Optimizing Sailing Trajectories

Hemanth Sarabu

ME 6103

Problem Statement:

The project focuses on developing two algorithms that address the issue of optimizing a sailboat's trajectory when a starting point and destination are given alongside static wind conditions. The underlying physics that govern the optimal path of a sailboat for a given set of conditions are highly coupled and dynamic, rendering the course very unintuitive to determine. Algorithms that are able to produce the path plan that takes the minimum amount of time to complete the course can be very helpful and the resulting work is presented in this report.

Models

Both algorithms developed in this project use the idea of calculating the Velocity Made Good (VMG) as a parameter relating the state of the sailboat at any given time to the time it would take to complete a given course.

Before going into details about how the models work, some basic sailing theory is introduced.

Velocity Made Good

It is not possible for boats to sail directly into the wind, requiring the course of the boat to alternate between headings. This process is called "tacking" and is used commonly by sailors to make their way to a mark that is upwind. On a tack, the sailor will generally point the sailboat as close to the wind as possible while still keeping the winds blowing through the sails in a manner that provides aerodynamic lift to propel the boat.

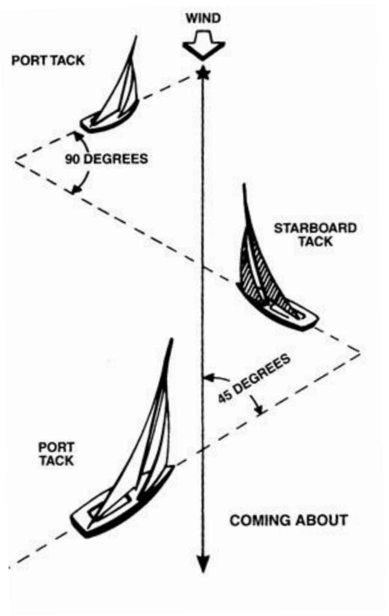


Figure 1: Upwind Tack

Then the boat is turned away from the wind in slight increments in order to generate more forward lift on the sails allowing it to move with greater speed, but less directly toward the destination. This may be seen in the schematic diagram depicted in Figure 1.

The range of heading that does not produce any significant lift is called the no-go zone. It is indicated in red in Figure 2.

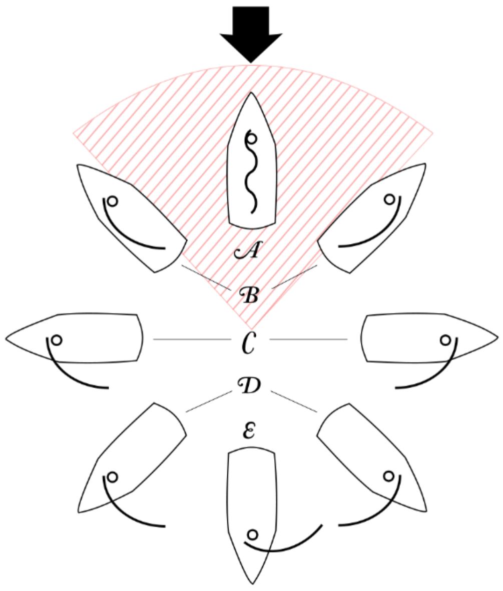


Figure 2: Points of sail – Red indicates no-go zone

The VMG can be calculated by using the following expression where V_{true} is the velocity of the sailboat with respect to stationary ground and θ is the angle between current heading and the direction to destination.

$$VMG = Vtrue * \ cos\theta$$

However, the relationship between the true velocity of the sailboat and the true wind velocity depends on what assumptions are made. Some instances take into consideration the fact that a sailboat's Speed Over Ground increases or decreases relative to the wind direction. In theory, a sailboat's speed increases while sailing from upwind to a downwind direction. Other parameters to factor in include the sail boat's specifics that depend on the make and design of the boat. These are the 'Velocity Increase Constant', 'No – Go Zone' and 'Degree Interval', which are normally provided by the manufacturer. The physical meaning of this constant expresses 'the sail boat increases speed by 10% for every X degrees from the wind'. Considering these factors, a True VMG can be calculated using the equations:

For Upwind:

$$VMG_D, \hat{y} = \frac{V_w}{cos\theta_s}(1 + \frac{\beta}{100})^{\frac{|\theta_o - \theta_s|}{i}} cos\theta_\gamma$$

For Downwind:

$$VMG_D, \hat{y} = \frac{V_w}{cos\theta_s}(1 + \frac{\beta}{100})^{\frac{|180^\circ - \theta_s - \theta_o|}{i}} cos\theta_\gamma$$

$Where,$

$VMG = Velocity\ Made\ Good\ towards\ destination$

$V_w = Velocity\ of\ Wind$

$\theta_s = Angle\ between\ heading\ and\ destination$

$\theta_0 = No - go\ zone$

$\theta_y = Angle\ between\ wind\ direction\ and\ heading$

$\beta = Velocity\ Increase\ Constant$

$i =\ Degree\ Interval$

In the expressions shown above the no-go zone (θo) , Degree Interval (i) and Velocity Increase Constant (β) are usually provided by the manufacturer. As these are specific to the boat's design, commonly quoted values are used throughout this project.

Shown below is a plot of the resulting VMG when θs (theta_rd) is varied for the case of sailing upwind (in blue) and for the case of sailing downwind (in yellow). The relative angle to the wind is kept constant at 40° or 0.698 rad.

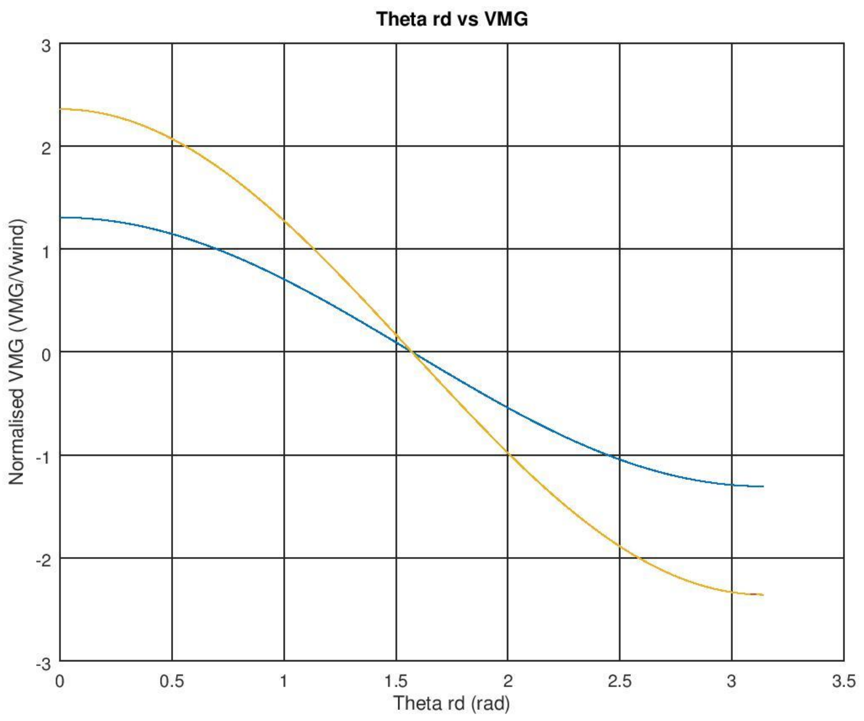


Figure 3: VMG vs Angle relative to destination

The plot indicates that a higher VMG is achieved in the downwind case than in the upwind case as expected from the model.

In order to model the no-go zone, the VMG calculating function incorporates an if statement that checks and assigns a value of zero if the boat's heading is in this region.

Figure 4 shows the relationship between the VMG and relative wind angle (theta rw) keeping bearing to destination a constant. It is important to note that in practice if the destination is kept at a constant location along with wind direction, changing the bearing of the boat changes both relative angles (with wind and destination). However, the plot shown below does not have a varying relative angle to destination implying the wind direction is only changing whereas the bearing of the sailboat and the location of the destination are kept constant.

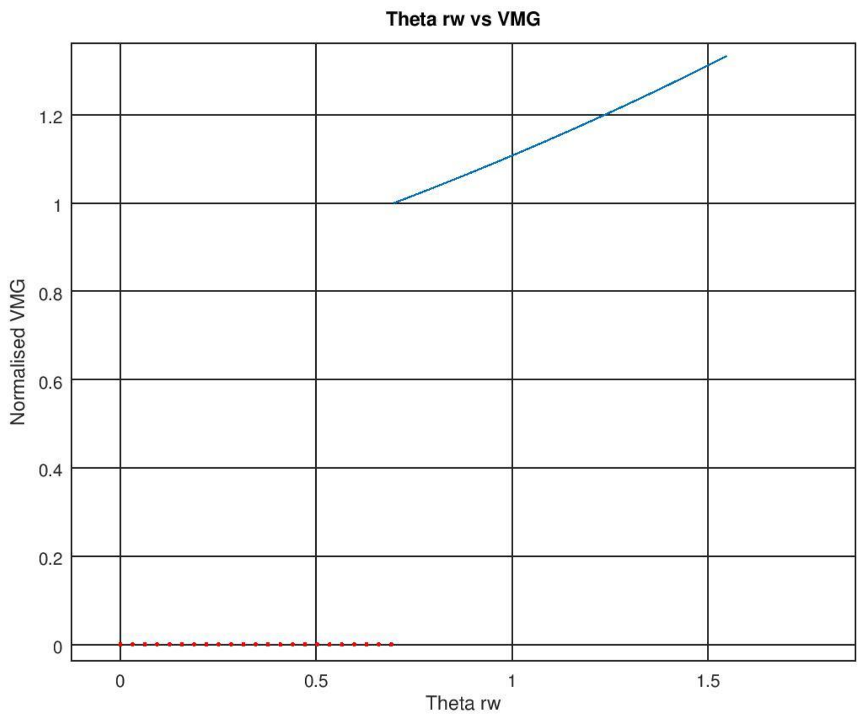


Figure 4: VMG vs Relative Wind Direction

Assumptions

In order to simplify the optimization problem, the following assumptions are made and incorporated into the models:

1. True velocity of boat at any point of time is a function of the velocity of wind and heading relative to wind direction only. This implies that at any point these are the only two variables required to calculate how fast the boat will be moving with respect to ground (true velocity) in the direction it is moving.
2. Effects of momentum are not considered. As the expressions shown earlier allow for the calculation of velocity instantaneously, the time-dynamic effects are not modelled. This means that acceleration and deceleration are not taken into account and if the boat moves from a current position to the next position, the momentum is not carried or lost but the position's assigned momentum is taken up.
3. Effects of drag are not considered. The effects of drag considered are only those specified by the velocity increase constant and the no-go zone, both provided by the manufacturer. The effects of drag due to the wind and the water due to the form of the boat are not taken into

consideration. In real life however, this phenomenon will have significant effects limiting the maximum velocity of the sailboat.

4. Effects of manoeuvres on the momentum of the sailboat are not considered. For example, tacking causes the sailboat to lose momentum due to the associated drag and loss of lift during the procedure. The loss of momentum is a transient process and is not modelled for this project.

Decision Variables

The objective of the algorithms is to produce paths that take the least amount of time to complete a given course. In order to describe the path taken by the sailboats, the variables required are the x and y coordinates of the sailboat at a given time. Instead of determining the coordinates for each time step, they are determined in consecutive steps- this means the coordinates X^{k+1} are determined relative to X^k in distance and direction as opposed to determining X^{k+1} as a function of time step t^{k+1}.

The problem of producing a locus of X,Y coordinates that are successive in nature can be transformed into producing a direction vector and a distance vector from each of the points to the next point. An example of this is- if X^k is known, in order to determine the next position X^{k+1} a direction vector and a step size are sufficient and required with relation to X^k. Hence the modelling problem is reduced to finding the optimal sequence of direction vectors (heading) if the step size is set to be constant.

In other standard optimization problems tackled during the course, the design space did not include a time dimension. However, this problem indeed has a time dimension and the variable (heading) will have to be determined as the time marches forward (in distance steps and not time steps).

Objective

The objective of the algorithm is to find the path of least resistance, or the path that takes the least amount of time to complete a given course. The time taken is calculated in a discretized manner: The velocity of the sailboat is found for each step and is used to divide the step size.

$$time\ step^k = \frac{step\ size}{velocity\ in\ direction\ of\ step}$$

The time steps are added for the duration of the course in order to register the total time taken.

$$time\ elapsed = \Sigma\ time\ step^k$$

The objective can then be expressed as:

Minimize: time elapsed

The objective is to optimize the path to achieve the lowest time elapsed value for any given course. In order to study the behaviour of the model and the algorithms, 7 cases are used to investigate.

