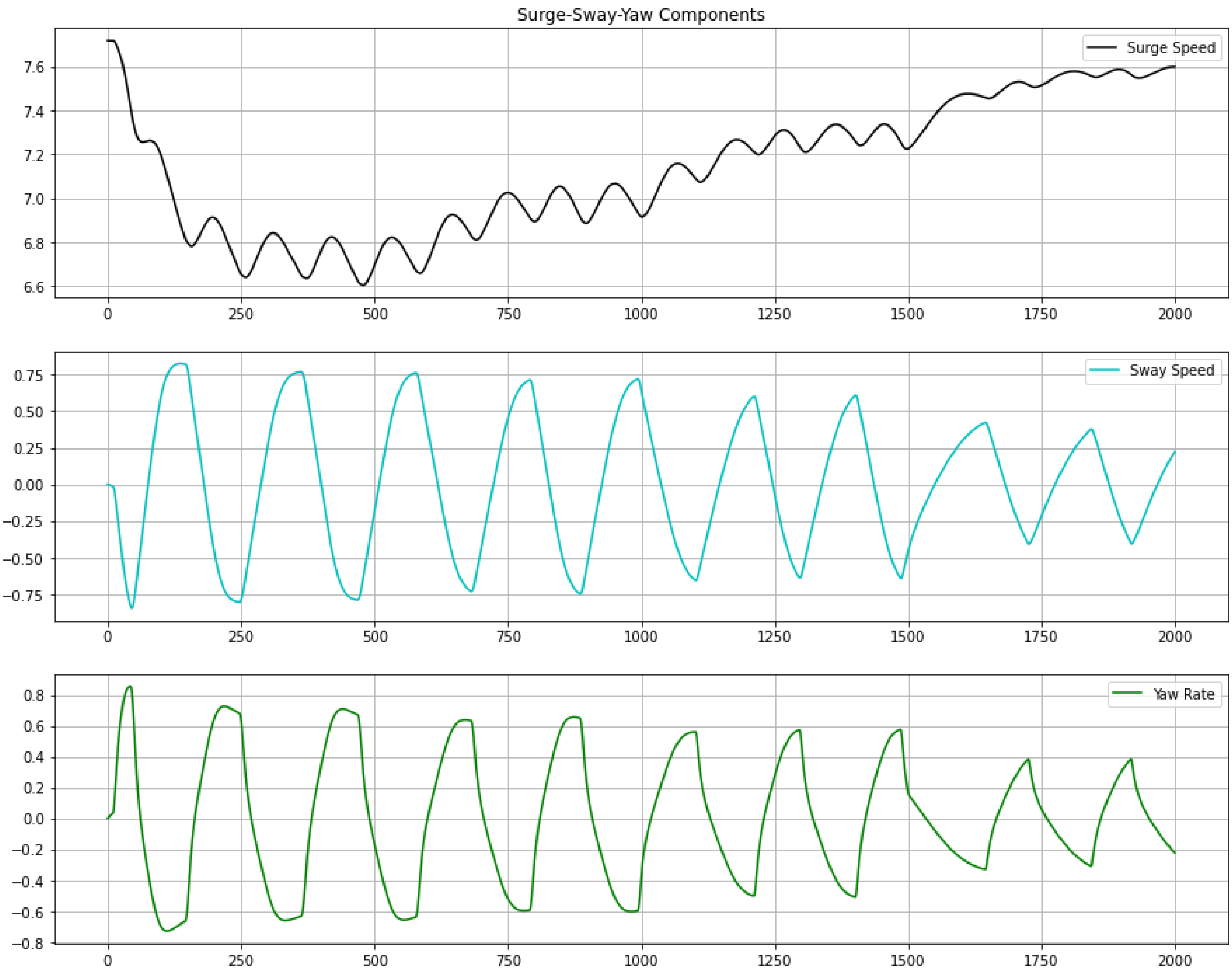
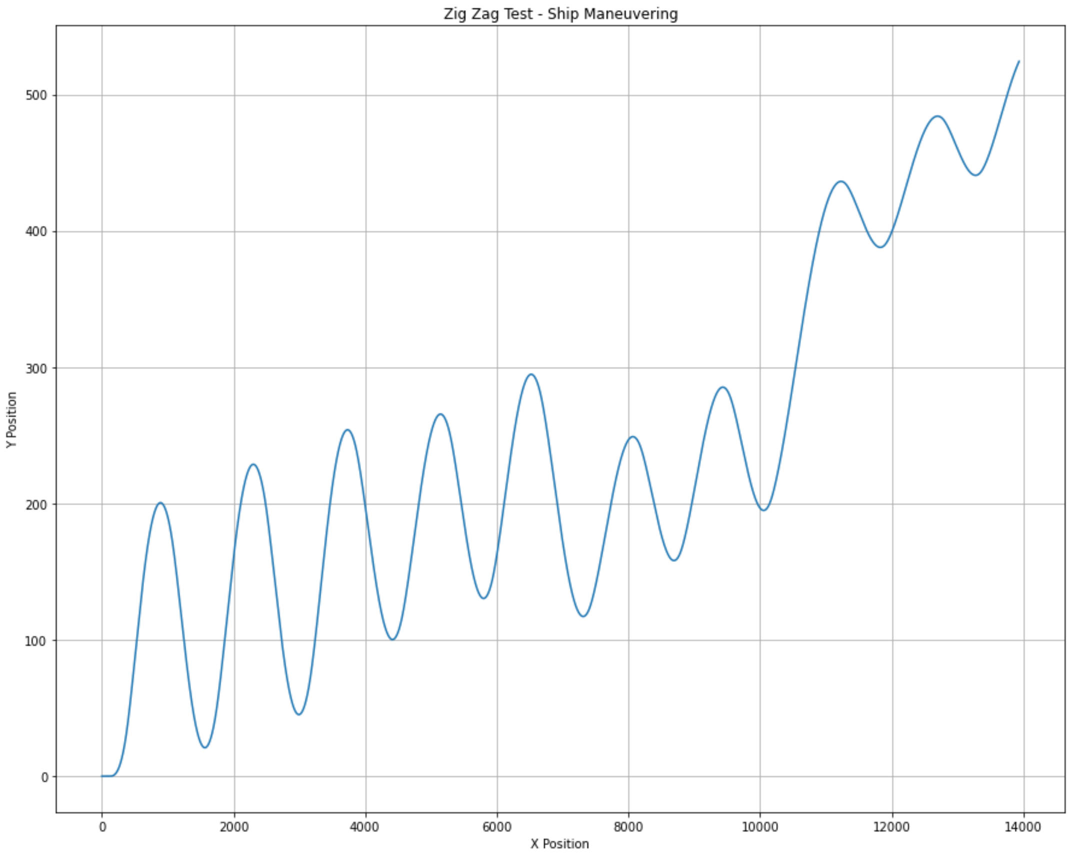
Data Simulation for Mariner Ship






Zig Zag Test - Ship Maneuvering
Y Position
X Position
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000
14000

```python
import numpy as np
import matplotlib.pyplot as plt
import zig_zag

"""
Generates the zigzag maneuver for given different type of ship

Author:   Sivaraman Sivaraj, Suresh Rajendran
Date:     01 December 2020

Reference Author :  Thor I. Fossen
Date:               23th July 2001

"""

Req_simulation_time = 2000           #Total simulation time (sec)
t_rudderexecute = 10     #time for rudder is executed at particular angle(sec) (bump in th
e graph,becuase of this)
h = 0.1                  #sampling time (sec)

print("Zig Zag test for given ship model is about to start...")

