JMWE (https://www.jmwe.org) is an open, peer-reviewed journal published under the CC BY-NC-SA 4.0 Creative Commons License. Image adapted from Anton van den Wyngaerde, 1653.

OpenVPP: An Open-Source Dynamic Velocity Prediction Program for sailing cargo vessels

Sergio E. Perez, Department of Marine Engineering, U.S. Merchant Marine Academy, Kings Point, NY, perezs@usmma.edu

Introduction: OpenVPP is a dynamic, open-source velocity prediction program (VPP) designed for large sailing cargo vessels. The purpose of this manual is to document the algorithm and provide instructions/best practices for the program's use. This document is available from www.jmwe.org.

With some knowledge of ships and elementary Python programming skills, users of OpenVPP can evaluate the sailing performance of sail-assisted and solely-sail-propelled ships. Required are basic ship data as well as hull and sail lift and drag parameters. The program and this document are published under Creative Commons license CC BY-NC-SA 4.0.

The software

OpenVPP assumes that ship roll (heel) due to wind force is less than about 10 degrees, and therefore has little effect on the performance of a large sailing ship. This is a reasonable assumption since, for similar hulls, heeling moment varies by the 3rd power of the ship length, while righting moment varies by the 4th power [1]. Vessels of this magnitude are then likely to be very resistant to roll.

OpenVPP also assumes that the vessel is balanced in yaw, so the effect of rudders is not considered. This is appropriate for the purpose of evaluation and comparison of designs as part of a first loop of a design-spiral. The addition of rudders to the code should not be difficult once rudder drag and lift data are known.

OpenVPP is written in Python language, and uses an *explicit or dynamic scheme* to determine sailing vessel speed and track.

There are two general techniques for solving the equations which result from analyzing the forces on a sailing vessel. The most common method is by setting the sum of forces to zero, and solving the algebraic equations simultaneously, or *implicitly*. This is sometimes referred to as an *equilibrium* method. The second technique, used in OpenVPP, involves allowing the vessel to begin at some initial velocity and direction, and suddenly applying a wind speed, from which forces on the vessel can be determined and hence its acceleration is known. The vessel is then allowed to accelerate through a small period of time or *time step*, after which the vessel has a new speed and direction. After many repeated time steps (marching forward in time), the vessel eventually reaches a speed and heading that

no longer change. This type of solution is known as, *explicit,* or *dynamic*, and in this case results in differential equations (sum of forces = m a = m dv/dt), whereas the equilibrium solution results in simultaneous algebraic equations. This makes the explicit technique seem more complicated, but this is not the case.

Implicit techniques are solved through what can be referred to as intelligent trial and error, and have the advantage of usually converging (reaching a solution to the simultaneous equations) more quickly than explicit schemes. However, difficulties can be encountered if a good estimate of the final solution is not known.

The advantage of the explicit scheme is that the solution method is easier. Equations do not need to be solved simultaneously. Instead, the known velocity and orientation at time = 0 are used to extrapolate to future steady values. In addition, the technique may be used to solve for unsteady parameters such as acceleration, rate of turn or response to wave action.

OpenVPP was first developed in JustBASIC language (https://www.justbasic.com) because of its simplicity. After the algorithm was developed, the switch was made to Python language, which has the advantage of running considerably faster as well as providing 15-digit precision in its most basic form. The change from 8 to 15-digit precision greatly improved the performance of the program.

Using Python and a modest home laptop (Intel i7 64-bit processor), the program requires 2-5 minutes (roughly some 200,000 loops) to run through one set of calculations involving varying the drift angle from 1 to 25 degrees in increments of 1 degree, at a constant sail angle of attack. For a complete run, this cycle is repeated between 5-10 times at a variety of sail angles of attack, which results in overall calculation times that are not overly impractical.

In its most basic form, time-marching can be very simply done using the well-known Euler's method. However, it was found that solutions were oscillating with time, and at quite a few apparent wind angles no solution was achieved. For this reason, a 4th order Runge-Kutta algorithm was used for time-marching, which has proven to be more effective. Readers are referred to Reference [2] for a very clear explanation of Euler and Runge-Kutta methods.

OpenVPP uses English units, selected because OpenVPP was to be tested against the known output of [3] for 15,000 CDWT Prolss-rigged sailing vessel of 525 ft LWL. All vessel data programmed into the OpenVPP software are for this vessel, and may easily be changed. More information about the vessel and its Prolss Dynarig sails may be found in references [3], [4] and [5].

The program takes as input the Beaufort Scale level, which is converted from m/s to ft/s as shown below, taken from OpenVPP. The equation is taken from [3].

```
vw = 0.836*beau**(1.5)*3.28  #calculates wind speed from beaufort and converts to
```

The Beaufort number is used to determine the effect of wave action on the speed of the vessel using a relation from [3]. The equation calculates a correction factor "corr" to the drag force on the ship, as a function of apparent wind speed, LWL, wind speed and vessel orientation to the apparent wind thetaap:

```
corr = 1+(vap/100.0)**2*750.0/lwl*math.exp(1-(thetaap/40.0)**4)  #correction
factor for effect of waves from MARAD 1975
```

It is not clear how the authors of [3] determined this equation, as no reference was cited in their report. Most likely it was based on the authors' experience as naval architects/marine engineers.

The general solution procedure is shown below:

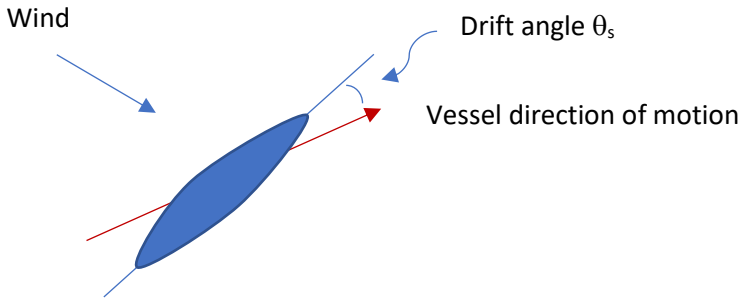
Solution procedure:

- 1 - Input a drift (leeway) angle, time step and Beaufort number. Select sail angle of attack. Program will solve for ship speed and ship path relative to wind.
- 2 - Program assumes an initial velocity of 10 knots, with vessel path at 90 degrees to wind. Ship orientation is relative to wind direction.
- 3 - Program solves for forces on sails and hull using ship speed and angle to wind.
- 4 - Program calculates acceleration of ship along vessel path, then a new ship speed and a new ship path at the end of the time step.
- 5 - If all of the forces in the direction of ship path cancel each other (sum of forces = 0), the program returns to step 1 and a new drift angle is used. Drift angles from 1 – 20 are run.
- 6 - If all of the forces in the direction of ship path do not cancel each other (sum of forces not equal) then go back to step 3.
- 7 – Select a new angle of attack and repeat 1 – 7. (Steps 1 -6 around 200,000 loops and 2 -3 minutes). About 5 angles of attack usually run.

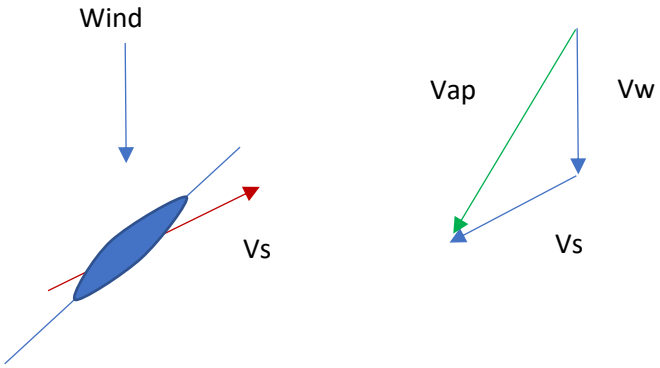
Aerodynamic forces on sails and hull

An OpenVPP run is started by inputting a drift angle (the angle between the ship's longitudinal axis and the ship's actual track) and a wind velocity. At time t = 0, the ship is randomly assumed to be moving at a speed of 10 knots relative to the earth and an angle of 90 degrees to the true wind direction. The true wind direction is always the datum for ship movement; the true wind direction (direction from which wind comes) can be thought of as true north on a chart, or a direction of zero degrees. The ship is allowed to move in a virtual 2-dimensional space using the true wind direction as reference, and the ship track is restricted to angles between 0 (directly into the wind) and 180 degrees (running with the wind).