In addition to reducing this parameter, one of the models is developed to first prioritize ease of navigation and then minimize the time elapsed parameter. The difference in performances of both models is investigated to understand the trade-off. The ease of navigation cannot be quantified; hence both the models are inherently different.

Case 1 – Upwind

Sailing prowess is often associated with how well a sailor is able to navigate their way to a mark directly in the upwind direction dead against the wind. The first case is used to test this very performance metric of the algorithms used. It is important to remember that it is not possible to sail directly into the wind (no-go zone). The starting point is at 0,0 and marked in green with the destination marked red.

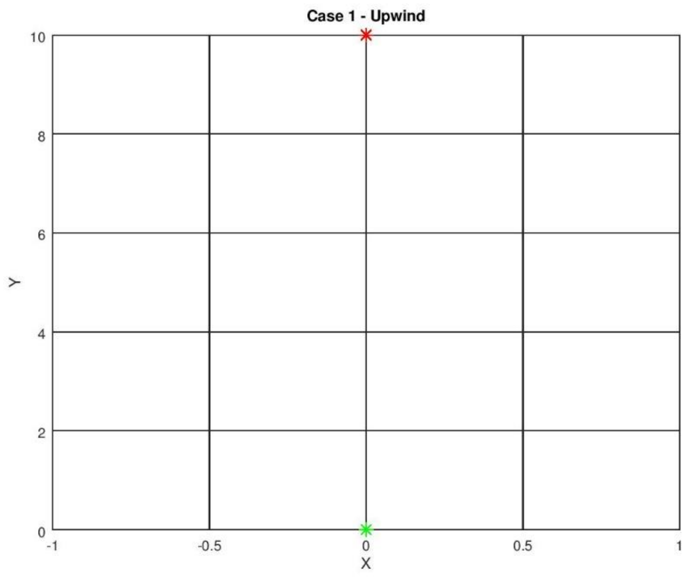


Figure 5: Test Case 1

Case 2 – 45deg Upwind

The second case tests the algorithms behaviour subjected to a destination that is upwind 45 degrees into the wind. This is right outside the no-go zone and the optimal path is fairly predictable.

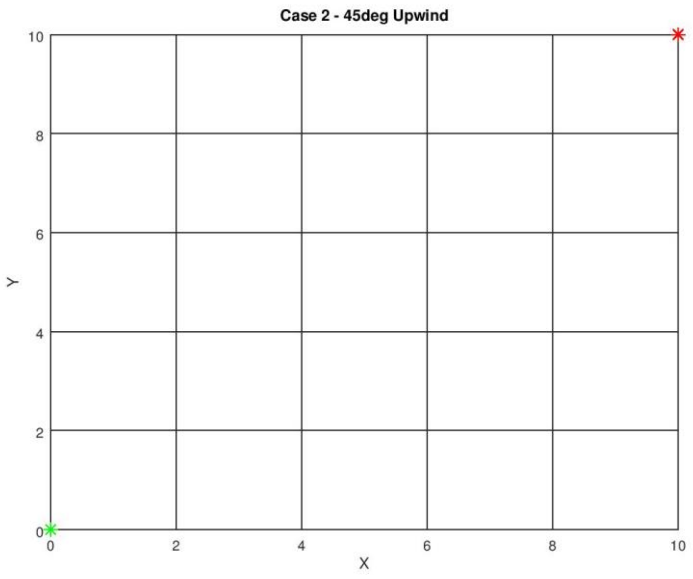


Figure 6: Test Case 2

Case 3 -10deg Upwind

Case 3 tests the algorithms' ability to produce the optimal path when destination is upwind and in the deadzone. Unlike the earlier cases, predicting the optimal path without computation, using pure intuition can be extremely difficult.

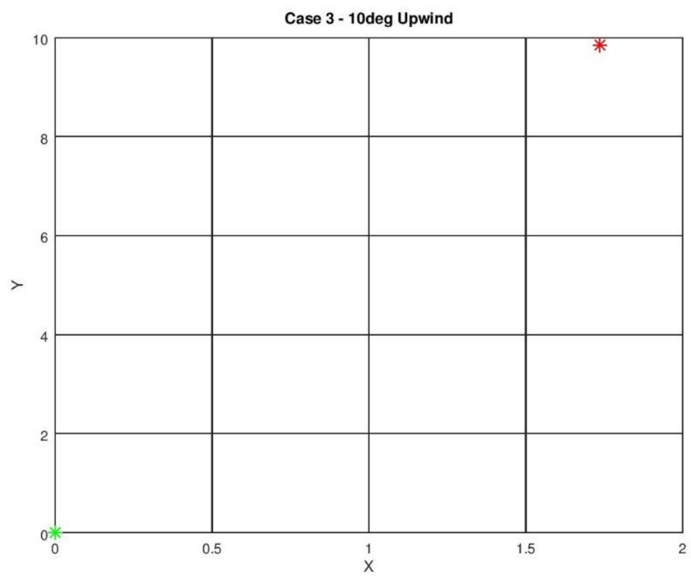


Figure 7: Test Case 3

Case 4 - Downwind

Case 4 requires the sailboat to navigate downwind. The optimal path for this scenario is fairly simply to predict, however this test case is included for completeness and to see if the algorithms produce any interesting, unexpected results.

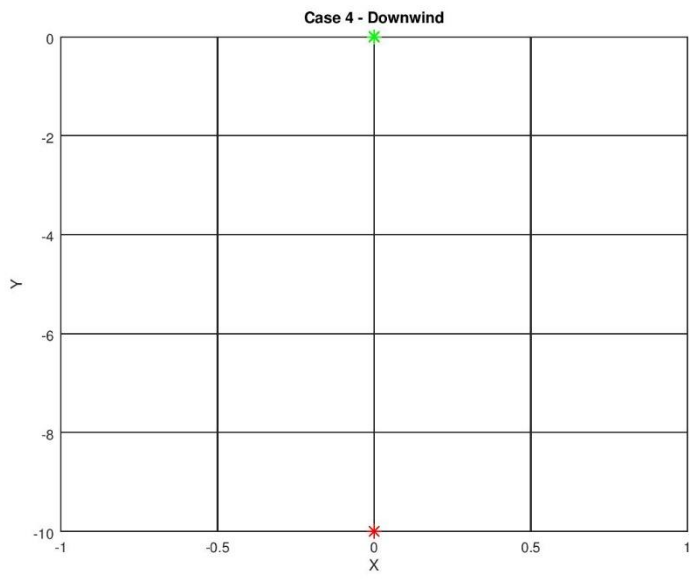


Figure 8: Test Case 4

Case 5 – 45deg Downwind

Case 5 involves sailing from origin 0,0 to the destination which is located downwind at a 45 degree angle.

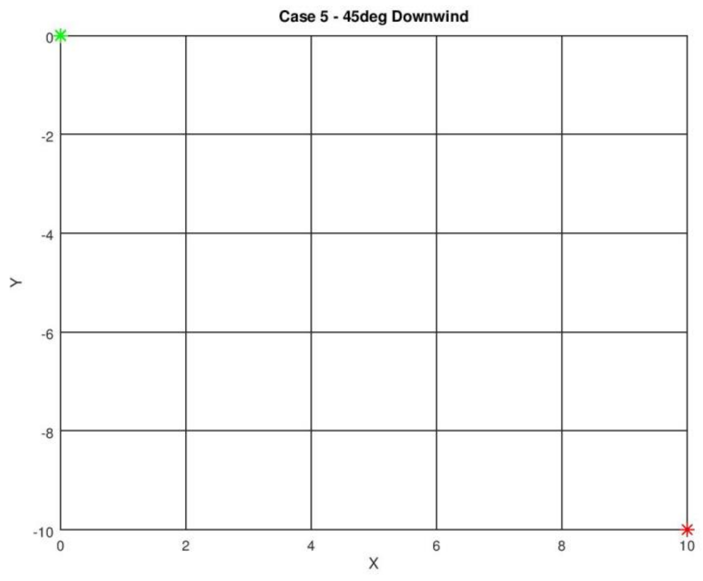


Figure 9: Test Case 5

Case 6 – 90deg

Case 6 involves sailing from origin 0,0 to the destination located downwind perpendicular to the direction of the wind.

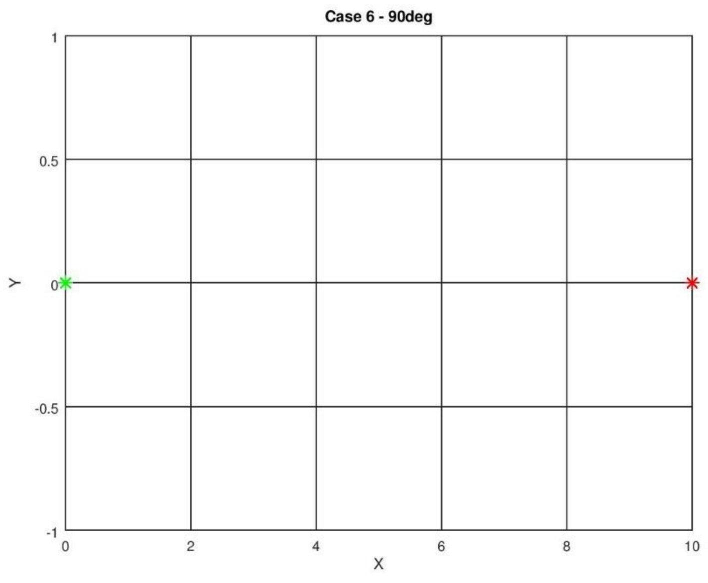


Figure 10: Test Case 6

Case 7 – Race Course

The final scenario is test case 7. Here the algorithms are required to start at the origin (0,0) and touch each of the marks in an anticlockwise direction. The marks are colored red and the path marked in blue is only displayed to indicate the shape of the track. The boat is not required to follow the blue path. The course incorporates elements of the cases seen previously: upwind into the no-go zone, downwind at extreme angles to wind direction.

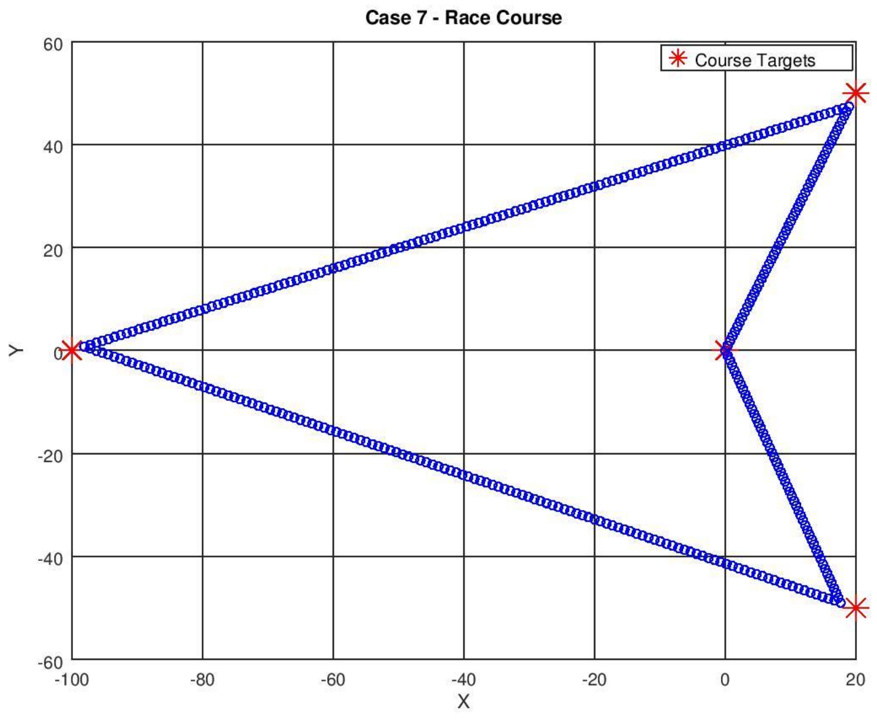


Figure 11: Test Case 7

Constraints

This path optimization problem only has one constraint – at no instance should the boat bear a heading into the no-go zone. In real life however, during a tack, the boat indeed faces the no-go zone briefly before regaining lift in the sails. It is its momentum that allows the boat to steer away from the no-go zone and into a heading that permits forward motion. In the models presented in the report, it is important to remember that the boat's momentum terms are not incorporated. This means that if at any point the algorithm produces a path that involves a heading into the no-go zone, it loses its velocity and cannot make further steps. The algorithms get past this flaw by limiting the heading from entering the no-go zone at all times.

This constraint is expressed as:

When travelling upwind,

$$\theta w > \theta o$$

Where,

$\theta o = no - go\ zone$

$\theta w = heading\ relative\ to\ wind\ direction$

Model Description

Two inherently different models are used to approach the problem. As no suitable and freely available algorithms were found on the public domain, both models have been developed and coded from the ground up using first principles of sailing and optimization.