xt = np.zeros((7,1)) #x  = [ u v r x y psi delta ]' (initial values)
ui = 0;

t,u,v,r,x,y,psi,U,delta,DATA = zig_zag.activate('mariner',xt,ui,Req_simulation_time,t_rudd
erexecute,h,[20,20])


t_a = np.array(t)
u_a = np.array(u)
v_a = np.array(v)
r_a = np.array(r)
x_a = np.array(x)
y_a = np.array(y)
psi_a = np.array(psi)
U_a = np.array(U)
delta_a = np.array(delta)

t = t.tolist()

v = v.tolist()
r = r.tolist()
x = x.tolist()
y = y.tolist()
psi = psi.tolist()
U = U.tolist()
delta = delta.tolist()

zig_zag.plot_components_psi_delta_U(t,psi,delta,U)
zig_zag.plot_components_xy(x,y)


def Plot_simulated_Data1():
```

```python
    plt.figure(figsize=(15,12))

    plt.subplot(311)
    plt.plot(t[:len(t)-2],7.7175+u[:len(u)-2],'k',label= "Surge Speed")
    plt.grid(b=0.1)
    plt.legend(loc="best")
    plt.title("Surge-Sway-Yaw Components")

    plt.subplot(312)
    plt.plot(t[:len(t)-2],v[:len(v)-2],'c',label= "Sway Speed")
    plt.grid(b=0.1)
    plt.legend(loc="best")

    plt.subplot(313)
    plt.plot(t[:len(t)-2],r[:len(r)-2],'g',label= "Yaw Rate")
    plt.grid(b=0.1)
    plt.legend(loc="best")
    plt.show()


def Plot_simulated_Data2():
    plt.figure(figsize=(15,8))

    plt.subplot(211)
    plt.grid(b=0.1)
    plt.plot(t[:len(t)-2],delta[:len(t)-2],'-r')
    plt.plot(t[:len(t)-2],psi[:len(t)-2],'-b')

    plt.subplot(212)
    plt.plot(t[:len(t)-2],U[:len(U)-2],'m',label= "Total Speed")
    plt.grid()
    plt.legend(loc="best")
    plt.show()

Plot_simulated_Data1()
# Plot_simulated_Data2()

du = u_a[2:]-u_a[1:len(u)-1]
dv = v_a[2:]-v_a[1:len(v)-1]
dr = r_a[2:]-r_a[1:len(r)-1]
output1=[du,dv,dr]

u = u_a + 7.7175
u = u.tolist()
output_zig_zag = np.asarray([t,u,v,r,psi,U,delta])
np.savetxt("2000_sec_20_15_10_5.csv", output_zig_zag.T, delimiter=",")



#################################################################################
#########################  NEXT FILE ############################################
#################################################################################
```

```python
import numpy as np
import matplotlib.pylab as plt
import mariner

def euler_integration(xdot,x,h):
    """
    Integrate a system of ordinary differential equations using
    Euler's 2nd-order method.

    x_next = euler_integration(xdot,x,h)

    Parameters
    ----------
    xdot : dx/dt(k) = f(x(k))
    x : x(k)
    h     - step size

    Returns
    -------
    x_next - x(k+1)

    """
    a = np.array(x)
    b = np.array(xdot)
    return a + (h*b)

def activate(ship,x,ui,Req_simulation_time,t_rudderexecute,h,maneuver=[20,20]):
    """
    It performs the zig-zag maneuver

    Input Variables
    ----------
    ship    : ship model. Compatible with the models under .../gnc/VesselModels/
    x       : initial state vector for ship model
    ui      : [delta,:] where delta=0 and the other values are non-zero if any
    t_final : final simulation time

    t_rudderexecute : rudder's time control input is activated

    h       : sampling time

    maneuver : [rudder angle, heading angle]. Default 20-
20 deg that is: maneuver = [20, 20]
               rudder is changed to maneuver(1) when heading angle is larger than maneuver
(2)

    Returns
    -------
    t                = time vector
    u,v,r,x,y,psi,U,delta_c = time series

    """

    N = round(Req_simulation_time/h)               #number of samples
```

```
xout = np.zeros((N+1,9))

print("Simulating the Maneuver data.....")