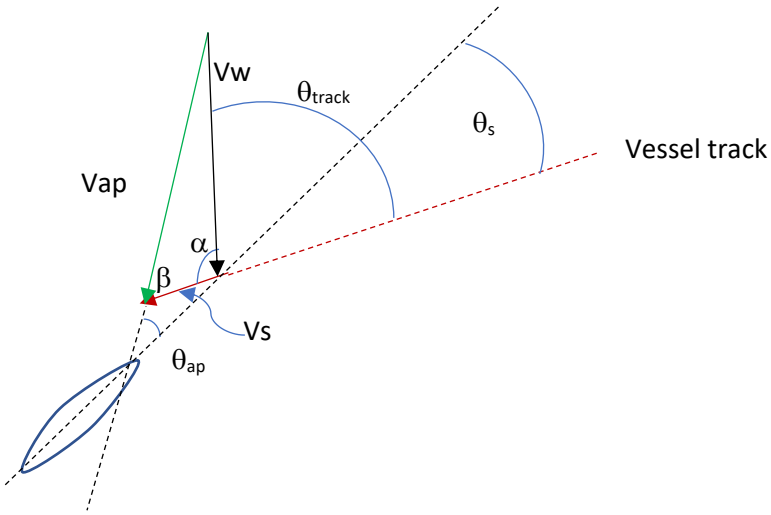
Sailing vessels usually do not point in the direction of motion or track (shown in red in all diagrams). Ships must move at an angle to the direction of motion to minimize downwind motion. The angle between the vessel longitudinal plane and the vessel track is referred to as the drift angle thetas θ_s.



A moving vessel experiences an apparent wind Vap that is the vector sum of the true wind Vw and the flow into the vessel due to the vessel's motion Vs along its track, as shown below.



The important angles involved are shown in the diagram below:



From the diagram above we can write the equations:

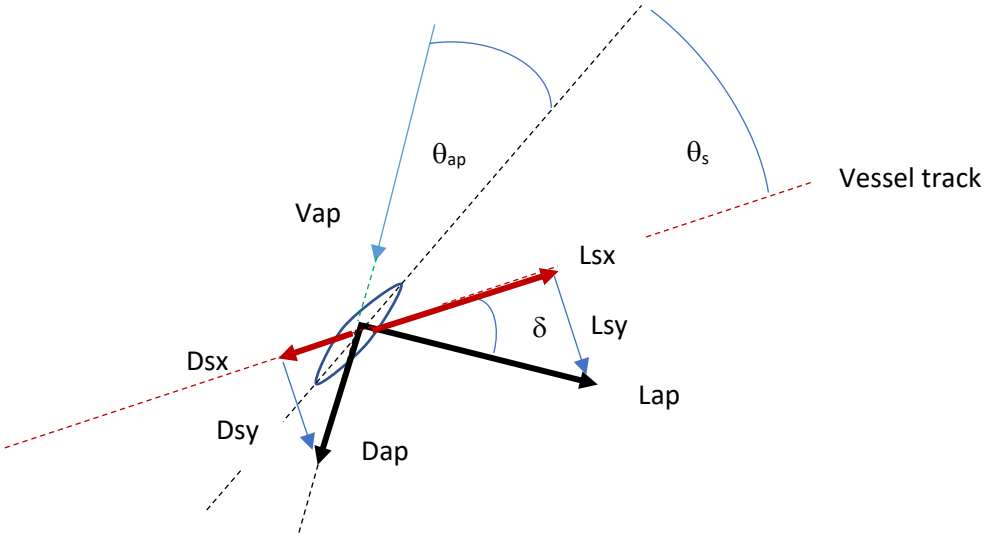
$$\alpha = 180 - \theta track \quad [1]$$

$$Vap^2 = Vs^2 + Vw^2 - 2\,Vs\,Vw\cos\alpha \quad [2]$$

$$\beta = arc\cos\left[\frac{Vw^2 - Vap^2 - Vs^2}{-2\,Vap\,Vs}\right] \quad [3]$$

$$\theta ap = \beta - \theta s \quad [4]$$

The sail lift force Lap is considered perpendicular to the apparent wind and drag Dap is colinear with the apparent wind. The projection of Lap and Dap onto the vessel track are important. These are Lsx and Dsx. Also required are the sail forces perpendicular to the ship track, Lsy and Dsy.



Using the diagram above as reference, forces along the track to the right are considered positive, and forces perpendicular to the track and down are negative:

The equations below describe the aerodynamic (sail) forces on the vessel:

$$\delta = 90 - \theta ap - \theta s \quad [5]$$

$$Lsx = Lap * \cos\delta \quad [6]$$

$$Lsy = Lap * \sin\delta \quad [7]$$

$$Dsx = Dap * \cos(\theta ap + \theta s) \quad [8]$$

$$Dsy = Dap * \sin(\theta ap + \theta s) \quad [9]$$

The lift and drag forces are obtained by:

$$Lap = \frac{1}{2}\,\rho Vap^2 A\,Cl \qquad [10]$$

$$Dap = \frac{1}{2}\,\rho Vap^2 A\,Cd \qquad [11]$$

Where ρ is the air density, A is the sail area, Vap is the apparent wind, Cl is the lift coefficient and Cd is the drag coefficient. Cl and Cd are obtained from [4].

The Cl and Cd values from [4] come from wind tunnel tests in which a model of a 6-masted merchant sailing vessel with Prolss Dyna-Rig sails is placed on the floor of the wind tunnel, at a variety of vessel angles into the wind (thetaap). At each thetaap setting, the sails were oriented through a range of angles of attack. Cl, Cd and moment coefficients were calculated at each setting. Lift and drag measurements were made for the entire model, not just the sails; in other words the influence of the hull is included in all measurements.

The OpenVPP software has a section in which Cl and Cd trendlines are given, for specified angles of attack at a variety of vessel angles into the wind (thetaap). Before an OpenVPP run, the user must select the trendline to use by "commenting out" all other trendlines.

The trendlines are mostly 4th degree polynomials, but some are 3rd and 5th degree. The higher the degree, the higher the accuracy of the trendline, but it was found that this could result in convergence difficulties. For this reason, the most accurate polynomial fit was not always selected – rather, the lowest order polynomial fit that gave reasonably good results was used.

Figure 1 below show the lift and drag data extracted from [4] for a variety of sail angles of attack. These were fit to polynomials, and the equations used may be seen on the OpenVPP source code.

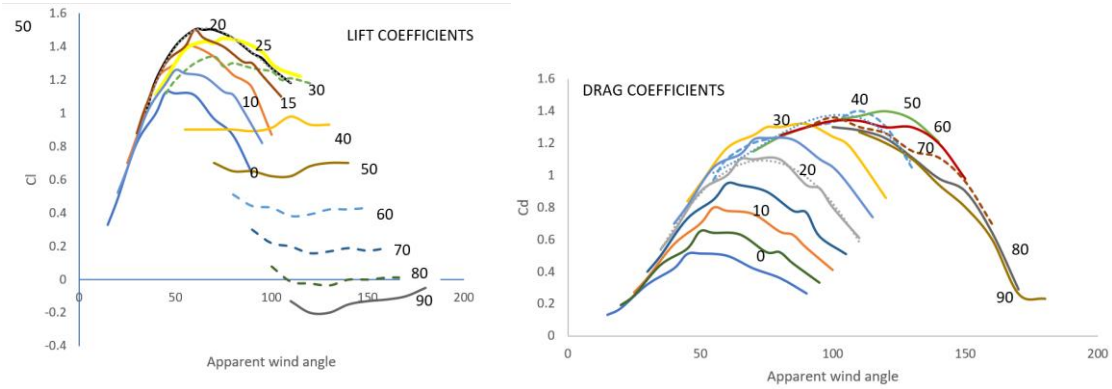


Figure 1. Dynarig Lift and drag coefficients at a variety of angles of attack and apparent wind angle. Extracted from [4].