Model 1 – Single Tack Method

The Single Tack Method (STM) is developed so that the path produced by the model is extremely easy to navigate. This is achieved by implicitly allowing the algorithm to produce a path between two points that incorporates a maximum of one tack (i.e. two leg journey). The algorithm is then required to produce the tack length and angle that results in the minimum time elapsed value. It is important to note that the manner in which ease of navigation is incorporated into this model is by limiting the number of tacks. By increasing the number of tacks, the solution process can get very complex.

To explain this, a scenario is discussed – If arbitrary location A is considered the starting point and a destination is set at B, to produce a path that can only have a single tack, the problem is reduced to finding a point Ts (tack point) such that when the boat travels from A to Ts, Ts to B, it takes less amount of time than if the boat sailed from A to T, T to B. Here T is an arbitrary tack location.

If two tacks were allowed, this would mean that the algorithm would have to find the best combination of two points that result in the smallest time elapsed value. Due to the highly coupled nature of the problem, navigating through two dimensions to find the best combination would be very difficult, and hence is out of the scope of this project.

Algorithm Architecture

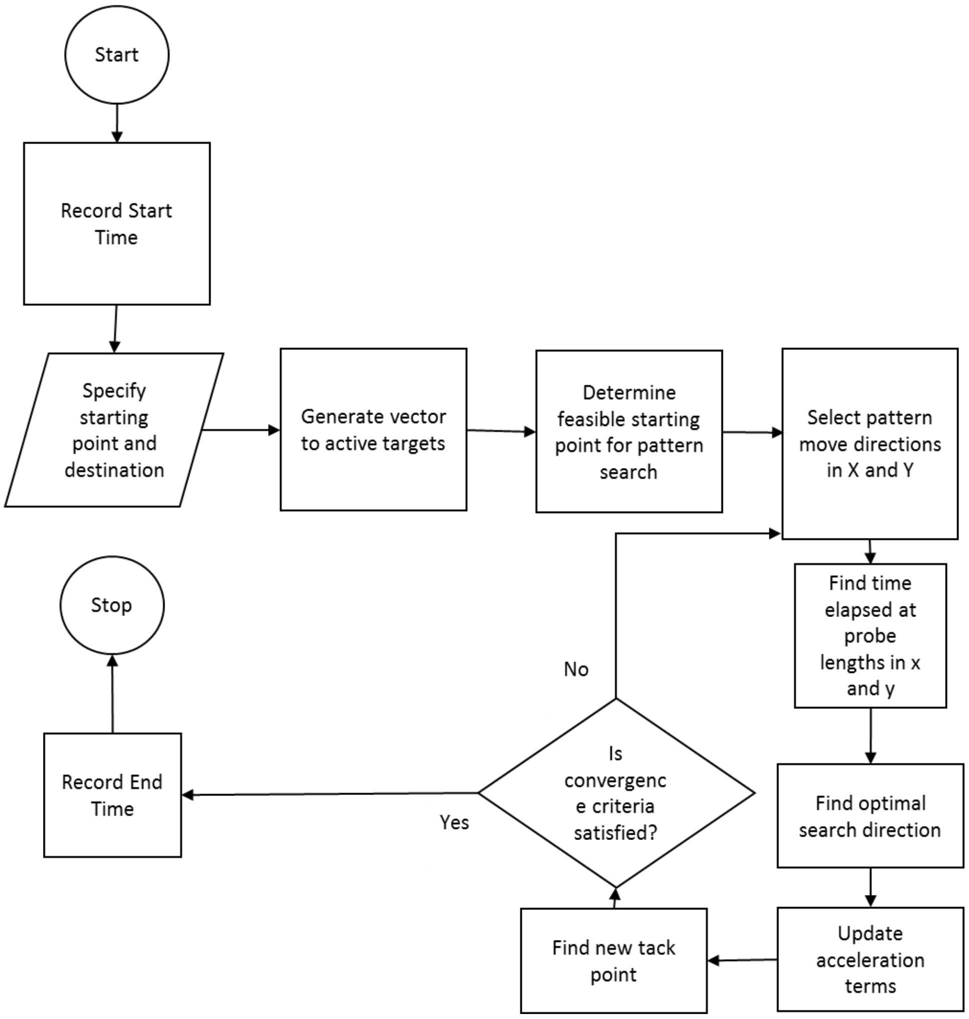
A brief description of the algorithm is discussed in this section. This model requires three routines to arrive at the solution. The first routine required is called *st_2.m* which is provided in Appendix A. The starting point and the current destination is specified in this routine. The programme then calculates an initial feasible point for the tack location. The model employs a pattern search algorithm incorporating an accelerating/decelerating step size. The pattern search algorithm requires to start in the feasible region because starting in the no-go zone results in an infinite path time value. Once, the pattern search algorithm is initialised, the path time is calculated in each of the probe directions (x+,y+,x-,y-) using the function handle *pathtime.m* (Appendix B). This function handle accepts the starting point of the sailboat, the tack point and the destination point and integrates along the two legs of the journey to find the time elapsed (or the path time). This is the objective function value that needs to be minimised. With the objective function values from all four probes, the pattern search algorithm chooses the directions in x and y that favour the minimizing of the path time and determines the new tack point. Depending on the search directions (in X and Y) recorded, the pattern search algorithm updates the acceleration terms to reduce the number of pattern moves required to achieve convergence. Once convergence is reached, the optimal tack point (*xt*) and the optimal time elapsed values are returned.

This is continued for all such points around the course. The routine *single_tack_path.m* (Appendix C) is used to plot the path produced by this algorithm.

Figure 12: Single Tack Method Algorithm

Model 2 – Continuous VMG Maximization

In contrast to the previously discussed model, the VMG maximization model (VMGM) works on the principle of determining the optimal heading at any given point and then taking a step in that direction, and repeating the process until the destination is reached. In a sense, if the previous model only allowed


Start
Record Start Time
Specify starting point and destination
Generate vector to active targets
Determine feasible starting point for pattern search
Select pattern move directions in X and Y
Find time elapsed at probe lengths in x and y
Find optimal search direction
Update acceleration terms
Find new tack point
Is convergence criteria satisfied?
No
Yes
Record End Time
Stop

for one tack point (two legs) with the tack lengths and angles as variable, this method allows unlimited number of legs (or tacks) but the tack length is constrained to a small value. This enables the path determined to be a curve, if it is what results in the optimal path time. At each step, the optimal heading is determined using a search algorithm that is similar to a line search algorithm. However, the search is conducted at intervals of π /90 radians (2 degrees) in a 2π (360 deg) range instead of in a line bracket.

Algorithm Architecture

In order to obtain the path using this model, three routines are required. The main routine is called *travel.m* and is supplied in Appendix D. This routine accepts a list of all the target markers specified in coordinates and creates a tracking variable that is able to update the active target marker based on whether or not the path has 'touched' it. This means that the tracker will let the algorithm know if the path has not passed any of the markers in order. The tolerance can be varied.

Once the markers are specified, the *travel* routine calls on the *vmg_2.m* function handle Appendix E This handle accepts the current position of the sailboat and the active target as arguments and returns the optimal heading that should be pursued. It also provides the velocity attained by the sailboat, which is used to calculate the time step in the *travel* routine as well as the cumulative time elapsed. The heading is used by the routine to march the simulation forward and produce the optimal path.

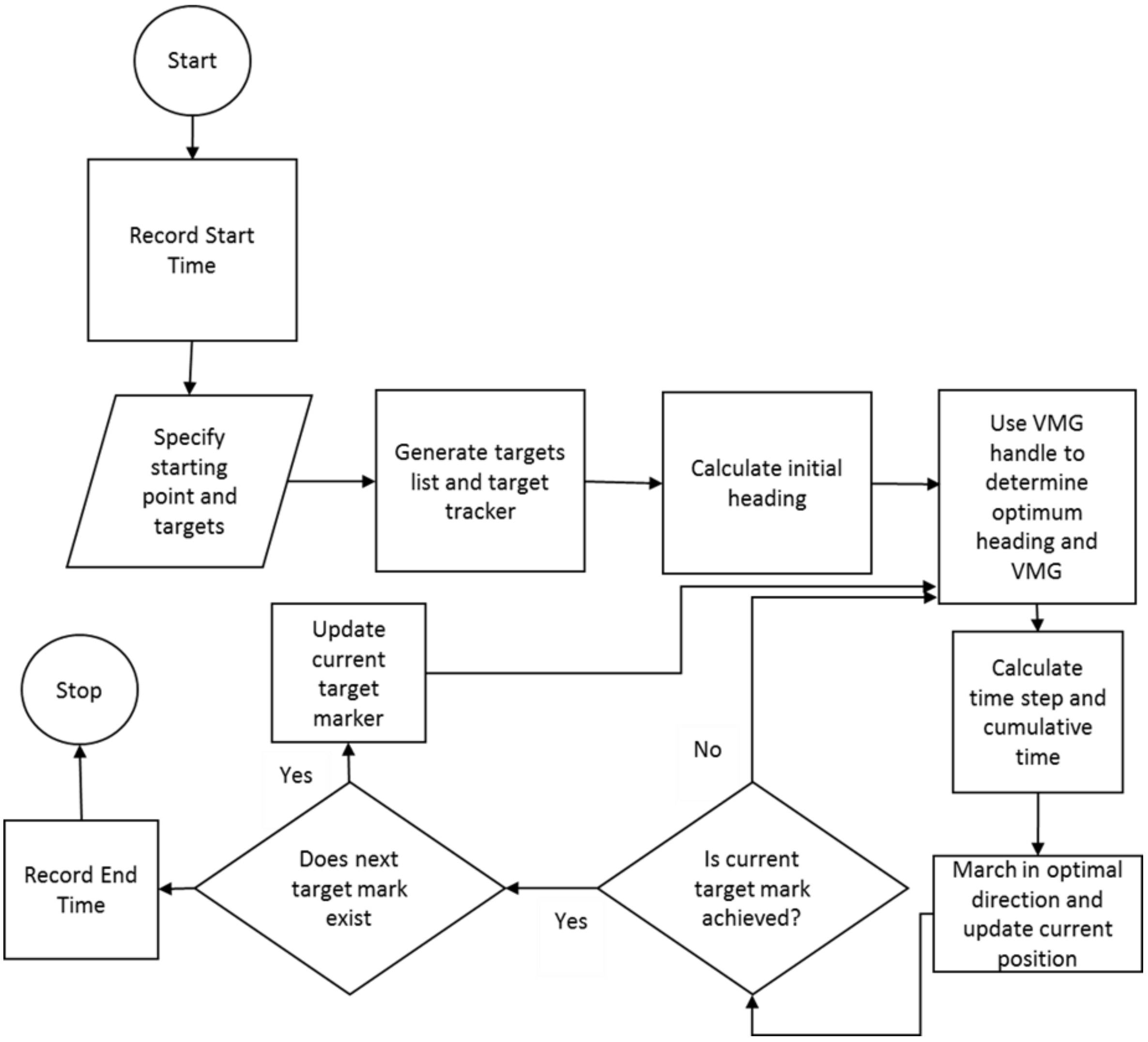

Start
Record Start Time
Specify starting point and targets
Generate targets list and target tracker
Calculate initial heading
Use VMG handle to determine optimum heading and VMG
Calculate time step and cumulative time
March in optimal direction and update current position
Is current target mark achieved?
No
Yes
Does next target mark exist
Yes
Update current target marker
Record End Time
Stop

Figure 13: Continuous VMG Maximization method

Results and Verification

For this project, in order to normalise the results, the velocity of the wind has been set to 1m/s. The results are presented and discussed in this section.

Convergence and Starting Points

In the VMGM method, the optimal path is determined by the algorithm only using the starting point and the location of markers. The path does in fact change with the starting point of the course, as is expected. However, the VMGM method does not use an optimization model where initial 'guesses' are required and is able to determine and build the path only based on the sailboats current location. Hence, this method is highly repeatable with no question of obtaining different results if different initial guesses are used.

However, the STM does require an initial guess for the tack point that is it be optimized to obtain the least path time. There is no need for the user to manually make this guess as it would require a fair bit of computation and can be an inconvenience. So, an initial feasible point calculator has been incorporated into the routine *st_3.m* that is able to do this for the user. Upon extensive testing for multiple cases, it is seen that the calculator is accurate and reliable 100% of the time. Nevertheless, in order to test whether the optimization algorithm is able to converge to the same optimal tack points for different starting points, the effects of different starting points are studied. Only three test cases are shown in this section. Plots for the remaining test cases are supplied in Appendix F.