u_ship=ui

for i in range(N):
    time = (i-1)*h
    if time > 500 and time < 1000:
        maneuver = [15,15]
    elif time>1000 and time<1500:
        maneuver = [10,10]
    elif  time>1500 :
        maneuver = [5,5]


    psi = x[5]*180/np.pi
    r   = x[2]

    if round(time) == t_rudderexecute:
        u_ship = maneuver[0]*np.pi/180

    if round(time) > t_rudderexecute:
        if psi >= maneuver[1] and r > 0:
            u_ship = -(maneuver[0]*np.pi)/180
        elif psi <= -maneuver[1] and r < 0:
            u_ship = (maneuver[0]*np.pi)/180

    xdot,U =  mariner.activate(x,u_ship)#feval(ship,x,u_ship)       #ship model

    x = euler_integration(xdot,x,h) #Euler integration
    ###########
    # xdot_q = np.squeeze(xdot).tolist()
    temp = list()
    temp.append(time)
    for j in range(6):
        temp.append(x[j])
    temp.append(U[0])
    temp.append(u_ship)
    xout[i,:] = temp     #[time,x[1:6].T,U,u_ship[0]]
    # print(temp)
    ############
    # print(i)
# maneuver
# time-series
t      = xout[:,0]
u      = xout[:,1]
v      = xout[:,2]
r      = xout[:,3]*180/np.pi
x      = xout[:,4]
y      = xout[:,5]
psi    = xout[:,6]*180/np.pi
U      = xout[:,7]
delta_c = xout[:,8]*180/np.pi
```

```python
    return t,u,v,r,x,y,psi,U,delta_c,xout



def plot_components_xy(x,y):
    plt.figure(figsize=(15,12))
    plt.grid()
    plt.plot(x[:len(x)-2],y[:len(y)-2])
    plt.xlabel("X Position")
    plt.ylabel("Y Position")
    plt.title("Zig Zag Test - Ship Maneuvering")
    plt.show()


def plot_components_psi_delta_U(t,psi,delta_c,U):
    plt.figure(figsize=(15,8))

    plt.subplot(211)
    plt.plot(t[:len(psi)-2],psi[:len(psi)-2],"g",label="ψ")
    plt.plot(t[:len(U)-2],delta_c[:len(U)-2],'r',label = "$ \delta_c $'")
    plt.grid()
    plt.legend(loc="best")
    plt.xlabel("Time in seconds")
    plt.title("Rudder Angle $ \delta $ & Yaw Angle ψ (deg)")


    plt.subplot(212)
    plt.plot(t[:len(U)-2],U[:len(U)-2], label ="Total Speed (U)")
    plt.grid()
    plt.legend(loc = "best")
    plt.xlabel("Time in seconds")
    plt.show()



#################################################################################
#########################  NEXT FILE ############################################
#################################################################################

import numpy as np
import matplotlib.pyplot as plt
import math

def activate(x,ui,U0 = 7.7175):
    """
    Parameters
    ----------
    x : [ u v r x y psi delta]
    ui : commanded rudder angle (rad)
    U0 : nominal speed (optionally). Default value is U0 = 7.7175 m/s = 15 knots.

    Returns
    -------
```

```
    xdot : Time derivative of the state vector
    U : speed


    Descriptions:
        for the Mariner class vessel L = 160.93 m, where
            u     = pertubed surge velocity about Uo (m/s)
            v     = pertubed sway velocity about zero (m/s)
            r     = pertubed yaw velocity about zero (rad/s)
            x     = position in x-direction (m)
            y     = position in y-direction (m)
            psi   = pertubed yaw angle about zero (rad)
            delta = actual rudder angle (rad)


    Reference: M.S. Chislett and J. Stroem-Tejsen (1965). Planar Motion Mechanism Tests
               and Full-
Scale Steering and Maneuvering Predictions for a Mariner Class Vessel,
               Technical Report Hy-5, Hydro- and Aerodynamics Laboratory, Lyngby, Denmark.