When trendlines are employed, it is vital that the range of the data is not exceeded. OpenVPP does this by ensuring the solution is between the proper maximum and minimum fit limits.

It is also noted that when using these polynomial trendlines, it can at times be extremely important to use a seemingly exaggerated number of digits in the trendline coefficients. It is also emphasized that the

Cl and Cd values from [4] are necessarily in reference to the apparent wind angle to the ship; Cl is perpendicular and Cd is parallel to the apparent wind angle.

To determine the thrust and heel coefficients (Cx and Cy, where Cx applies along the ship's apparent heading (or along the ship's longitudinal axis) and Cy applies perpendicular to the ship's apparent heading – the reader is reminded that the difference between the apparent heading and actual heading is the drift angle of the vessel)):

$$Cx = Cl * \sin(\theta ap) - Cd * \cos(\theta ap) \quad [12]$$

$$Cy = Cl * \cos(\theta ap) + Cd * \sin(\theta ap) \quad [13]$$

Using data from [4], Figure 2 below plot below shows Cx and Cy.

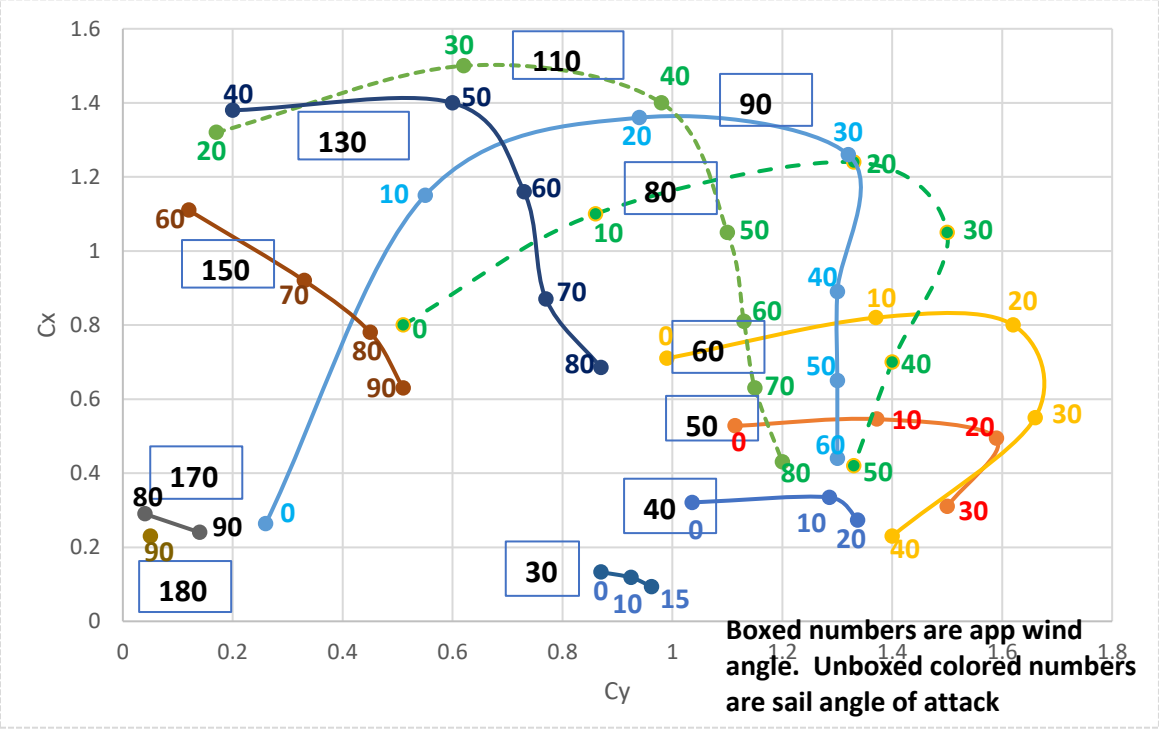


Figure 2. Thrust and heeling coefficients.

The plot is less confusing than it first appears, and is extremely useful for knowing approximately which angles of attack to use in an OpenVPP run. The plot can serve as a labor-saving device.

The x and y axes are the coefficients Cy and Cx described by equations 12 and 13. Cx and Cy are important because they represent the relative magnitude of sail thrust forces in the apparent direction of motion (Cx) and in a direction perpendicular to the ship's apparent motion (Cy). The perpendicular heeling forces obviously are undesirable and large thrust forces are highly sought.

The numbers enclosed by boxes indicate the apparent wind angle. Each curve is made up of several points with a number next to each point. The numbers indicate the angle of attack at that point. For

example, the solid blue line applies to winds perpendicular to the vessel fore-and-aft line, and is made of sail angles of attack between 0 and 60 degrees. It appears that, for the solid blue line, angles of attack between 10 and 20 degrees might give the best combination of high Cx and low Cy. Therefore, these are the angles of attack that should be investigated for thetaap value of 90 degrees.

The software takes as input the drift angle thetas and Beaufort number BF, and thetaap is not known until the end of a run. However, one can make a reasonable guess of thetaap after some moderate level of experience, allowing the user to select an appropriate range of angles of attack to attempt.

Underwater Hull Forces:

The hull forces are more straight-forward than the sail forces, and are calculated directly in relation to the vessel track as a function of the drift angle. Lift is in the direction perpendicular to vessel motion, while drag is colinear with and in the opposite direction of vessel motion.

The drag force is calculated by assuming that drag can be broken up into two components, one from the vessel moving at zero angle of attack to the vessel track and another from the vessel being at an inclined angle to the vessel track. The sum of these two is the total drag on the hull.

The "zero angle" portion comes from towing tank test results from Ref [6]. The Ct plot for a Series 60 hull with Block Coefficient Cb = 0.6 was used to generate a 4th degree polynomial for resistance in lbf units (Equation 18 below) as a function of ship velocity vs. The curve fit applies for vessel velocities between 0 and 42 ft/s, and is shown in Figure 3 below. It should be noted that the resistance coefficient Ct in [6] needed to be divided by 1000 to achieve the correct resistance values. In addition, the area used to calculate the resistance force was the wetted area of the hull, taken to be 47,000 ft^2 [6].

The zero angle drag force is multiplied by a correction factor "corr" [3], which accounts for increased drag due to wave action. The multiplication by 1.1 accounts for fouling of the ship hull due to normal use [3].

The "angled hull" resistance, referred to as induced drag (Eq. 15, 16), is taken from [5]. The lift force on the hull is calculated using the "clhull" portion of the code below (Eq. 14). Angled hull forces are calculated using the vessel displacement to the 2/3 power, as shown by the "lhy" term in Eq. 20 below.

```
clhull = (1.9735e-4*thetas**2+ 3.6154e-3*thetas)                    [14]

cdhullinduced = (1.5219e-4*thetas**2- 4.5123e-4*thetas)             [15]

draghullinduced = 0.5*64/32.2*vs**2*cdhullinduced*displ**0.6667     [16]

corr = 1+(vap/100.0)**2*750.0/lwl*math.exp(1-(thetaap/40.0)**4)     [17]

draghulltow = (4.511886E-01*vs**4 - 2.609315E+01*vs**3 + 5.835723E+02*vs**2 -
2.420343E+03*vs)*corr*1.1                                           [18]
dhx =draghullinduced + draghulltow                                  [19]
lhy = 0.5*64/32.2*vs**2*displ**0.6667*clhull                        [20]
```