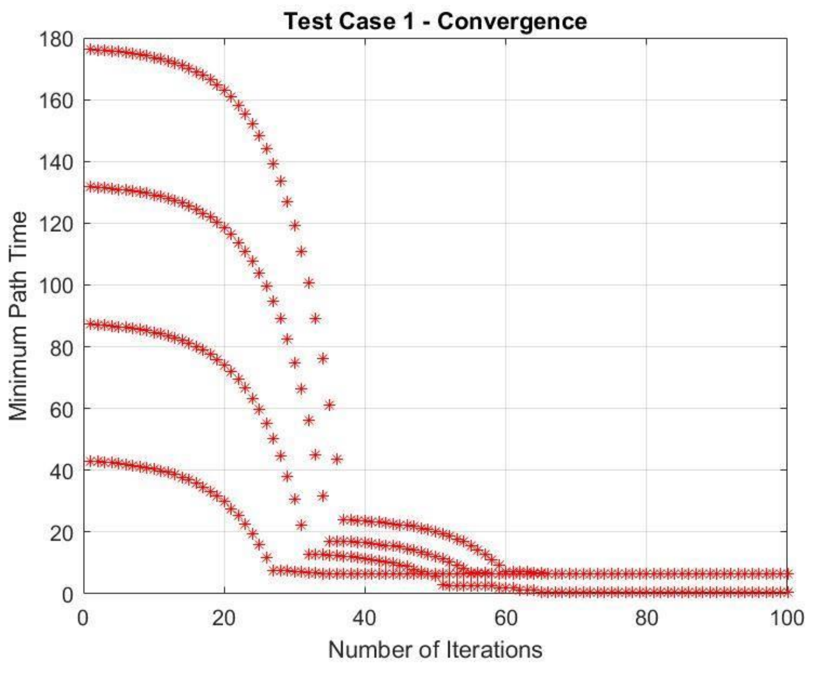


Figure 14: Convergence for Case 1

It is seen that the algorithm converges to the same location of the tack point (xt) within a 2% error and is very reliable for test case 1.

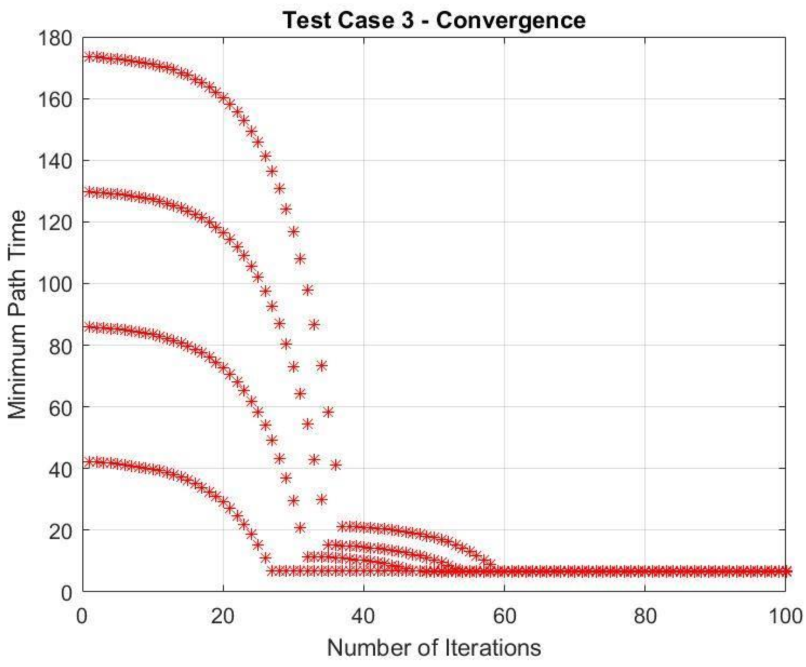


Figure 15: Convergence for Case 3

Test case 3 is an interesting one as the sailboat is required to travel to a target marker that is 10deg into the wind, in the no-go zone. It was suspected that the STM algorithm would have trouble converging, operating around the no-go zone but it is seen that this is not the case. Convergence has been achieved for all starting points tested. The tack points converged to within 0.1% for Test Case 2.

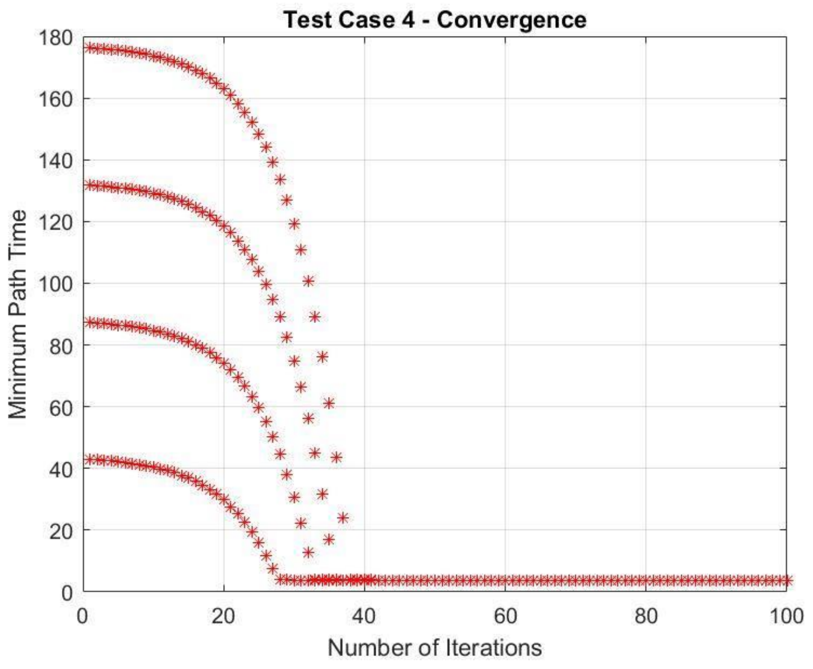


Figure 16: Convergence for Case 4

Similar to the performance seen in the earlier cases, the sensitivity to the initial guesses of the tack point is negligible and has virtually no effect on convergence,

Optimal Paths

The optimal paths obtained are discussed in this section. For each case, the path produced by the STM method and the VMGM method are presented. The locus produced by STM is colored Magenta whereas that produced by the VMGM method is coloured blue. All cases have starting point at origin (0,0). The origin and destination are indicated by large red circles.

<u>Case 1</u>

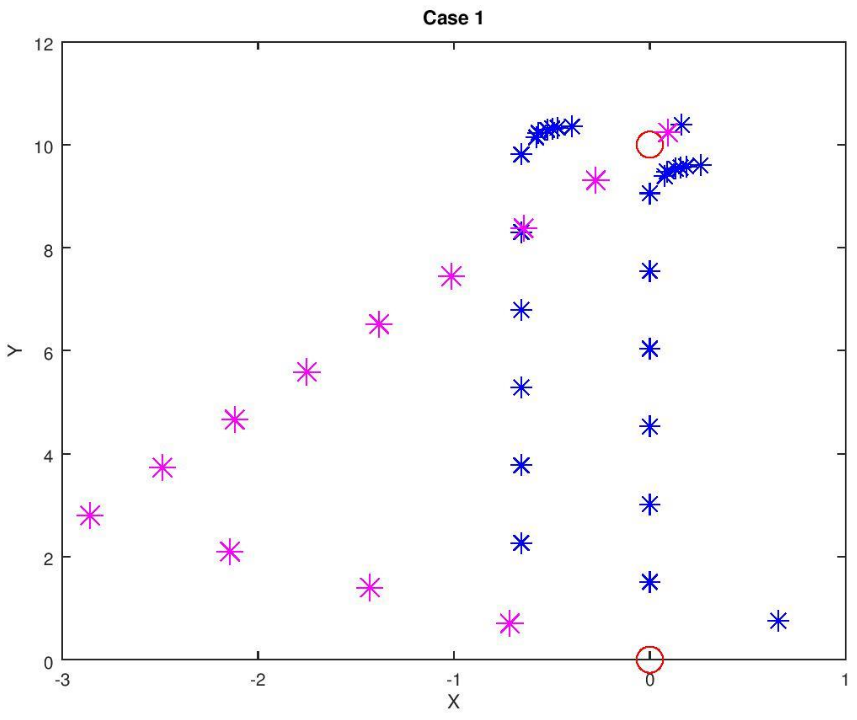


Figure 17: Optimal Paths for Case 1

For Case 1 where the sailboats are required to reach a destination directly upwind to the starting point, the optimal path produced by both methods are what is expected. It is seen that the tack angles for both paths are in fact the same. Considering both models approach the problem differently, it is assuring to find that both use the same tack angles. However, the obvious difference is that the VMGM method tacks many times where as the STM only tacks once.

Case 1

	STM	VMGM
Time (s)	6.314112	4.29286

The times obtained indicate that the VMGM method is able to produce a superior, quicker path. This es expected because the VMGM method is not restricted by the number of legs it can have in its path and continuously tries to optimize for max VMG.

Case 2

In this test case, the sailboat is required to start at origin and reach a destination that is 45 deg upwind. Although the paths taken by both algorithms are similar, the tack angles are in fact different if one were to look closely. The VMGM method (blue) approaches the destination at a higher angle and tacks just before reaching the destination, whereas the STM (magenta) tacks right at the beginning of its journey approaches the destination at a shallower angle.

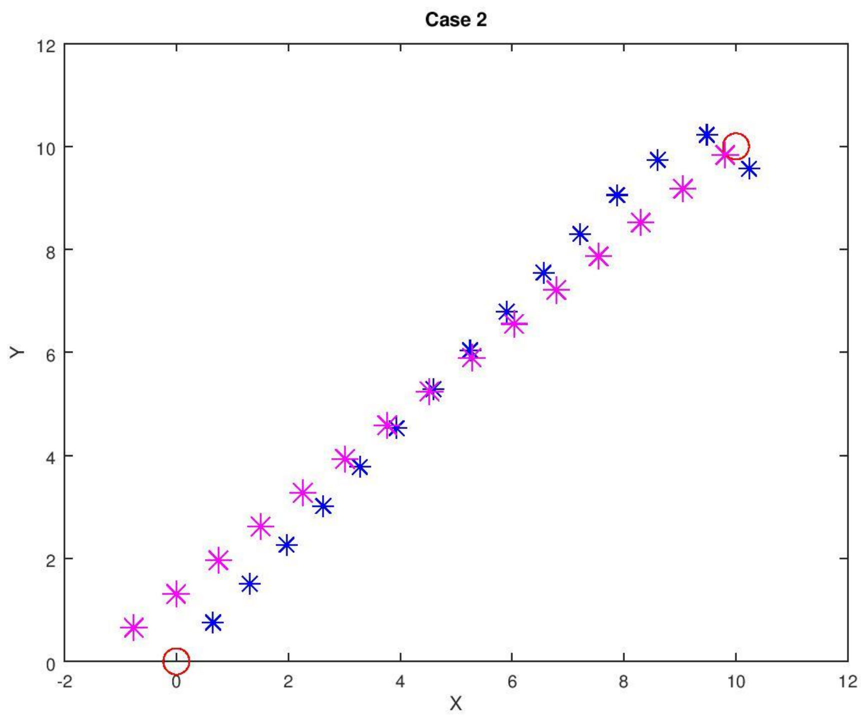


Figure 18: Optimal Paths for Case 2

In this case as well, the VMGM method proves to be superior. It is surprising that there can be such a significant difference in the times with what appears to be a fairly small difference in trajectory. However, upon conducting calculations, integrating along these paths with finer step sizes to validate the obtained times, it was confirmed that the step sizes are sufficiently small and the obtained times are valid as per the model.

CASE 2

	STM	VMGM
TIME (S)	8.506722	5.7529

Case 3

This test case involves sailing to a destination that is 10deg upwind of the starting point (origin). It is interesting to see that the paths taken by the models are completely different. The tack angles, however appear to be similar. As seen in test case 1, the VMGM method takes advantage of the freedom on the number of tacks it can use and produces a more efficient path.