     Author:    Trygve Lauvdal
     Date:      12th May 1994
     Revisions: 19th July 2001 (Thor I. Fossen): added input/ouput U0 and U, changed order
 of x-vector
                20th July 2001 (Thor I. Fossen): replaced inertia matrix with correct valu
es
                11th July 2003 (Thor I. Fossen): max rudder is changed from 30 deg to 40
                                deg to satisfy IMO regulations for 35 deg rudder execute



    """
    # Normalization variables
    L = 160.93
    u1 = U0+x[0]
    U = np.sqrt((u1**2) +(x[1]**2))

    #Non-dimensional states and inputs
    delta_c = -ui   #% delta_c = -ui such that positive delta_c -> positive r

    u     = x[0]/U
    v     = x[1]/U
    r     = x[2]*L/U
    psi   = x[5]
    delta = x[6]

    #Parameters, hydrodynamic derivatives and main dimensions

    delta_max  = 40           #max rudder angle      (deg)
    Ddelta_max = 5           #max rudder derivative (deg/s)

    m  = 798e-5
    Iz = 39.2e-5
```

```
    xG = -0.023

    [Xudot,Xu,Xuu,Xuuu,Xvv,Xrr,Xdd,Xudd,Xrv,Xvd,Xuvd] = [-42e-5,-184e-5,-110e-5,-215e-5,-
899e-5,
                                                         18e-5,-95e-5,-190e-5,798e-5,93e-
5,93e-5]

    [Yvdot,Yrdot,Yv,Yr,Yvvv,Yvvr,Yvu,Yru,
     Yd,Yddd,Yud,Yuud,Yvdd,Yvvd,Y0,Y0u,Y0uu ]=[-748e-5,-9.354e-5,-1160e-5,-499e-5,-8078e-
5,15356e-5,
                                               -1160e-5,-499e-5,278e-5,-90e-5,556e-5,278e-
5,-4e-5,
                                               1190e-5,-4e-5,-8e-5,-4e-5]

    [Nvdot,Nrdot,Nv,Nr,Nvvv,Nvvr,Nvu,Nru,
     Nd,Nddd,Nud,Nuud,Nvdd,Nvvd,N0,N0u,N0uu]=[4.646e-5,-43.8e-5,-264e-5,-166e-5,1636e-5,
                                              -5483e-5,-264e-5,-166e-5,-139e-5,45e-5,-
278e-5,
                                              -139e-5, 13e-5,-489e-5,3e-5,6e-5,3e-5]

    # Masses and moments of inertia
    m11 = m-Xudot
    m22 = m-Yvdot
    m23 = m*xG-Yrdot
    m32 = m*xG-Nvdot
    m33 = Iz-Nrdot

    #Rudder saturation and dynamics
    if abs(delta_c) >= (delta_max*np.pi)/180:
        delta_c = np.sign(delta_c)*delta_max*np.pi/180

    delta_dot = delta_c - delta

    if abs(delta_dot) >= Ddelta_max*np.pi/180:
        delta_dot = np.sign(delta_dot)*Ddelta_max*np.pi/180

    # Forces and Moments

    X = Xu*u + Xuu*(u**2) + Xuuu*(u**3) + Xvv*(v**2) + Xrr*(r**2) + Xrv*r*v + Xdd*(delta**
2)+\
        Xudd*u*(delta**2) + Xvd*v*delta + Xuvd*u*v*delta

    Y = Yv*v + Yr*r + Yvvv*(v**3) + Yvvr*(v**2)*r + Yvu*v*u + Yru*r*u + Yd*delta+\
        Yddd*(delta**3) + Yud*u*delta + Yuud*(u**2)*delta + Yvdd*v*(delta**2) +\
        Yvvd*(v**2)*delta + (Y0 + Y0u*u + Y0uu*(u**2))

    N = Nv*v + Nr*r + Nvvv*(v**3) + Nvvr*(v**2)*r + Nvu*v*u + Nru*r*u + Nd*delta +\
        Nddd*(delta**3) + Nud*u*delta + Nuud*(u**2)*delta + Nvdd*v*(delta**2) +\
            Nvvd*(v**2)*delta + (N0 + N0u*u + N0uu*(u**2))

    # Dimensional state derivative
    detM22 = m22*m33-m23*m32

    xdot = [X*((U**2)/L)/m11 ,
```

```
        -(-m33*Y+m23*N)*((U**2)/L)/detM22,
        (-m32*Y+m22*N)*((U**2)/(L**2))/detM22,
        (np.cos(psi)*(U0/U+u)-np.sin(psi)*v)*U,
        (np.sin(psi)*(U0/U+u)+np.cos(psi)*v)*U,
        r*(U/L),
        delta_dot]

    return xdot,U
```