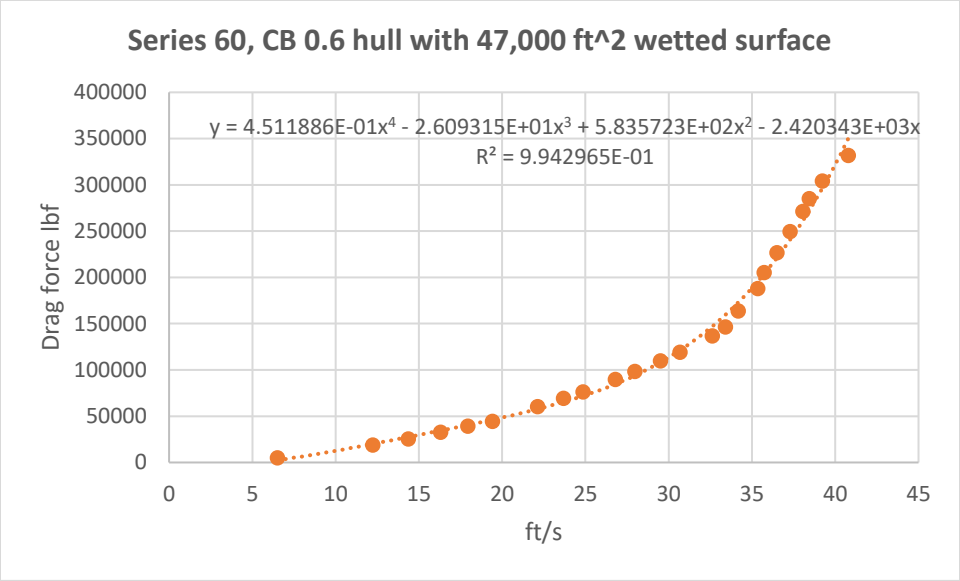
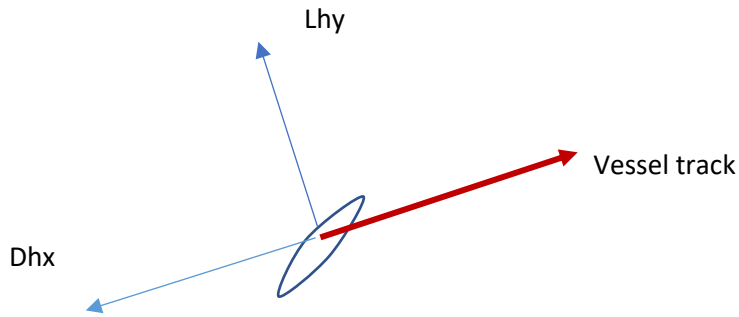


Figure 3. Towing tank results from [6]

The diagram below shows the hull lift and drag forces:



The forces are summed in x direction (along ship track) and perpendicular to ship track (y), as demonstrated in equations 21 and 22:

```
forcex = lsx-dsx-dhx                                    [21]

forcey = lhy-dsy-lsy                                    [22]
```

Once the forces are known we can calculate the new velocity and ship direction by marching forward in time. First discussed is Euler's method, sometimes referred to as 1st order Runge Kutta. Euler's method is essential for understanding and using 4th order R-K.

Solution by Euler's method (1st order R-K)

We write Newton's Law, expressed using a forward difference in time:

$$\sum \Sigma = \quad Forces = ma = m\frac{dV}{dt} = m\,\frac{(V_t+1-V_t)}{\Delta t} \qquad [23]$$

Where Δt is a time step, and the subscripts Vt and Vt+1 represent a velocity at a time step t and a velocity at the next future time step (t+1).

Solving for Vt, we obtain:

$$V_{t+1} = \frac{\Sigma Forces}{m}\Delta t + V_t \qquad [24]$$

The sum of forces/mass is the acceleration, or the rate of change of velocity with time. It is then a slope, and will be referred to here as rungslopex in the x direction and rungslopey in the y direction.

The velocity at a time step t+1 is then a function of the velocity at a previous time step t and the slope, permitting forward time marching that begins with an initial known velocity and initially known forces at time t:

$$V_{t+1} = \text{rungslope}\ \Delta t + V_t \qquad [25]$$

Slopes rungslopex and rungslopey are calculated:

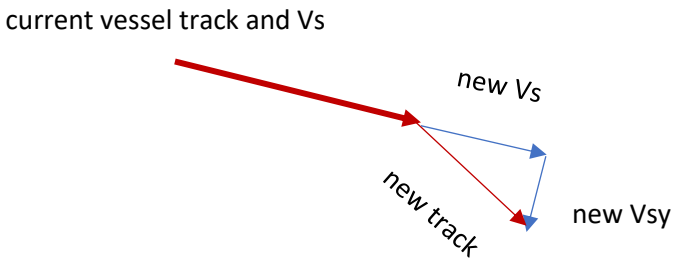
rungslopex = forcex/vessel mass [26]

rungslopey = forcey/vessel mass [27]

We calculate a new velocity in x and y directions (note the equations apply to 4th order Runge-Kutta. For Euler's method there would be no division by 2.0 in the equations):

```
vs = vs + rungslopex*tstep/2.0   #'velocity in x at mid step   [28]
vsy = vsy + rungslopey*tstep/2.0 #'vel in y at mid time step   [29]
```

We calculate a new vessel track, by adding artctan(vsy/vs) to the previous track:




```
changeslope = math.atan(abs(vsy/vs))
                    thetatrackR = changeslope+ thetatrackR          [30]
                    if(thetatrackR < 0.96):
                        thetatrackR = 0.96 # the limit is set at 55 degrees into
                        the wind
```

Once a new velocity Vs and track thetatrack are known, the procedure is repeated by solving equations 1-30 using the new velocities and track, until forcex (sum of forces along vessel track) are zero or close to zero. When this happens, the vessel can be assumed to have reached a steady velocity, and the solution has been found.

It should be noted that it is unnecessary to wait until forcex = 0. Doing so might never yield a result, or require large computation times. For this case, the criterion used for stopping the calculations (balanced forces) was when the sum of forcex was less than 10 lbf, although 100 or even 1000 lbf seemed to give results that were about as good. This is the purpose of the 'tol' parameter in the program, set to 10 ; when sumforcex is less than tol, the program stops and proceeds to the next drift angle.

4th order Runge-Kutta

Using the 4th order R-G scheme, each time step is broken down into 4 steps (referred to as inner steps here). The reader is reminded that the drift angle thetas and wind velocity are given from the previous time step, and that the goal is to find the vessel speed Vs and vessel track thetatrack.

- Inner step 1 of 4:
  K1 = rungslopex at the beginning of the inner time step (calculated from forcex in the previous time step). Calculate new velocities vs and vsy using equations 25 and 26, but using a half-time step. Calculate new apparent velocity, apparent wind angle, and sum of forces in x and y.

- Inner time step 2 of 4:
  Calculate rungslopex k2 based on previous step's sum of forces. Calculate new apparent velocity, apparent wind angle, and sum of forces using k2, again over a half-time step.

- Inner time step 3 of 4:
  Calculate runglsopex k3 based on previous step's forces and go through an entire time step using k3. As before, calculate sumforcex and sumforcey.

- Inner time step 4 of 4:

  Calculate runglsopex k4 based on previous results, then use a weighted average to calculate the final k:

  k = k1/6 + k2/3 + k2/3 + k4/6

  Using Eq [25], calculate a new vs and vsy, as well as a new track. If forcex is less than tol, the program stops and writes the solution to a file. If the program has not converged, a new Runge-Kutta loop is begun and the process repeated.

Before making a VPP Run/Tips:

1.Enter vessel data and run parameters into the program. Thetas is the drift angle, and the program will loop through a variety of thetas values up to the parameter "maxthetas", in increments of 1. The thetas near the top of the program represents the very first value of thetas attempted minus 1. For a full range of thetas, 2 to 20 is appropriate. The vessel data are explained in the code, with comments.

```
thetas = 2.0        #lowest drift angle value wanted minus 1
maxthetas = 20.0    # highest drift angle wanted
tstep = 0.001           #time step in seconds
tol = 10.0              # when summation of forces = 10 lbf, the program stops..
this number can go as high as 100-1000
nloops = 1e6            #max no of time step to do... 1e6 usually good
pi = 3.14159
# ship parameters
conv = 57.296           #radians = degrees/conv
lwl = 565.0             #ft waterline length of 15 kDWT Mariner type hull
wt = 40.0e6             # ship mass in lbm also the weight in lbf
displ = wt/64.0         #ship displacement in ft^3
GM = 6.3                #ship GM ft
lp = 525.0              #length between perpendiculars ft
asails = 88000.0        #sail area in ft^2
```

2. Remove the comment indicator (#) from the sail angle of attack equations that you wish to run. Parameters lowtheta and hitheta represent the minimum and maximum apparent wind angles used to make a trendline. Solutions found using relative wind angles outside of the range are not used.