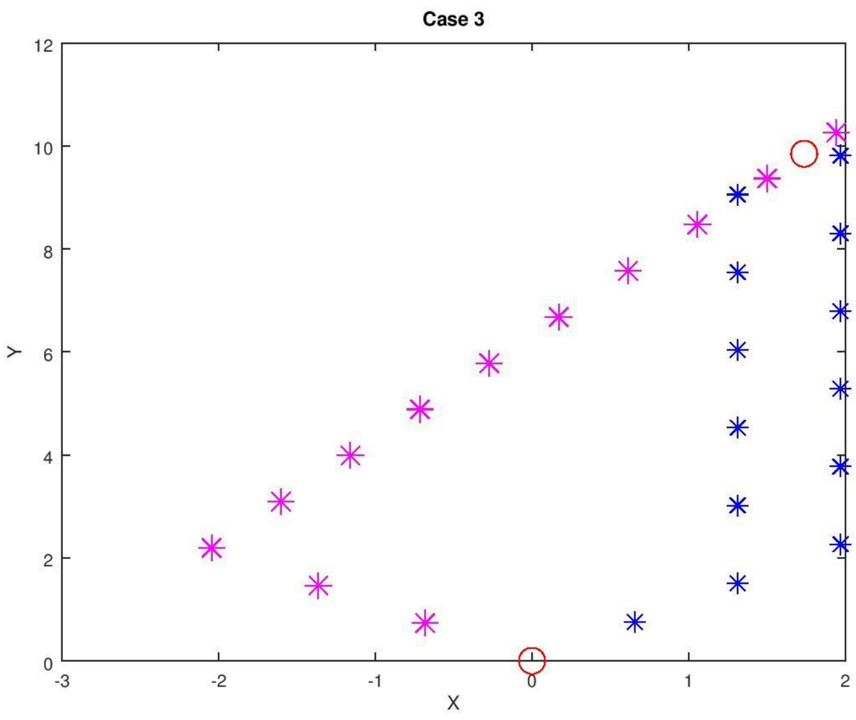


Figure 19: Optimal Paths for Case 3

Once again, the VMGM method produces a more efficient path. It is important to take note of the fact that tacking actually drains momentum from the boat. In reality, the VMGM produced path could actually be less efficient than its competitor purely due to the number of tacks.

CASE 3

	STM	VMGM
TIME (S)	6.347779	4.138

The VMGM method prefers to stay close to the destination in the X direction and tack continuously upwind whereas the STM method tends to sail away and then towards to destination.

<u>Case 4</u>

It is very interesting to see this behaviour in test case 4. In practice, sailors gybe their way to a target that is directly downwind to the starting point. Gybing is the opposite of tacking, where the heading is alternated from side to side in order to maximise the ground velocity and the velocity made good. Both algorithms have produced paths that gybe down from origin to the target at (0,-10).

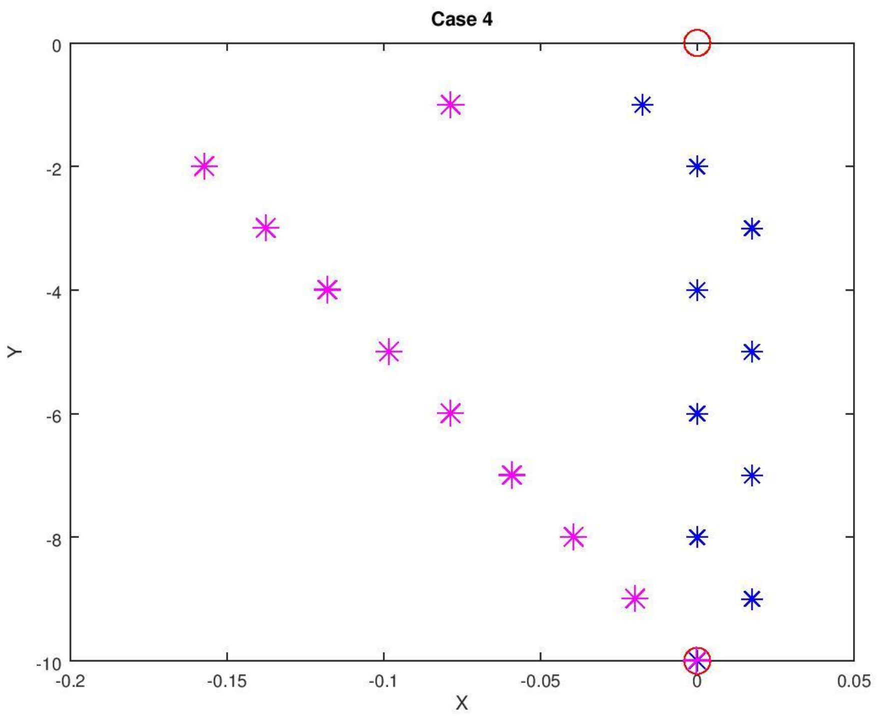


Figure 20: Optimal Paths for Case 4

The VMGM method is seen to take many more gybes and behave as indicated in previous cases whereas the SMT method moves away from the destination first and then changes course to approach it.

CASE 4

	STM	VMGM
TIME (S)	3.508444	3.2193

VMGM method produces a path shorter in the time dimension, however the difference between the results is not as high as that seen in other test cases. It is also observed that the times obtained for this test case are also the shortest of all cases.

Case 5

In test case 5, the sailboat is required to reach the destination that is 45deg downwind. While the VMGM method produces a smooth curve, the SMT method approaches the destination head on.

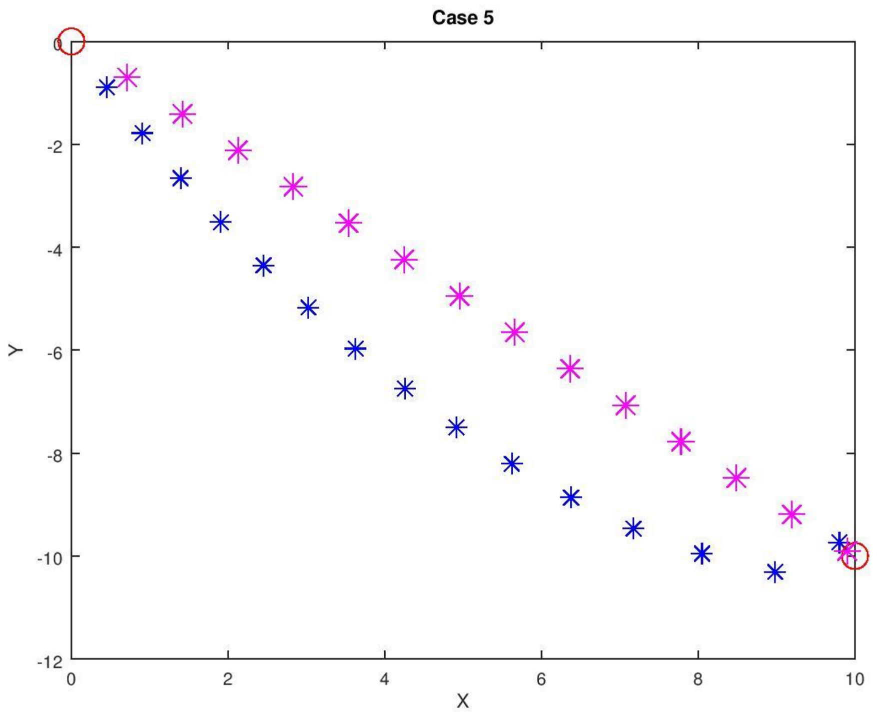


Figure 21: Optimal Paths for Case 5

It is clear that the SMT method does not tack in this test case, this only means that the optimal tack point located by the algorithm lies in between the destination and the starting point, thus producing a straight line. As expected, the VMGM method yields the better path due to its ability to produce curves.

CASE 5

	STM	VMGM
TIME (S)	6.203451	5.7052

Case 6

Test case 6 involved navigating from starting point at the origin to a destination that is perpendicular to direction of wind. IN this case the destination is located at (10,0).

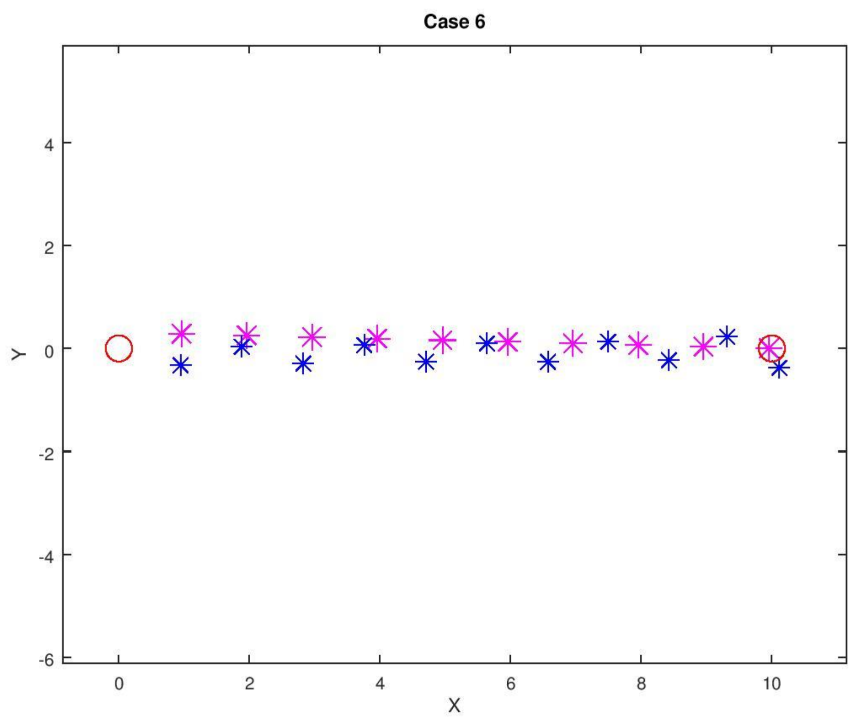


Figure 22: Optimal Paths for Case 6

As expected, the VMGM method pursues the maximization of VMG. This causes it to constantly keep manoeuvring to stay optimal. In contrast, the STM method algorithm produces a path that is virtually a straight line. In practice, it is common to see sailors taking a head on approach in a scenario such as this.

CASE 6

	STM	VMGM
TIME (S)	5.448371	4.5236

Verification of Optimality

The VMGM method employs a search algorithm that is similar to a 1 dimensional line search in cylindrical coordinates. However, this is done iteratively at each of the positions until the destination is reached. Due to the nature of the search algorithm used and because very fine increments were used, optimality at all points is certain. For the STM method however, it is important to check if optimality is achieved because it is based on a pattern search algorithm. In order to assure optimality, the acceleration terms and coefficients were monitored and altered throughout the process of developing the algorithm. It also incorporates a subroutine that detects and prevents the tack point from entering the infeasible region (no-go zone). To confirm optimality, the gradient of objective function (time) and the hessian matrix are produced for test cases 1 through 6.

Case	Gradient Vector	Hessian Matrix	
	-2.09E-04	3.49E-03	1.19E-06
1	8.35E-06	1.19E-06	4.91E-03
	-1.86E-04	4.79E-03	3.74E-04
2	1.81E-03	3.74E-04	6.74E-03
	-2.07E-04	3.51E-03	5.24E-04
3	1.46E-03	5.24E-04	4.94E-03
	-2.34E-04	3.51E-03	5.24E-04
4	-1.04E-07	5.24E-04	4.94E-03
	-7.76E-05	3.86E-03	7.31E-04
5	3.41E-04	7.31E-04	5.43E-03
	1.29E-03	3.30E-03	4.59E-04
6	-3.47E-04	4.59E-04	5.71E-03

It is seen that the gradients are in very small order of magnitude, approaching 0 and the Hessian matrix terms are positive and semi-definite. These are the minimum requirements that need to be satisfied in order to confirm optimality. Although these requirements are satisfied by the obtained solutions, the author has generated random paths, head on paths and created paths based on the solutions with slight deviations to see whether the obtained objective value is lower. This was not the case, confirming that the results lie in local minima at the very least. By starting the solution at different 'guess points' for the tack position (as discussed in an earlier section) it was confirmed the solutions converged to same coordinated. This further improves likelihood of the obtained solutions being optimal.

Case 7 - Racecourse

Cases 1 through 6 are used to create test scenarios that can be validated on their own so that when a composite course is created with elements from each of these, one can be more confident of the results obtained.

Case 7 involves completing a racecourse starting at origin (0,0) and touching each of the targets in an anticlockwise manner. It is seen that the paths produced by the models are very different. The VMGM model tacks toward the initial destination (20,50) first, maintains bearing until the target is dead north. Then it appears to generate tacks with really small tack lengths (high frequency) to arrive at the destination. The STM method tacks away from the no-go zone, clears it and then tacks bearing towards the first target. The paths from the second (20,50) to third targets (-100,0) and the third to fourth (20,-50) targets are in fact similar (but mirror images). In the downwind direction the VMGM method tends to curve to the same X coordinate as the destination and then proceed towards it in the Y direction. The STM however tacks very slightly, late as it approaches the destination at a shallower angle.