3. Usually, approximately 200,000 loops are required per angle of attack, for a run of about 20 drift angles, requiring roughly 2-5 minutes. After a run, the user might try a different set of angles of attack to

ensure the best angle is found (highest resulting vessel speed) – not all angles of attack yield a high speed.

4. Extract the data from the output file vppdat. Microsoft Notepad works well for storing the data. The output is thetrack in degrees (angle of vessel track to the true wind, with directly into the wind being 0 degrees and a tailwind 180 degrees) and vessel speed along the track, in knots. The Beaufort number will be at the top of the file, as shown in the sample below:

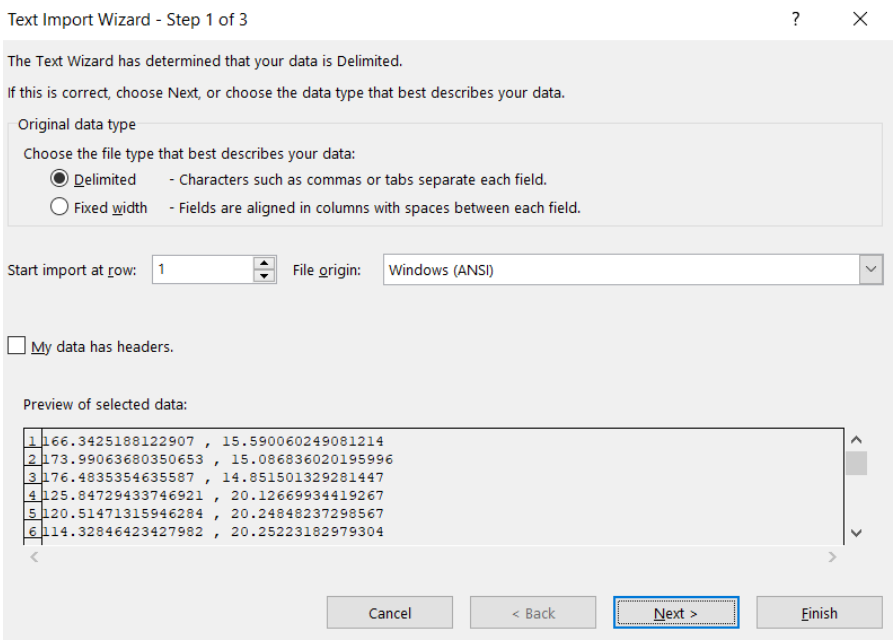
Beaufort7.0

166.3425188122907 , 15.590060249081214

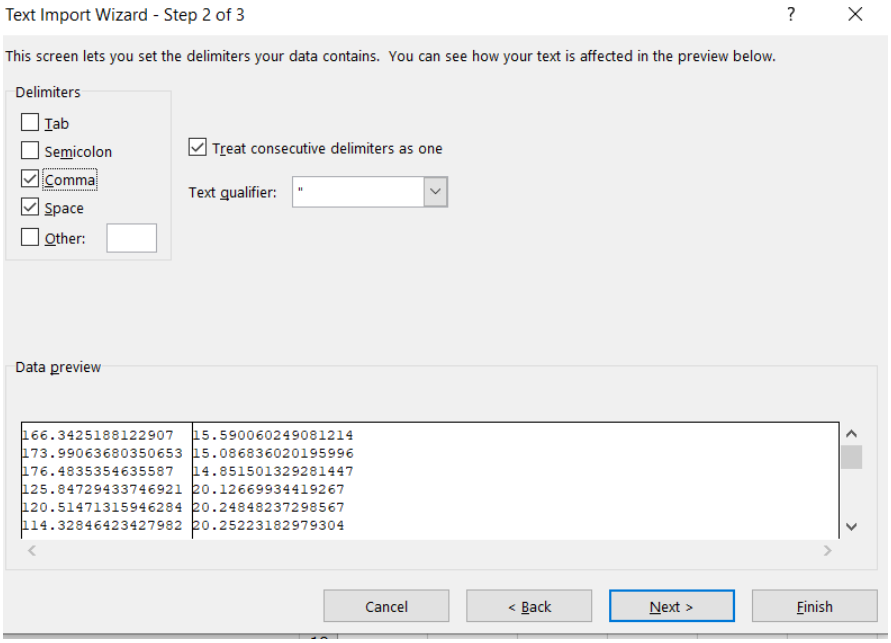
173.99063680350653 , 15.086836020195996

176.4835354635587 , 14.851501329281447

5. To plot, copy the track and direction and open an Excel sheet. Click where you would like the data to be on the Excel sheet, and in upper left hand corner of Excel sheet, select “Paste”. Then “Text Import Wizard”.



Press Next on screen shown above.



Press Finish. The data will appear on the spreadsheet, ready for plotting.

6. The user should run experiments to determine the highest possible time step to use. This is done by trying progressively smaller time steps until there is no longer significant change in the results. Figure 4 below shows the ship speed and track resulting from varying the time step. The results showed that a time step of 0.001 seconds was appropriate.

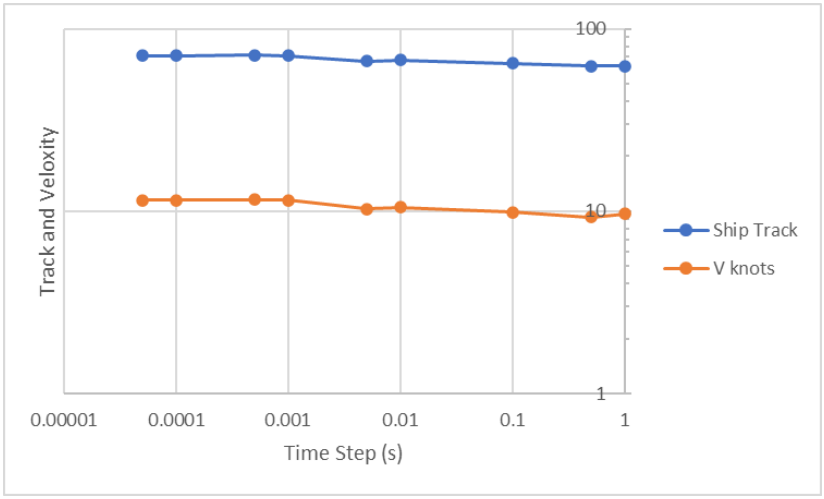


Figure 4. Effect of time step size on solution of ship speed and direction. Below roughly 0.001 s, the speed and direction appear to remain a constant value.

In addition, the user should experiment with the “tol” parameter. For this study involving the Series 60 hull and Dynarig sails, a tol value of 10 worked well. But for one run of 10 degree angle of attack, it became necessary to raise the tolerance to 20 in order to “fill in” the plot, where a relatively wide space

without results had left an unwanted gap. In this case, the new point on the plot which appeared as a result of the looser tolerance was completely consistent with the points ahead and behind it.

7. When making curve-fits to sail and hull data, use the lowest possible degree for polynomial fits. Otherwise, you may have difficulties finding solutions.

OpenVPP Verification

OpenVPP performance was checked against VPP results from [3] (MARAD report). Figure 5 below shows Reference [3] highest possible speeds as a red line at Beaufort 6-7. Points of a multitude of colors represent OpenVPP results at a variety of angles of attack ranging from 0 to 60 degrees at Beaufort 7. As the plot shows, not all angles of attack are optimal, but connecting the markers with highest speed allows comparison with [3]. OpenVPP is in very good agreement with [3] results except in the very low and very high angles into the wind.

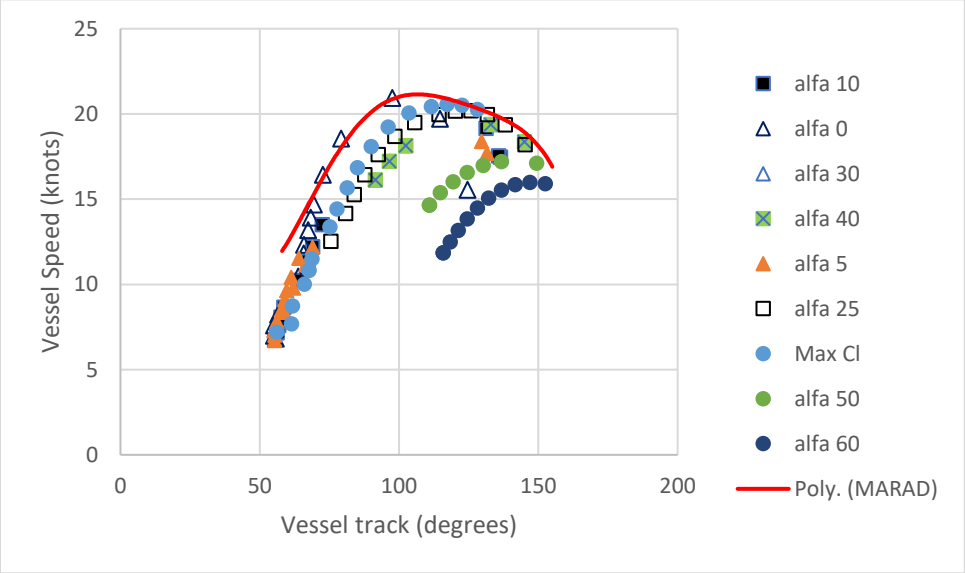


Figure 5. The solid red line shows results from [3]. The triangle, square, circle symbols represent OpenVPP results for a variety of sail angles of attack. OpenVPP and [3] are in very close agreement, except at the extreme ends of the red plot.

It was noticed that the results are very sensitive to relatively small differences in the polynomial curve fits to hull drag and sail lift and drag. Without access to the code used in [3], it is impossible to verify this explanation for the OpenVPP inaccuracy. But since OpenVPP was very accurate throughout most of the tested range, and the general trend of the results was the same for [3] and OpenVPP, it is likely that the differences in curve-fitting explain the discrepancy.

The second source of concern is that the results from [3] shown in Figure 5 are for a range of Beaufort numbers from 6 to 7. The use of a range is not explained in [3], but it points to uncertainties in their results.

References:

[1] Skene's Elements of Yacht Design; Francis Kinney, Dodd, Mead and Company

[2] Approximation of Differential Equations by Numerical Integration; Erik Cheever https://lpsa.swarthmore.edu/NumInt/NumIntFourth.html

[3] Feasibility of sailing ships for the American merchant marine; J. Woodward, R. Beck, B. Scher and C. Cary, Report #168, Feb 1975, University of Michigan Department of Naval Architecture and Marine Engineering, report to the U.S. Maritime Administration

[4] Wind tunnel tests for a six-masted sailing vessel by Prölss: Translation of B. Wagner's "Windkanalversuche für einen sechsmastigen Segler nach Prölss"; L.A. Ribarov, May 2022, Journal of Merchant Ship Wind Energy, www.jmwe.org

[5] Translation of B. Wagner's Angled towing tests for a sailing vessel hull with and without bar keel and for the "Mariner" (Schrägschleppversuche für einen Seglerrumpf mit und ohne Balkenkiel und für den "Mariner"); L.A. Ribarov, September 2022, Journal of Merchant Ship Wind Energy www.jmwe.org

[6] Series 60 methodical experiments with models of single-screw merchant ships; F.H. Todd, 1963, David Taylor Model Basic Report 1712.