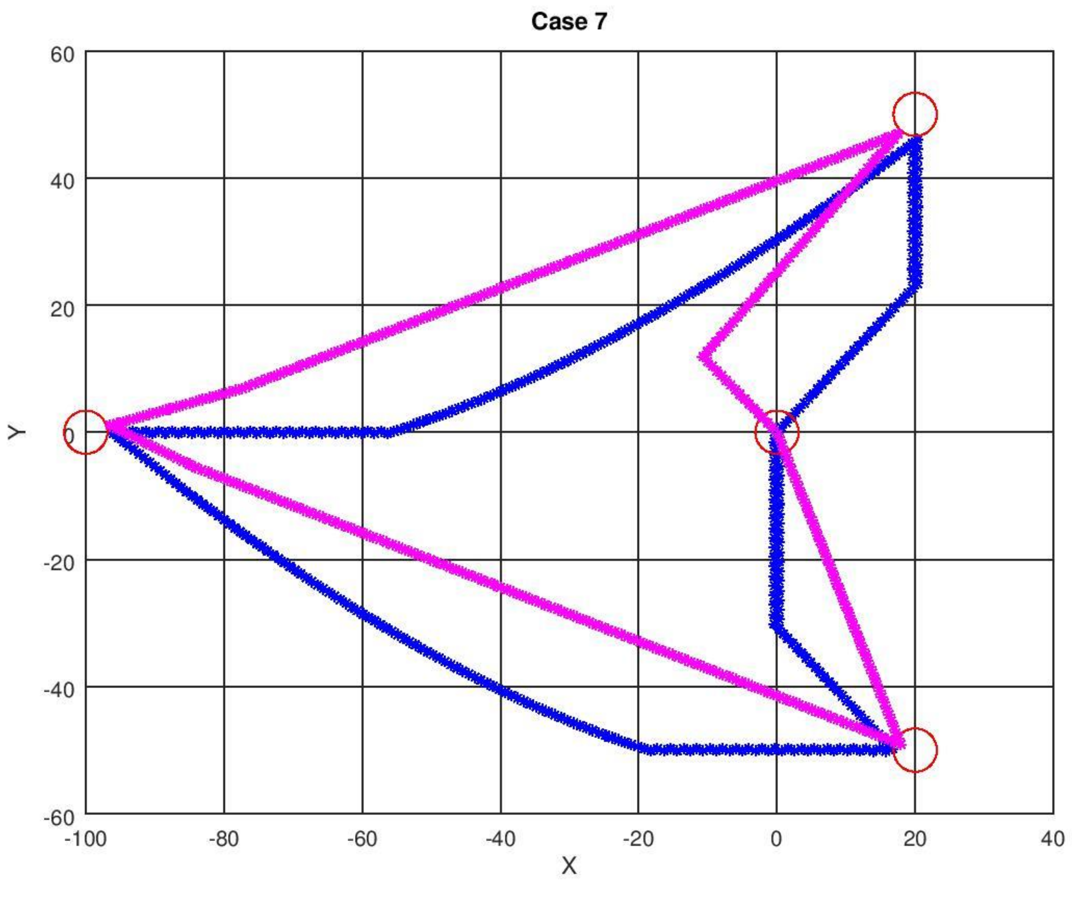


Figure 23: Optimal Paths for Case 7

The final legs from (20,-50) to the origin (0,0) are again similar to the first legs from (0,0) to (20,50). This is expected as the points are symmetric about the Y axis. The VMGM method executes a tack to position the destination directly north of it. And as seen earlier, it manoeuvres using a high frequency, small length tacks to reach the destination.

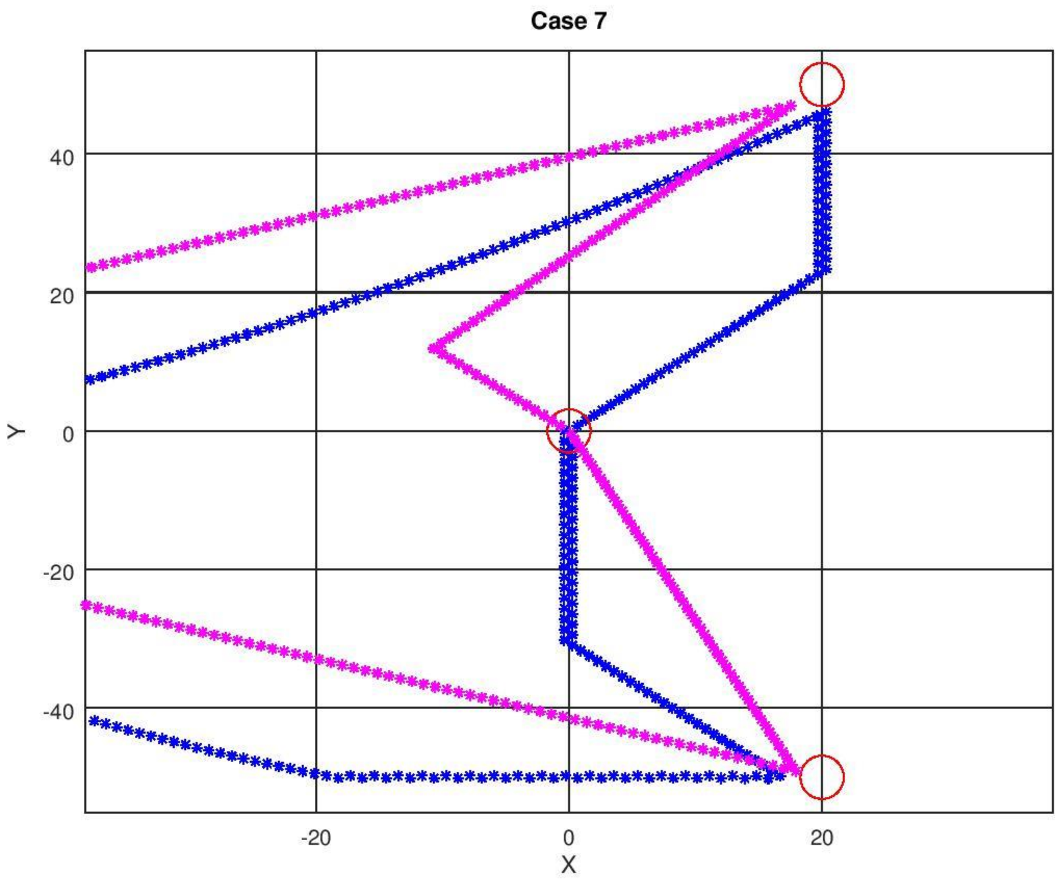


Figure 24: Optimal Paths from (0,0) to (20,50)

As expected, the VMGM method has produced a path resulting in a shorter path time.

CASE 7

	STM	VMGM
TIME (S)	6.347779	4.138

Solver Performance

The solver performance for this project is characterised by the path time and computation time. Shown in figure 25 is the path times obtained for the 6 test cases presented earlier. Test case 7 has not been included as it is a combination of the others and is significantly longer than the other test cases.

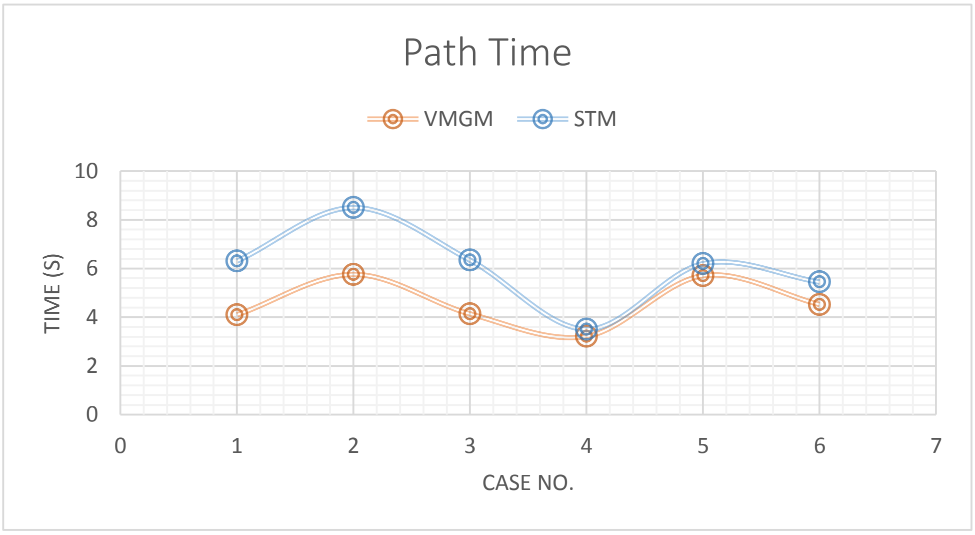


Figure 25: Path Times for Test Cases

The VMGM model has consistently produced the more desirable path, as seen from the plot shown above. Although the difference is not much, it is important to remember that these cases are tested for fairly short courses. As the course size increases, the times are expected to vary linearly. This is similar to how a lead of a few tenths of a second in a Formula 1 Grand Prix lap stack up over the entire race, creating a vast gap towards the finish. On the other hand however, the VMGM method ignores the fact that high frequency manoeuvres drains the momentum (speed) from the sail boat. It would be very interesting to quantify these effects and model them in a future piece of work.

The VMGM solver is also easier to use, and more robust than the STM solver. It produces highly repeatable results whereas the STM algorithm still needs to calculate an initial feasible point. The VMGM solver also deals with the no-go zone in a hassle-free manner whereas the STM algorithm requires additional control terms to prevent instability.

The computation time for each run was recorded and plotted. This is shown in figure 25. The STM solver uses a function handle that calculates the path time by integrating over a set of discretized line segments. This is a computationally expensive process which happens simultaneously in the VMGM algorithm as it marches forward.

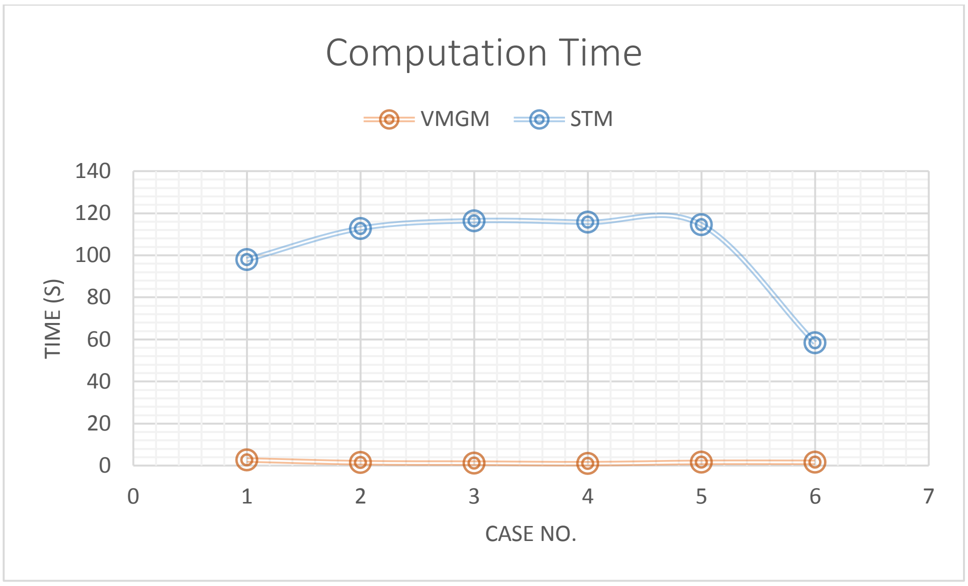


Figure 25: Computation Times for Test Cases

The iterations were not plotted because each algorithm has multiple inner iterations running at the same time. Furthermore, computation time is a much more intuitive and tangible parameter to compare algorithms.

Use of Alternate Solvers

During the start of the project, the solvers used to conduct the optimization in the STM model were based on the method of steepest descent (with line search) and the pure Newton method. The method of steepest descent was incredibly sensitive to the starting points. Even if the initial points lied in the feasible region, the algorithm would soon become unstable and enter an infinite loop. The reason behind this is the way the no-go zone is modelled. Instead of using a Penalty-Barrier approach, the no-go zone is modelled as a step function that drops the velocity to zero in the region. This singularity is not handled well by the method of steepest descent or the Newton methods. For the starting points that the method of steepest descent did work for, convergence was achieved very quickly and the likelihood of the solution being optimal was very high. The Newton method performed even poorer. Most trials and starting points ended in infinite loops as the algorithm struggled to calculate the Hessian matrices. Codes supplied in Appendix G.

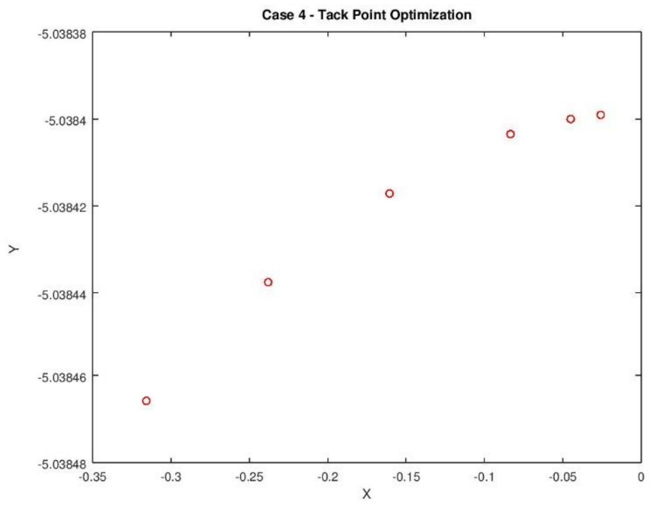


Figure 27: Successful run of steepest descent

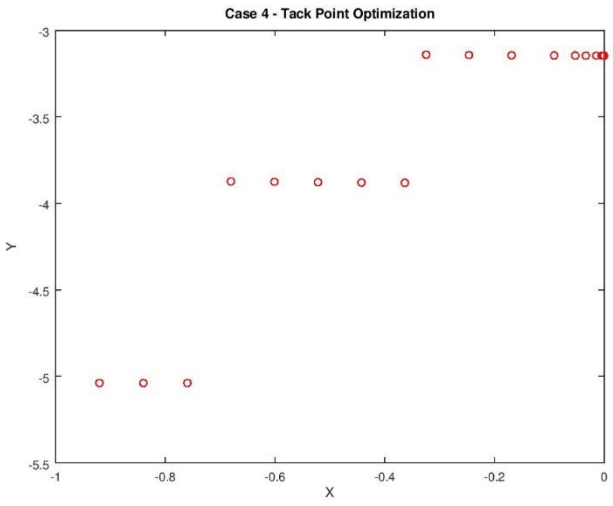


Figure 28: Infinite loop due to starting point sensitivity

Appendix A

```
%Routine to determine the best tack point between two marks and valculate time

x_tar_c = [0,0];  %active destination

x0 = [20,-50];    %starting point


%calculating initial xt--------------------------------------------------------

  vect_tar = (x_tar_c - x0);    %generate vector to active target

  mid =0.5*(vect_tar) + x0;       %place point in the middle

  deviation = ((x_tar_c(2)-x0(2))^2/(x_tar_c(1)-x0(1)))/(atan(deg2rad(15)));  %define deviation of
point from vect_tar


  xt0 = [-deviation,0] + mid; %initial tack point

  xt = xt0;


  %Introducing probes for pattern search


  plength_x  = [10,0]; %probe length in x

  plength_y  = [0,10]; %probe length in y


  iter=0;

  ax=0.1;        %initial stepsize

  ay=0.1;        %initial stepsize


  adpxt=1;       %initialise acceleration determining parameter in x

  adpyt=1;       %initialise accleration determining parameter in y


while iter<100    %set maximum number of iterations


  %calculate path times at probes

pathtime_xtc = pathtime(x_tar_c,x0,xt); %path time if current tack point is used
```

```
pathtime_xtc_xplus = pathtime(x_tar_c,x0,xt + plength_x);
pathtime_xtc_xminus = pathtime(x_tar_c,x0,xt - plength_x);

pathtime_xtc_yplus = pathtime(x_tar_c,x0,xt + plength_y);
pathtime_xtc_yminus = pathtime(x_tar_c,x0,xt - plength_y);

%probing in x and updating xt

%find minumum pathtime
if min([pathtime_xtc, pathtime_xtc_xplus, pathtime_xtc_xminus]) == pathtime_xtc
  xt = xt;
  sx = 0;     %set search direction
  adpx = 0;   %set acceleration term

 elseif min([pathtime_xtc, pathtime_xtc_xplus, pathtime_xtc_xminus]) == pathtime_xtc_xplus

  sx = +1;    %set search direction
  adpx = +1;  %set acceleration term

  elseif min([pathtime_xtc, pathtime_xtc_xplus, pathtime_xtc_xminus]) == pathtime_xtc_xminus

  sx = -1;    %set search direction
  adpx = -1;  %set acceleration term

  endif

%probing in y and updtaing xt

  if min([pathtime_xtc, pathtime_xtc_yplus, pathtime_xtc_yminus]) == pathtime_xtc

  sy = 0;   %set search direction
  adpy = 0; %set acceleration term

 elseif min([pathtime_xtc, pathtime_xtc_yplus, pathtime_xtc_yminus]) == pathtime_xtc_yplus

    sy = 1; %set search direction
  adpy = 1; %set acceleration term

  elseif min([pathtime_xtc, pathtime_xtc_yplus, pathtime_xtc_yminus]) == pathtime_xtc_yminus

    sy = -1;  %set search direction
  adpy = -1;  %set acceleration term

  endif

  iter = iter + 1;

  %plot(xt(1),xt(2)); hold on;



  ax=ax*1.15^(adpx*adpxt);  %update acceleration in x
  ay=ay*1.15^(adpy*adpyt);  %update acceleration in y

  xt_p=xt;    %update xt previous

  xt = xt + [ax*sx,ay*sy];  %update tack position



 adpxt=adpx;             %store acceleration terms
  adpyt=adpy;

  minpathtime = pathtime(x_tar_c,x0,xt);  %calculate corresponding minpathtime
```

```
  minpathtime_p = pathtime(x_tar_c,x0,xt_p);
  %plot(iter,minpathtime,'*r',"markersize",15); hold on;

  %prevent entering into infeasible region

  if  minpathtime_p<minpathtime,
    xt=xt_p;
    minpathtime = minpathtime_p;
    ax= 0.1;    %reset acceleration
    ay = 0.1;   %reset acceleration
    endif
endwhile
```

Appendix B

```
%pathtimeintegrator
function t = pathtime(x_tar_c,x0,xt)


 %x_tar_c = active target
 %x0      = current position
 %xt      = tack coordinates
 vel_wind = 10;         %wind velocity
 wind_dir=[0,-1];       %wind direction
 theta_nogo = deg2rad(40);     %deadzone
 velcons = 3;   %velocity increase constant
 deg_int = 5;           %constant provided by manufacture

 step_inc = 0.1; %step size



 tack_vect_1 = xt - x0;  %tack vector 1
 tack_vect_2 = x_tar_c - xt; %tack vector 2



 steps_1 = norm(xt - x0)/step_inc; %divide it into increments of .1 metres



 step_c_1 = 0;  %set initial step

 heading = tack_vect_1;
 xc = x0;
 time_elapsed_1 = 0;
 %--------------------------------------------------------------------------
 while step_c_1<steps_1

 target_vect = x_tar_c - xc;  %vector to target from current x
```

```
theta_rd = real(acos(dot(heading,target_vect)/(norm(heading)*norm(target_vect))));
theta_rw = real(acos(dot(heading,wind_dir)/(norm(heading)*norm(wind_dir))));


  if dot(wind_dir,heading)/(norm(wind_dir)*norm(heading))<0

   u = cos(theta_rd)*(vel_wind/cos(theta_nogo))*(1+0.01*velcons).^(abs(theta_rw-
theta_nogo)*180/(pi*deg_int)); %calculate velocity made good
   v = (rad2deg(acos(dot(heading,wind_dir)/(norm(heading)*norm(wind_dir))))<140)*u;

   elseif dot(wind_dir,heading)/(norm(wind_dir)*norm(heading))>0

   v = cos(theta_rd)*(vel_wind/cos(theta_nogo))*(1+0.01*velcons).^(abs(pi-theta_rw-
theta_nogo)*180/(pi*deg_int));
   %if downwind

   endif


vel_tack = v/cos(theta_rd);

if vel_tack>10e-2
time_increment_1 = 0.1/vel_tack; %time elapsed in incremement

time_elapsed_1 = time_elapsed_1 + time_increment_1;
xc = 0.1*heading/norm(heading) + xc;

step_c_1 = step_c_1 + 1;

elseif vel_tack<10e-2
 time_elapsed_1 = 10e6;
 break
 endif


%plot(xc(1),xc(2)); hold on;
endwhile


t_1 = time_elapsed_1;
%----------------------------------------------------------------------------
%--calculate time from tackpoint to active target


 steps_2 = norm(x_tar_c - xt)/step_inc; %divide it into increments of .1 metres
 step_c_2 = 0;  %set initial step


heading = tack_vect_2;
xc = xt;
time_elapsed_2 = 0;


while step_c_2<steps_2


target_vect = x_tar_c - xc;  %vector to target from current x


theta_rd = real(acos(dot(heading,target_vect)/(norm(heading)*norm(target_vect))));
theta_rw = real(acos(dot(heading,wind_dir)/(norm(heading)*norm(wind_dir))));



  if dot(wind_dir,heading)/(norm(wind_dir)*norm(heading))<0


   u = cos(theta_rd)*(vel_wind/cos(theta_nogo))*(1+0.01*velcons).^(abs(theta_rw-
theta_nogo)*180/(pi*deg_int)); %calculate velocity made good
   v = (rad2deg(acos(dot(heading,wind_dir)/(norm(heading)*norm(wind_dir))))<140)*u;


   elseif dot(wind_dir,heading)/(norm(wind_dir)*norm(heading))>0
```

```
    v = cos(theta_rd)*(vel_wind/cos(theta_nogo))*(1+0.01*velcons).^(abs(pi-theta_rw-
theta_nogo)*180/(pi*deg_int));

    %if downwind


    endif




 vel_tack = v/cos(theta_rd);


  if vel_tack>10e-2


 time_increment_2 = 0.1/vel_tack; %time elapsed in incremement


 time_elapsed_2 = time_elapsed_2 + time_increment_2;

 xc = 0.1*heading/norm(heading) + xc;


 step_c_2 = step_c_2 + 1;


 elseif vel_tack<10e-2

  time_elapsed_1 = 10e6;

  break

  endif


 %plot(xc(1),xc(2)); hold on;

 endwhile


 t_2 = time_elapsed_2;


 t = [t_1+t_2];

 %}


 end
```

Appendix C

```
%plot single tack path
t0 = clock ();  %record start time of function


x0 = [0,0]; %starting points


target_tol = 4; %radius in metres within target to qualify
xp = x0;    %current position
x_tar= [-13.34947689    14.84067334;20,50;-80.87069946          5.397111456;-100,0;-80.75569946
            -7.316388104;20,-50;19.61212559         -49.87259138;0,0]; %specify targets in io




%Initialize target tracking-------------------------------------------------------------
num_targets = [size(x_tar)](1);  %get number of targets


itt=1;  %target tracker iteration


while itt<num_targets+1 %create file indicating no targets have been achieved


target_tracker(itt) = 0;
itt=itt+1;
endwhile
save target_track.mat target_tracker;
%---------------------------------------------------------------------------


x_tar_c = [x_tar(1,1),x_tar(1,2)]; %set initial target
tar_act=1;    %active target is 1
%---------------------------------------------------------------------------


while norm(x_tar_c - xp)>target_tol/10  %if target is not achieved


  step_direction = (x_tar_c - xp)/(norm(x_tar_c-xp));
  stepsize = 1;
  xp_p=xp;  %store previous location
  xp = step_direction*stepsize + xp;



   plot(xp(1),xp(2),'*m','markersize',5); hold on;




  if norm(x_tar_c - xp)<target_tol
    if tar_act<size(x_tar)(1)
    target_tracker(tar_act)=1;
    x_tar_c=[x_tar(tar_act+1,1),x_tar(tar_act+1,2)];
    tar_act=tar_act+1;

     end
  end
%---------------------------------------------------------------------------
    solvertime = etime (clock (), t0);  %record elapsed time

  endwhile
```

Appendix D

```
%Routine that advances the algorithm in steps till destination is reached
t0 = clock ();  %record start time of function
x0 = [0,0]; %starting points
vel_wind = 10; %absolute wind velocity
wind_direction = [0,-1];

target_tol = 4; %radius in metres within target to qualify
xp = x0;    %current position
x_tar= [20,50;-100,0;20,-50;0,0]; %specify targets in io

time_elapsed = 0;

%Initialize target tracking------------------------------------------------------------
num_targets = [size(x_tar)](1);  %get number of targets

itt=1;  %target tracker iteration

while itt<num_targets+1 %create file indicating no. targets have been achieved

target_tracker(itt) = 0;
itt=itt+1;
endwhile
save target_track.mat target_tracker;
%---------------------------------------------------------------------------

x_tar_c = [x_tar(1,1),x_tar(1,2)]; %set initial target
tar_act=1;    %active target is 1
%----------------------------------------------------------------------------

heading=[1,1]/(norm([1,1]));  %set intial heading
```

```
%---------------------------------------------------------------------------
while norm(x_tar_c - xp)>target_tol/10  %if target is not achieved


  opt = vmg_2(xp,x_tar_c,heading); %obtain optimal direction, vmg


  step_direction = [opt(1), opt(2)];  %in optimal direction
  stepsize = 1;                       %step size is 1 metre


  xp_p=xp;  %store previous location of sailboat
  xp = step_direction*stepsize + xp;  %march forward in time


  time_step = opt(4);                       %calculate time_step
  time_elapsed = time_elapsed + time_step;  %calculate cumulative time_elapsed


   plot(xp(1),xp(2),'*b','markersize',5); hold on;              %plot path



  %change target to next one specified if current target is achieved
  if norm(x_tar_c - xp)<target_tol
    if tar_act<size(x_tar)(1)
    target_tracker(tar_act)=1;
    x_tar_c=[x_tar(tar_act+1,1),x_tar(tar_act+1,2)];
    tar_act=tar_act+1;


     end
  end
%---------------------------------------------------------------------------
  %calculate relative directions
  heading=xp-xp_p;
  solvertime = etime (clock (), t0);  %record elapsed time


 %plot(0,0,'or',"markersize",15); hold on; plot(10,0,'or',"markersize", 15);
plot(10,10,'ow',"markersize", 15)
```

```
title ("Case 7");

xlabel ("X");

ylabel ("Y");

  endwhile


  grid on;

 plot(20,50,'or',"markersize",20); hold on; plot(-100,0,'or',"markersize", 20); plot(20,-
50,'or',"markersize", 20); plot(0,0,'or',"markersize", 20);

%  title ("Case 7 - Race Course");
```

Appendix E

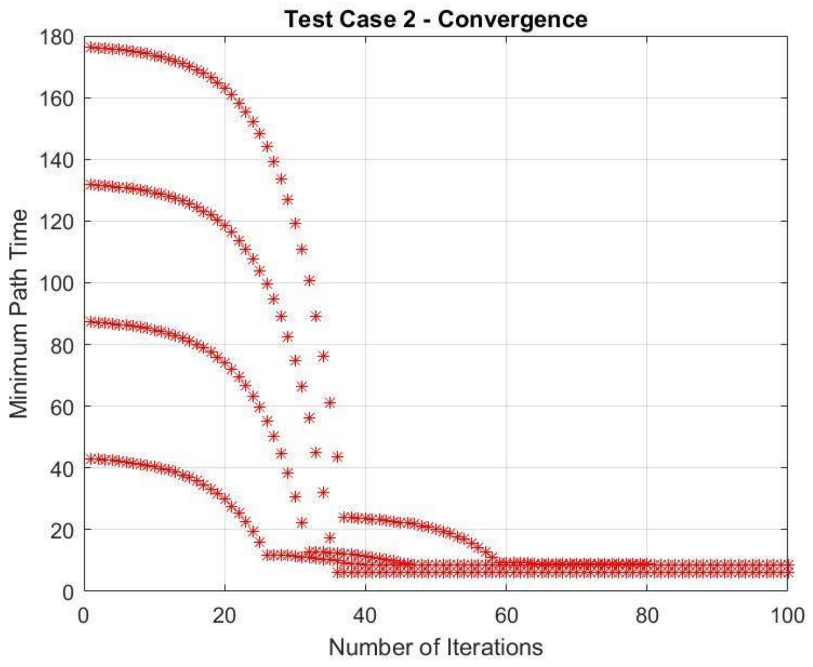
VMG - Function Handle to calculate optimal heading, max vmg and time step

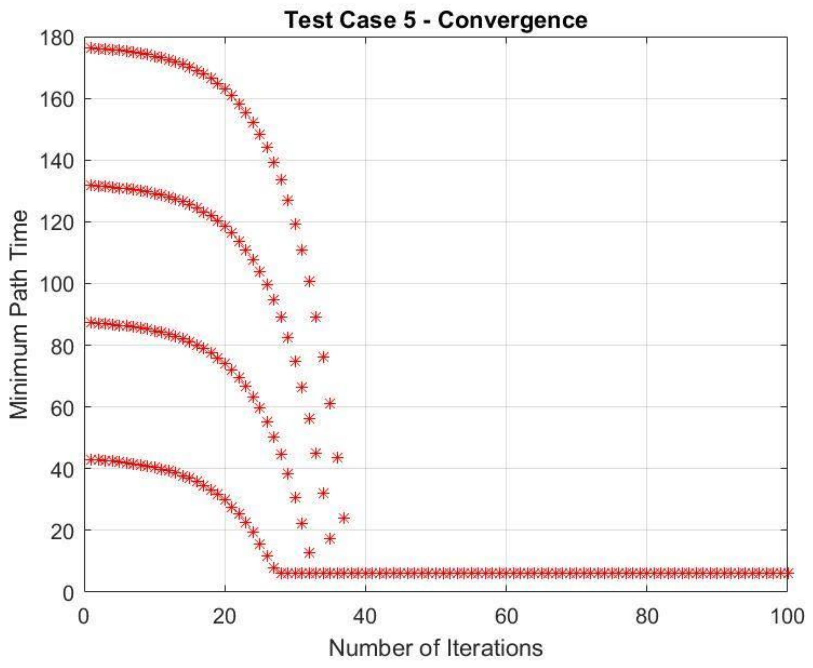
```
function opt =vmg_2(xp,x_tar_c,heading_c)

%theta_rw,theta_rd

%CONSTANTS

vel_wind = 1;         %wind velocity

wind_dir=[0,-1];       %wind direction

theta_nogo = deg2rad(40);     %deadzone

velcons = 3; %was3  %velocity increase constant

deg_int = 5;           %constant provided by manufacture


%theta_rd_deg = rad2deg(theta_rd);

%theta_rw_deg = rad2deg(theta_rw);


%VARIABLES

%theta_rw = angle of direction of travel relative to wind

%theta_rd = angle of direction of travel relative to destination



heading = heading_c;    %current heading

vect_tar = x_tar_c-xp;  %vector to target

vmg_iter = 0;            %number of iterations

heading_opt =[1,1];     %initial heading optimal

vmg_opt = 0;


%while loop to find optimal direction


while vmg_iter<360


%theta_rd is the angle between heading and destination

theta_rd = real(acos(dot(heading,x_tar_c-xp)/(norm(heading)*norm(x_tar_c-xp))));
```

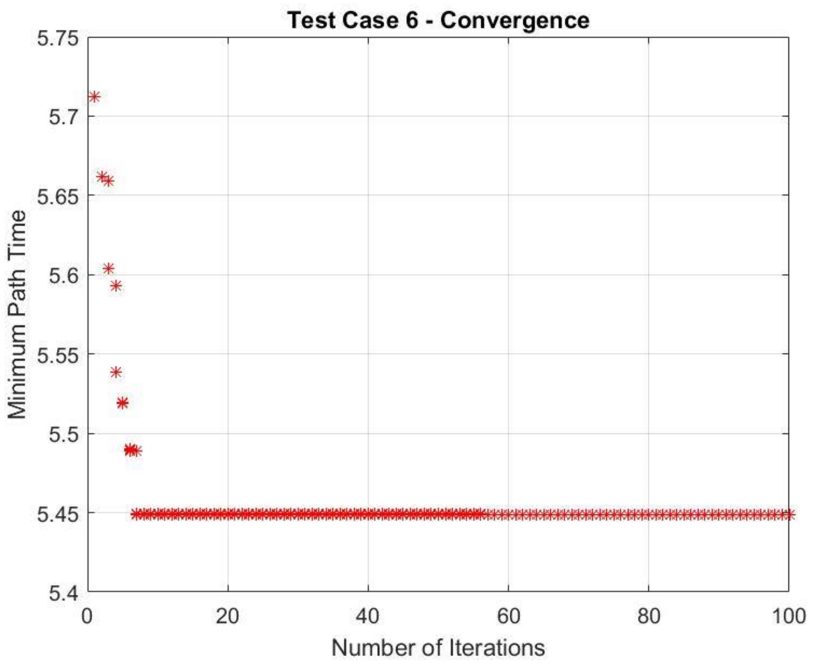
```
%theta_rw is the angle between wind direction and heading

theta_rw = real(acos(dot(heading,wind_dir)/(norm(heading)*norm(wind_dir))));


%calculate VMG based on upwind or downwind

if dot(wind_dir,heading)/(norm(wind_dir)*norm(heading))<0


%calculate VMG

u = cos(theta_rd)*(vel_wind/cos(theta_nogo))*(1+0.01*velcons).^(abs(theta_rw-
theta_nogo)*180/(pi*deg_int)); %calculate velocity made good


%assign 0 value to VMG if the heading is in the no-go zone

v = (rad2deg(acos(dot(heading,wind_dir)/(norm(heading)*norm(wind_dir))))<140)*u;


elseif dot(wind_dir,heading)/(norm(wind_dir)*norm(heading))>0


v = cos(theta_rd)*(vel_wind/cos(theta_nogo))*(1+0.01*velcons).^(abs(pi-theta_rw-
theta_nogo)*180/(pi*deg_int));

%if downwind


endif


%search routine to find optimal heading

if v>vmg_opt

  vmg_opt = v;

  heading_opt = heading;


elseif v<vmg_opt

  vmg_opt = vmg_opt;


vmg_iter=vmg_iter+1;


%plot(vmg_iter,vmg_opt); hold on;
```

```
endif

  heading = rot(pi/90,heading);

endwhile



%calculate time_step
time_step = (1)/(vmg_opt/cos(theta_rd));

%generate return vector
opt = [heading_opt,vmg_opt,time_step];

end
```

Appendix F


Test Case 2 - Convergence
Minimum Path Time
Number of Iterations
0
20
40
60
80
100
120
140
160
180
0
20
40
60
80
100


Test Case 5 - Convergence
Minimum Path Time
Number of Iterations
0
20
40
60
80
100
120
140
160
180
0
20
40
60
80
100


Test Case 6 - Convergence
Minimum Path Time
Number of Iterations
5.75
5.7
5.65
5.6
5.55
5.5
5.45
5.4
0
20
40
60
80
100

Appendix G

```
function Xk =SteepestDescent_DLS(x)


Xj=[0;0];
Xs=x;     %Starting points
Xi=[Xs(1);Xs(2)];
Xt=Xi;    %Xt holds Xs information from previous iteration


i=1;      %Initialise iterations


%convergence criteria to detect near zero gradient
while abs(grad(Xi(1),Xi(2)))>[10e-7,10e-7]


Xj=Xi;
gradi=grad(Xi(1),Xi(2));    %gradient of objective function at x,y


al=10;    %search bracket length
a=0.5*al; %initial search length to enter while statement
ai=al;
il=0;     %linesearch iteration
as=0;     %as is lower bracket of line search
at=al;    %at is higher bracket of line search


  while abs(a-ai)>10e-6    %convergence criteria to detect very small changes
     ai=a;


    %Dichotomous line search is employed with ap and aq used to
    %probe function value at either side of a length
    %the lengths are expressed as a function of the local Xj vector
     ap=a*0.99;         %ap is probe length (at Xj)
     aq=a*1.01;         %aq is probe length in opposite direction (Xj)
    %objp and objq are function values at either side of the search length
    objp=obj([Xj-ap*gradi](1),[Xj-ap*gradi](2));
    objq=obj([Xj-aq*gradi](1),[Xj-aq*gradi](2));



   %select side of the bracket with lower function value
   if objp<objq     %if left side gives lower function value
      at=a;        %set right bracket as a
      as=as;       %leave left bracket as is

   elseif objp>objq  %if right side gives lower function value
         at=at;    %leave right bracket as is
         as=a;     %set left bracket as a
   endif



 a=0.5*(at-as);        %set new search length as in between the brackets
 il=il+1;              %update iteration for line search

 %plot(i,obj([Xj-ai*gradi](1),[Xj-ai*gradi](2))); hold on;
 %plot(i,ai); hold on;

  endwhile   %end of linesearch algo

Xj=Xi-a*gradi; %update Xj vector
            %Xj= Xj-0.001*gradi;

Xt=Xi;         %update Xt
Xi=Xj;         %update Xj

i=i+1;         %update iteration value
```

```
plot(Xj(1),Xj(2),'ro'); hold on; %plot how solution travels



endwhile        %end solution search

objs=obj(Xj(1),Xj(2));  %objective function at final solution

Xk=[Xj;i;objs];

ti=i; %total iterations

%plot(i,(Xj(1)),'-'); hold on; plot(i,(Xj(2)));
%plot(i,obj(Xj(1),Xj(2))); hold on;

%contour plot of objective function

%m=[(min(0,min(x))):0.1:(max(10,x))];
%n=[(min(0,min(x))):0.1:(max(10,x))];
%[M,N]=meshgrid(m,n);
%O=(12+(M.^2)+(1+N.^2)./(M.^2)+(((M.*N).^2+100))./((M.*N).^4))*0.1;
%contour(M,N,O,[10e5,10e5,10e4,10e4,5000,5000,1000,1000,100,100,10,10,8,8,6,6,4,4,2,2]);



title ("Case 4 - Tack Point Optimization");

xlabel ("X");
ylabel ("Y");

end
```