APPENDIX

The OpenVPP code:

```
# Published under Creative Commons license CC BY-NC-SA 4.0. Please consult Appendix for more information
# solves for sailing ship velocity and track angle to wind, given drift angle thetas
# USES THE VELOCITIES IN X AND Y TO DETERMINE NEW SHIP TRACK
# USES 4th order Runge Kutta
# ******************************************
# program considers sail forces and hull forces, and uses an explicit (time-marching) scheme to calculate the ship's velocity and angle to wind
# solution given for a range of thetas (drift angle)
import math

# program parameters
thetas = 2.0        #lowest drift angle value wanted minus 1
maxthetas = 22.0    # highest drift angle wanted
tstep = 0.001           #time step in seconds
tol = 10.0              # when summation of forces = 10 lbf, the program stops.. this number can go as high as 100-1000
nloops = 1e6            #max no of time step to do... 1e6 usually good
pi = 3.14159
# ship parameters
conv = 57.296           #radians = degrees/conv
lwl = 565.0             #ft waterline length of 15 kDWT Mariner type hull
wt = 40.0e6             # ship mass in lbm also the weight in lbf
displ = wt/64.0         #ship displacement in ft^3
GM = 6.3                #ship GM ft
lp = 525.0              #length between perpendiculars ft
Awet = 47.3             #ft^2, from Todd report.. S/displ^2/3 = 6.481 for Series 60 hull with CB - 0.6.. 47,397
asails = 88000.0        #sail area in ft^2
rhoair = 0.07           #air density in lbm/ft^3
relax = 1.0             #relaxation parameter angle
relaxv = 1.0            # relaxation parameter velocity

print("input beaufort number ")
beau = float(input())
vw = 0.836*beau**(1.5)*3.28  #calculates wind speed from beaufort and converts to ft/s, eq'n from MARAD 1975 report by Woodward
with open('vppdat','w') as fout:
    fout.write('{}{}\n'.format("Beaufort", beau))
```

```
    while thetas < maxthetas:                    # ************** MAIN LOOP FOR
INCREMENTING DRIFT ANGLE THETAS ******* LOOP 1
        lowcount = 0.0
        thetas = thetas + 1.0   #increase drift angle
        istep = 0               # counts the number of loops for each drift angle
thetas
        jstep = 0               # counts number of runge-kutta loops
        thetasR = thetas/conv
        icountv1 = 0            #these were for debugging.. may need it sometime
        icountv2 = 0
        icountv3 = 0
        icountv4 = 0

        #initialize unknowns
        thetaap = 80.0          # initialize apparent wind angle measured from
ship long line    was 80 and 80
        thetatrack = 80.0       # initialize angle of wind rel to wind
        thetatrackR = thetatrack/conv
        vap = vw                #initialize Apparent velocity of wind, relative
to ship long plane
        vs = 10.0               #initialize true speed of ship along true
path     was 10
        vsy = 8.0
        forcex = 10000.0        #initalize sum of forces in x

        #SAIL ANGLES AND FORCES

        while (abs(forcex)>tol):        # program stops when sum of forces in x
is 10 lbf or less
            if (istep>nloops):          # stop looking after max no of loops is
surpassed
                break
            for jstep in range(1,5):    # Runge- Kutta loop goes from 1 to 4  as
configured in range
                if (vap <= 0):
                    vap = 0.1
                if (vs <= 0):
                    vs = 0.1

                alfa = 180.0 - thetatrack   #this alfa is not the angle of
attack, but a wind triangle angle
                alfar = alfa/conv
                vap = (vs**2+vw**2-2*vs*vw*math.cos(alfar))**0.5
                part = (vw**2-vap**2-vs**2)/(-2*vap*vs)
```

```
                if(part<-1 ):
                    part = -0.99
                if(part>1):
                    part = 0.99
                betar = math.acos(part)
                beta = betar*conv
                thetaap = beta-thetas
                delta = 90.0 - thetaap-thetas
                deltaR = delta/conv
                thetaapR = thetaap/conv

# Below are sail lift and drag coefficients at a variety of angles of attack, as
function of thetaap (apparent vessel angle to wind)
# For other angles of attack, make another run
# lowtheta and hitheta give the range of thetaap over which the curve fit is
applicable

# ***  cl and cd for the max cl possible and the accompanying Cd (uses alfa that
gives highest Cl)
#                lowtheta = 25.0
#                hitheta = 160.0
#                alfax = -100  # the -100 denotes max cl
 ##                #cl = -2.5881e-8*thetaap**4 + 1.01375E-5*thetaap**3 - 1.5664E-
3*thetaap**2 + 1.0528e-1*thetaap - 1.0734 #from wagner for Dynarig, thetaap in
degrees
#                cl = 9.70405269E-12*thetaap**6 - 5.35131215E-09*thetaap**5 +
1.13774423E-06*thetaap**4 - 1.16430730E-04*thetaap**3 + 5.62032981E-03*thetaap**2
- 9.44965229E-02*thetaap + 1.02526822
#                cd = -1.13619194E-09*thetaap**5 + 4.97084789E-07*thetaap**4 -
7.94712981E-05*thetaap**3 + 5.53230954E-03*thetaap**2 - 1.49676800E-01*thetaap +
1.70649230
##                 cd = 5.3718e-8*thetaap**4 - 1.5977E-5*thetaap**3 + 1.4348E-
3*thetaap**2 -3.1846e-2*thetaap + 0.4887

# alfa = 0  fit good between 15 and 90 theta ap.. don't use it outside this range
                lowtheta = 15.0
                hitheta = 90.0
                alfax = 0.0
                cl =  2.21916E-06* thetaap**3 - 8.08737E-04*thetaap**2 +
6.79214E-02* thetaap - 5.40917E-01
                cd = 1.056154E-07* thetaap**4 - 2.185241E-05 * thetaap**3 +
1.320190E-03 * thetaap**2 - 1.758923E-02* thetaap + 1.602717E-01

# alfa = 5  fit good from 20 to 95 degrees theta ap
```

```
#                lowtheta = 20.0
#                hitheta = 95.0
#                alfax = 5.0
#                cl = 2.632904E-06* thetaap**3 - 8.754878E-04* thetaap**2 +
7.520260E-02* thetaap - 6.738967E-01
#                cd = 9.141549E-08* thetaap**4 - 2.042657E-05* thetaap**3 +
1.306624E-03* thetaap**2 - 1.597901E-02* thetaap + 1.286027E-01

# alfa = 10   curve fit range 25 - 100 degrees theaap
#                lowtheta = 25.0
#                hitheta = 100.0
#                alfax = 10.0
#                cl = 2.370286E-06* thetaap**3 - 8.721324E-04* thetaap**2 +
8.053825E-02* thetaap - 8.117323E-01
#                cd = 8.205687E-08* thetaap**4 - 1.928333E-05* thetaap**3 +
1.256630E-03* thetaap**2 - 1.094621E-02* thetaap + 2.100036E-02

# alfa = 15   fit good 30 - 105 degrees thetaap
#                lowtheta = 30.0
#                hitheta = 105.0
#                alfax = 15.0
#                cl = -1.280833E-08* thetaap**4 + 7.430414E-06* thetaap**3 -
1.464184E-03* thetaap**2 + 1.093319E-01* thetaap - 1.277095
#                cd = 9.641223E-08* thetaap**4 - 2.446442E-05* thetaap**3 +
1.833831E-03* thetaap**2 - 3.248504E-02* thetaap + 3.027453E-01
# alfa = 20 fit good from 35-100 degrees thetaap
#                lowtheta = 35.0
#                hitheta = 100.0
#                alfax = 20.0
#                cl = 4.511625E-06* thetaap**3 - 1.262737E-03* thetaap**2 +
1.078671E-01* thetaap - 1.404577
                ##cd = 6.193627E-08* thetaap**4 - 1.727462E-05* thetaap**3 +
1.337412E-03* thetaap**2 - 1.592567E-02* thetaap + 1.038270E-01
#                cd = -3.801122E-04*thetaap**2 + 5.576407E-02*thetaap -
9.517079E-01

# alfa = 25 fit good from 40-115 degrees
#                lowtheta = 40.0
#                hitheta = 115.0
#                alfax = 25.0
#                cl = 2.437690E-09* thetaap**5 - 9.245469E-07* thetaap**4 +
1.385103E-04* thetaap**3 - 1.046498E-02* thetaap**2 + 4.071641E-01* thetaap -
5.189448
```

```
#                cd = 3.138163E-07* thetaap**3 - 4.360957E-04* thetaap**2 +
6.210647E-02* thetaap - 1.115316

# alfa = 30  fit good from 45 - 120 degree thetaap
#                lowtheta = 45.0
#                hitheta = 120.0
#                alfax = 30.0
#                cl = 2.677905E-06* thetaap**3 - 7.642728E-04* thetaap**2 +
6.844733E-02* thetaap - 6.486807E-01
#                cd = -3.336362E-04* thetaap**2 + 5.564676E-02* thetaap -
9.922022E-01

#alfa = 40 fit good from 55 - 130 degrees thetaap
#                lowtheta = 55.0
#                hitheta = 130.0
#                alfax = 40.0
##                # cl = 1.505830E-09* thetaap**5 - 7.351991E-07* thetaap**4 +
1.394428E-04* thetaap**3 - 1.281202E-02* thetaap**2 + 5.698976E-01* thetaap -
8.923271
#                cl= 6.620553E-04*thetaap + 8.556028E-01
##                # cd = 1.750167E-09* thetaap**5 - 1.001627E-06* thetaap**4 +
2.145473E-04* thetaap**3 - 2.190977E-02* thetaap**2 + 1.082781* thetaap -
1.971582*10
#                cd = -3.303898E-06*thetaap**3 + 6.987115E-04*thetaap**2 -
3.910892E-02*thetaap + 1.602722E+00

# alfa = 50 fit good from 70 - 140
#                lowtheta = 70.0
#                hitheta = 140.0
#                alfax = 50.0
##                 cl = -5.492424E-08* thetaap**4 + 2.190657E-05* thetaap**3 -
3.139962E-03* thetaap**2 + 1.906383E-01* thetaap - 3.457652
#                cl = 0.75
##                 cd = -8.143939E-08* thetaap**4  + 3.188131E-05* thetaap**3  -
4.702462E-03* thetaap**2 + 3.138443E-01* thetaap - 6.757348
#                cd = -2.323232E-06*thetaap**3 + 5.806277E-04*thetaap**2 -
4.139430E-02*thetaap + 2.008788E+00

#  alfa = 60 fit good from 80 - 150
#                lowtheta = 80.0
#                hitheta = 150.0
#                alfax = 60.0
##                 cl = -1.060606E-06* thetaap**3 + 4.278139E-04* thetaap**2 -
5.614610E-02* thetaap + 2.807013
```

```
#                cl = -8.095238E-04*thetaap + 5.205952E-01
#                cd = -1.791667E-04*thetaap**2 + 3.817262E-02*thetaap -
6.750595E-01

##                cd = -9.943182E-08* thetaap**4 + 4.323864E-05* thetaap**3 -
7.079451E-03* thetaap**2 + 5.169510E-01* thetaap - 1.286665*10

# alfa = 70  fit good from 90 - 160 these probably need re-doing as well as 80=90
below
#               lowtheta = 90.0
#                hitheta = 160.0
#               alfax = 70.0
#               cl = -1.275253E-06* thetaap**3 + 5.380411E-04* thetaap**2 -
7.476154E-02* thetaap + 3.598247
#               cd = -1.609848E-07* thetaap**4 + 7.912879E-05* thetaap**3 -
1.455133E-02* thetaap**2 + 1.179435* thetaap - 3.410521*10

# alfa = 80  fit from 100 t0 170
#               lowtheta = 100.0
#               hitheta = 170.0
#               alfax = 80.0
#               cl = -2.214646E-06* thetaap**3 + 9.502056E-04* thetaap**2 -
1.337105E-01* thetaap + 6.158182
#               cd = -1.396780E-07* thetaap**4 + 7.240846E-05* thetaap**3 -
1.409834E-02* thetaap**2 + 1.211523* thetaap - 3.731189*10


# alfa = 90  fit from 110 to 180 / cd fit is somewhat inaccurate at very end..
from thetaap 170-180 use 0.23 for cd
#               lowtheta = 110.0
#               hitheta = 180.0
#               alfax = 90.0
#               lowtheta = 178.0
#               hitheta = 180.0
#               cl = 1.022727E-07* thetaap**4 - 6.073232E-05* thetaap**3 +
1.343371E-02* thetaap**2 - 1.309370* thetaap + 4.721426*10
#               cd = 1.969697E-07* thetaap**4 - 1.108586E-04* thetaap**3 +
2.301439E-02* thetaap**2 - 2.103060* thetaap + 7.285584*10




                # Resolve sail forces in direction of vessel track
```

```
                lap = 0.5*rhoair/32.2*vap**2*cl*asails    #lift from sails perp
to thetaap
                dap = 0.5*rhoair/32.2*vap**2*cd*asails    #drag from sails along
thetaap ... used to be ds

                lsx = lap*math.cos(deltaR)
                dsx = dap*math.cos(thetaapR+ thetasR)
                lsy = lap*math.sin(deltaR)
                dsy = dap*math.sin(thetaapR + thetasR)

#
#HULL FORCES in direction of ship track
#

                clhull = (1.9735e-4*thetas**2+ 3.6154e-3*thetas)       #from
wagner angle towing thetas in degrees.. use only between 0 and 20 degrees
                cdhullinduced = (1.5219e-4*thetas**2- 4.5123e-4*thetas)
                draghullinduced = 0.5*64/32.2*vs**2*cdhullinduced*displ**0.6667

                corr = 1+(vap/100.0)**2*750.0/lwl*math.exp(1-
(thetaap/40.0)**4)  #correction factor for effect of waves from Woodward 1975

                # folllowing is from series 60 report by Todd 1963
                draghulltow =  (4.511886E-01*vs**4 - 2.609315E+01*vs**3 +
5.835723E+02*vs**2 - 2.420343E+03*vs)*corr*1.1
                dhx =draghullinduced + draghulltow
                lhy = 0.5*64/32.2*vs**2*displ**0.6667*clhull

#'*****************************************
#'SOLVE FOR NEW VELOCITIES in x and y directions
#' *****************************************

                forcex = lsx-dsx-dhx
                rungslopex = forcex/(wt/32.2)

                forcey = lhy-dsy-lsy
                rungslopey = forcey/(wt/32.2)


                if (abs(forcex)< tol and jstep == 1):   # converged, on to next
iloop (new thetas)
```

```
                    istep = nloops
                    break

    #  goto 12
    # end if

                if (jstep == 1):     # find velocities to calculate the slope at
t/2
                    if (istep ==nloops*0.2 or istep == nloops* 0.4 or istep ==
nloops* 0.6 or istep == nloops* 0.8 or istep == nloops*0.9):
                        print ("working.. forcex ")
                        print(forcex)
                        print("vsknots")
                        print(vs*0.59)
                        print("thetatrack")
                        print(thetatrack)
                        print("istep")
                        print(istep)
                        print("rungslopex")
                        print(rungslopex)
                        print(" ")

                    k1 = rungslopex
                    vsold = vs
                    vsoldy = vsy
                    thetatrackold = thetatrack
                    thetatrackRold = thetatrackR
                    vs = vs + rungslopex*tstep/2.0   #'velocity in x at mid step
                    vsy = vsy + rungslopey*tstep/2.0 #'vel in y at mid time step

                if (jstep == 2):
                    k2 = rungslopex
                    vs= vsold
                    vsy = vsoldy
                    thetatrack = thetatrackold
                    thetatrackR = thetatrackRold
                    vs = vs + rungslopex*tstep/2.0   #velocity at mid-step, using
the new slope from jstep = 1
                    vsy = vsy+ rungslopey*tstep/2.0

                if (jstep == 3): # we now go all the way across the time step

                    k3 = rungslopex
                    vs= vsold
```

```
                    vsy = vsoldy
                    thetatrack = thetatrackold
                    thetatrackR = thetatrackRold
                    vs = vs + rungslopex*tstep
                    vsy = vsy+ rungslopey*tstep

                if (jstep == 4):  #from jstep 3, new runglsope values were
found... k4 = runglsopex
                    k4 = rungslopex
                    kmean = (k1/6.0+k2/3.0 + k3/3.0+k4/6.0)
                    vs= vsold
                    vsy = vsoldy
                    thetatrack = thetatrackold
                    thetatrackR = thetatrackRold
                    vs = vs + kmean*tstep   #velocity at mid-step, using the new
slope, from jstep = 1
                    done = abs(kmean*tstep)
                    vsy = vsy+ rungslopey*tstep




#'*************************************************************************
****
#'calculate new trajectory, by adding or subtracting arctan(vy/vx) to previous
angle
#'arctan function cannot be negative. As vy/vx goes to infinity, arctan function
goes
#'90 degrees. As vy/vx goes to zero arctan function goes to zero
#'*************************************************************************
****
                if (vsy < 0 and vs > 0):

                    icountv1 = icountv1 + 1
                    changeslope =  math.atan(abs(vsy/vs))
                    thetatrackR = changeslope+ thetatrackR
                    if(thetatrackR < 0.96):
                        thetatrackR = 0.96 # the limit is set at 55 degrees into
the wind
                    thetatrack = thetatrackR*conv

                if (vsy < 0 and vs < 0):    #this will never occur but giving in
to paranoia. Vs never negative
```

```
                    icountv2 = icountv2 + 1
                    changeslope = math.atan(abs(vsy/vs))
                    thetatrackR = pi - (thetatrackR - changeslope)
                    if(thetatrackR < 0.96):
                        thetatrackR = 0.96 #' the limit is set at 55 degrees into
the wind
                    thetatrack = thetatrackR*conv

                if (vsy > 0 and vs > 0):
                    icountv3 = icountv3 + 1
                    changeslope = math.atan(abs(vsy/vs))
                    thetatrackR = (changeslope)*-1+ thetatrackR
                    if (thetatrackR < 0.96):
                        thetatrackR = 0.96 # the limit is set at 55 degrees into
the wind
                    thetatrack = thetatrackR*conv

                if (vsy > 0 and vs < 0):   #this will never occur either. Vs will
never be negative
                    icountv4 = icountv4+ 1
                    changeslope = math.atan(abs(vsy/vs))
                    thetatrackR = (changeslope)*-1+ thetatrackR
                    if (thetatrackR < 0.96):
                        thetatrackR = 0.96
                    thetatrack = thetatrackR*conv


#' account for possibility that new thetatrack may be less than zero or greater
than 180 degrees.. the limits of the simulation field
#' if theatrack is negative (zero is directly into the wind), then abosulte value
is taken, as this gives the same angle into the wind
#' if thetatrack is > 180, then the number of degrees angle goes beyond 180 is
subtracted from 180, giving same angle to wind.
                if (thetatrack < 0):
                    thetatrack = (90.0 - abs(thetatrack))
                thetatrackR = thetatrack/conv

                if (thetatrack > 180):
                    thetatrack = 360.0 - abs(thetatrack)
                thetatrackR = thetatrack/conv

            # bottom of jstep loop    done with r-k
loop    ********************************************************** END LOOP 3
            istep= istep+1
```

```
        # bottom of done loop .. if we make it here, a solution was found or
exceeded nloops (max no of loops)  ******* END LOOP 2
        vwindknots=vw*0.59
        vsknots = vs*0.59  #vessel speed along track in knots

        tolchecker = rungslopex*tstep
        print("tolerance checker value rungslope*tstep should be less than 1e-12,
thetas 33333333      333333  ")
        print(tolchecker, thetas)
        if(abs(forcex)< tol and jstep ==4):
            print("partly in heaven     x x x x Thetaap  lowtheta   hitheta")
            print(thetaap, lowtheta, hitheta)
            print("no of loops")
            print(istep)
            if (thetaap <= hitheta and thetaap >= lowtheta):  # check thetaap
within curve fit limit for sails
                fout.write('{}{}{}\n'.format(thetatrack, " , ",vsknots))
                print("---------------------")
                print("converged ... thetas , vsknots, thetatrack, thetaap, 0000
vs,vw, vap ")
                print(thetas)
                print(vsknots)
                print(thetatrack)
                print(thetaap)
                print("0000")
                print(vs,vw, vap)
                print("draghullinduced, draghulltow, CORR")
                print(draghullinduced, draghulltow, corr)
                print("--- icount v1-4 ---")
                print(icountv1)
                print(icountv2)
                print(icountv3)
                print(icountv4)
                print(" degrees of roll ")
                thetan = conv* ( math.asin((math.cos(thetaapR)*lap +
math.sin(thetaapR)*dap)*(96.0+35.0/2.0)/GM/wt))
                print(thetan)
                print(" ")
            # end of if
        else:
            print("***** failed to converge or outside fit lmits... thetas,
forcex")
            print(thetas)
            print(forcex)
```

```
            print("")

        #
*************************************************************************************
**************************  END LOOP 1
    #
fout.close()
```

LICENSE INFORMATION: