Explanation of Used Algorithms

**extractSteadySpeeds**

1) Finding preliminary steady-speed intervals using *fifo\_maxdev* algorithm. This algorithm does not always give good results/intervals. It is, let's say, the first iteration. The resulting intervals are just used for estimating the parameters that will be used in the sieve-algorithm. It is about *referent variance* and *degrees of freedom*. In other words *mm2* and *ff* which will be used in *sieve\_maxdev* function.

1a - In statistical sense, not all the intervals are good. There are intervals which does not satisfy the statistical *F-test* and these intervals are not considered to be 'good'. We find *mm2* as the maximal statistical-dispersion of all good intervals. The corresponding *ff* is a number of totes inside that interval.

1b - The good intervals cannot be considered as optimal. The result of *fifo\_maxdev* algorithm depends on the direction of iterating and they are never best regarding the coverage.

2) Finding steady-speed intervals using the sieve-algorithm.

3) Adjusting the resulting intervals (*adjustDevTouchingIntervals* function). At this point, it is possible to have two neighboring intervals which are both good but not perfect. For example, if we have two touching intervals 100-140 and 140-200, it might be that the more optimal combination would be 100-150 and 150-200. The used criterion for finding the most optimal combination is a classical statistical criterion - minimal sum of squares of deviations. The results after applying *adjustDevTouchingIntervals* function, can be considered optimal.

4) Extending the resulting intervals. In rare (but possible) cases, it's possible that sections (on the left or right side) which obviously belong to the steady interval are omitted. An example is: github.com/debrief/LegAlgorithms/blob/master/test\_results/problem.png. This example looks like a paradox but it is simply so from the aspect of mathematical statistics. If we shift the depicted interval to the left, we could only achieve shorter intervals (an item is gained on the left, two or more items would be lost on the right) . Function for overcoming this situation have been added. If such rare cases happen, the intervals will be properly extended.

5) (Optionally) Merging intervals, if we want do that at all. If there are touching intervals whose mean values do not differ more than 1 knots, we can merge these intervals using *mergeSimpleIntervals* function/algorithm.

**extractSteadyHeadings**

1) Finding preliminary steady-course intervals using *fifo\_maxdev* algorithm. Analogous to the algorithm extractSteadySpeeds the resulting intervals are just used for estimating the parameters that will be used in the sieve-algorithm. We find *mm2* as the maximal statistical-dispersion of all good intervals. The corresponding *ff* is a number of totes inside that interval.

2) Finding steady-course intervals using the sieve-algorithm.

3) Adjusting the resulting intervals (*adjustDevTouchingIntervals* function). At this point, it is possible to have two neighboring intervals which are both good but not perfect. For example, if we have two touching intervals 100-140 and 140-200, it might be that the more optimal combination would be 100-150 and 150-200. The used criterion for finding the most optimal combination is a classical statistical criterion - minimal sum of squares of deviations. The results after applying *adjustDevTouchingIntervals* function, can be considered optimal.

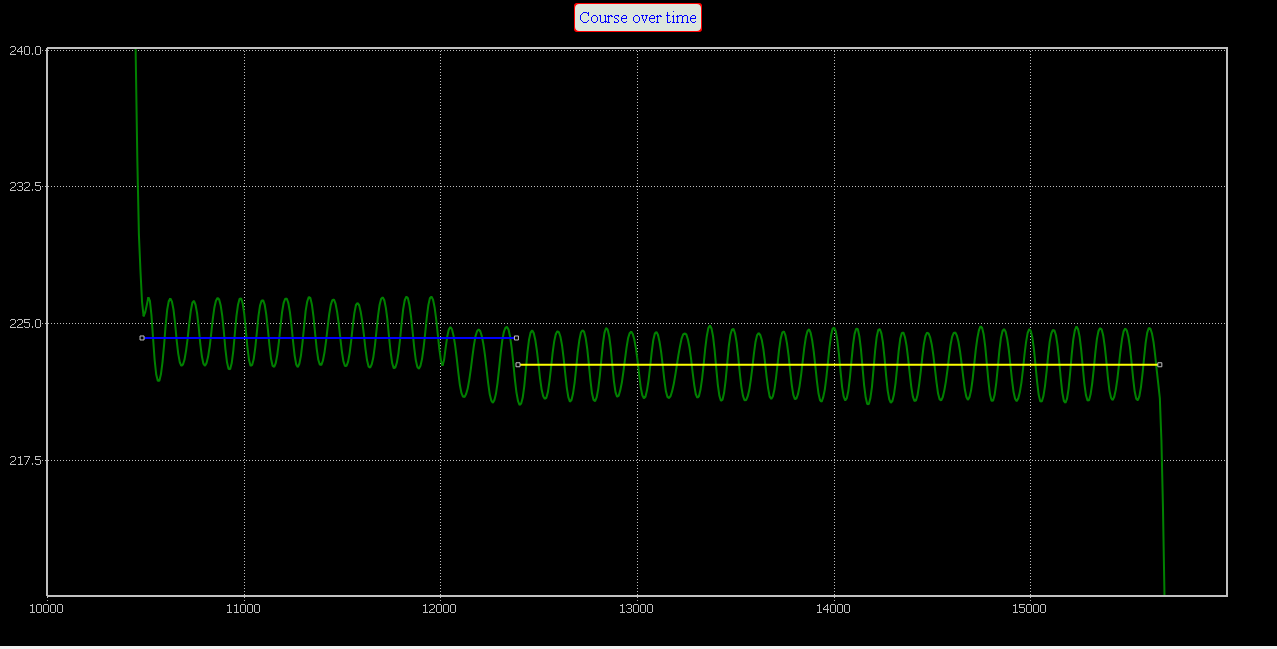


Figure - before AdjustDevTouchingIntervals

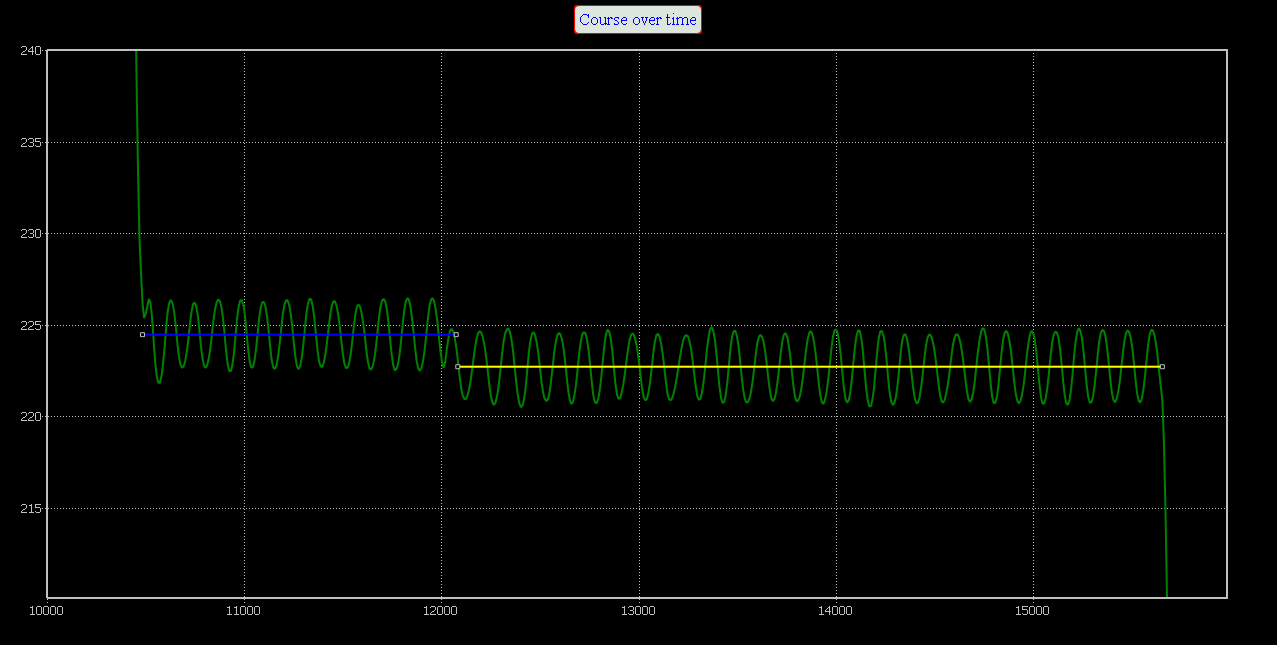


Figure - after AdjustDevTouchingIntervals

4) Extending the resulting intervals. In rare (but possible) cases, it's possible that sections (on the left or right side) which obviously belong to the steady interval are omitted. An example is: github.com/debrief/LegAlgorithms/blob/master/test\_results/problem.png. This example looks like a paradox but it is simply so from the aspect of mathematical statistics. If we shift the depicted interval to the left, we could only achieve shorter intervals (an item is gained on the left, two or more items would be lost on the right) . Function for overcoming this situation have been added. If such rare cases happen, the intervals will be properly extended.

5) (Optionally) Merging intervals, if we want do that at all. If there are touching intervals whose mean values do not differ more than 5 degrees, we can merge these intervals using mergeSimpleIntervals function/algorithm.

**fifo\_maxdev**

This functions is used for finding preliminary steady-intervals. The procedure is as follows:

1) The iterating goes from left to right i.e. the iterations start at the tote[0]. The end of the first examined intervals is determined having in mind the request that the elapsed time must be greater than *mintime* parameter (which is 5min at the moment).

2) If the range (maxvalue - minvalue) is less than predefined very small value (0.5 degrees for course; 0.1 knots for speed) we can consider the interval as steady.

3) Otherwise, if the maximal deviation does not exceed the statistically allowed limits, we can consider the interval as steady. The statistical test which is used Student's test:

- the value (heading or speed) which maximally differs from the mean value

- the mean value

- the estimated value of standard deviation

- number of totes in interval

- quantile of Student's distribution with 0.05 significance level and n-2 degrees of freedom

4) If the interval is steady, it must be additionally tested using the regression analysis. The regression analysis test must confirm that the line is horizontal.

5) If the interval is steady, the end index of the interval will be increased and new (increased) interval will be examined. Basically, we are looking for steady-interval with maximal length.

6) If there is no good interval beginning at the start-index, the start-index will be increased and the procedure will be repeated.

**Regression Analysis**

The goal is to determine the parameters of linear regression and to test statistically whether the line is horizontal (i.e. whether the interval can be considered steady). The mathematical model is: . The parameters of linear regression are estimated using the matrix formula:

Where:

The estimated corrections can be calculated as:

And, the standard deviation of unit's weight is:

The covariance matrix will be:

Accordingly, the standard deviations of estimated parameters can be calculated as:

The first standard deviation (ma) is particularly important. It is used for testing whether the regression line is horizontal using the Student's test:

**Sieve Algorithm**

This algorithm is used for finding 'almost optimal' intervals of steady headings or steady speeds. The results do not depend upon the direction of iterating and they are optimal concerning the coverage of input data. The applied statistical tests are very similar to those in *fifo\_stdev* function, but the way of iterating is completely different. Instead of "**f**irst-**i**n-**f**irst-**o**ut tote" iterations, we have applied "from the bigger to the smaller interval" principle. So, we have the following procedure within the *sieve-algorithm*:

1) The iterating starts form the interval of maximal length i.e. from the entire input interval, where all the totes are included. Number of elements in calculation is the number of all totes in the input data (let's say *N*).

2) If we came to the interval-length of *k*, we examine (by doing statistical tests) every possible interval of that length. So, we start with [1,*k*] and then go on with [2, *k*+1], [3, *k*+2], [4, *k*+3] until [*N*-*k*+1, *N*].

3) The statistical tests will be performed in the following way:

3a - first we are checking are all the deviations in the allowed limits

3b - If the interval is good, it has to be additionally tested using the regression analysis. The regression analysis test must confirm that the line is horizontal

4) If none of examined intervals pass statistical test we shall repeat the entire procedure with the interval-length of *N-1*.

5) If there are several good intervals, we shall put the best one into the list. The best one is chosen using the standard deviation criterion. In other words, the good interval with minimal standard deviation will be chosen.

6) The sieve algorithm is recursive. When a good interval is found and chosen, the initial interval will be split and the sieve algorithm will be performed again for both left and right subintervals. For example if an interval [*i*, *j*] has been chosen, we will continue with sieve([1, *i*-1]) and sieve([*j*+1, *N*])

7) Before examining an interval it is first verified that the elapsed time is greater than *mintime* parameter (which is 5min at the moment).

**AdjustDevTouchingIntervals**

It is possible sometimes to have two touching intervals which are both good but not perfect. For example, if we have two touching intervals 100-140 and 140-200, it might be that the more optimal combination would be 100-150 and 150-200. The used criterion for finding the most optimal combination is a classical statistical criterion - minimal sum of squares of deviations. The results after applying this function, can be considered optimal.

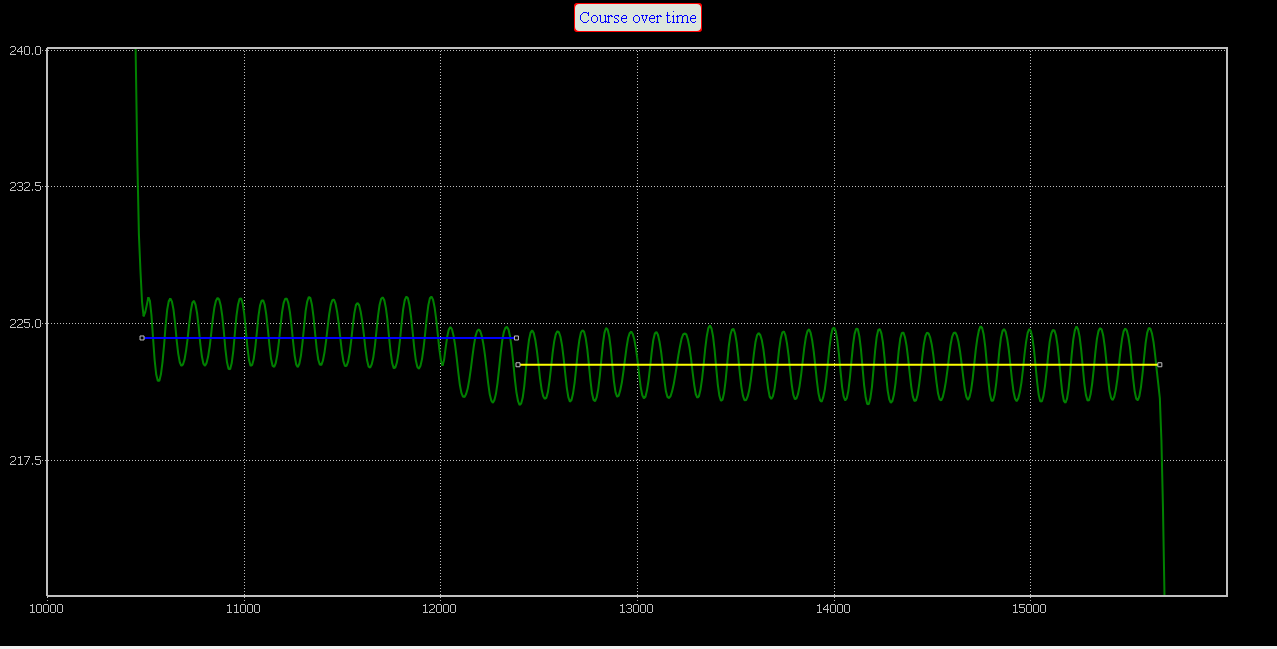


Figure 3 - before AdjustDevTouchingIntervals

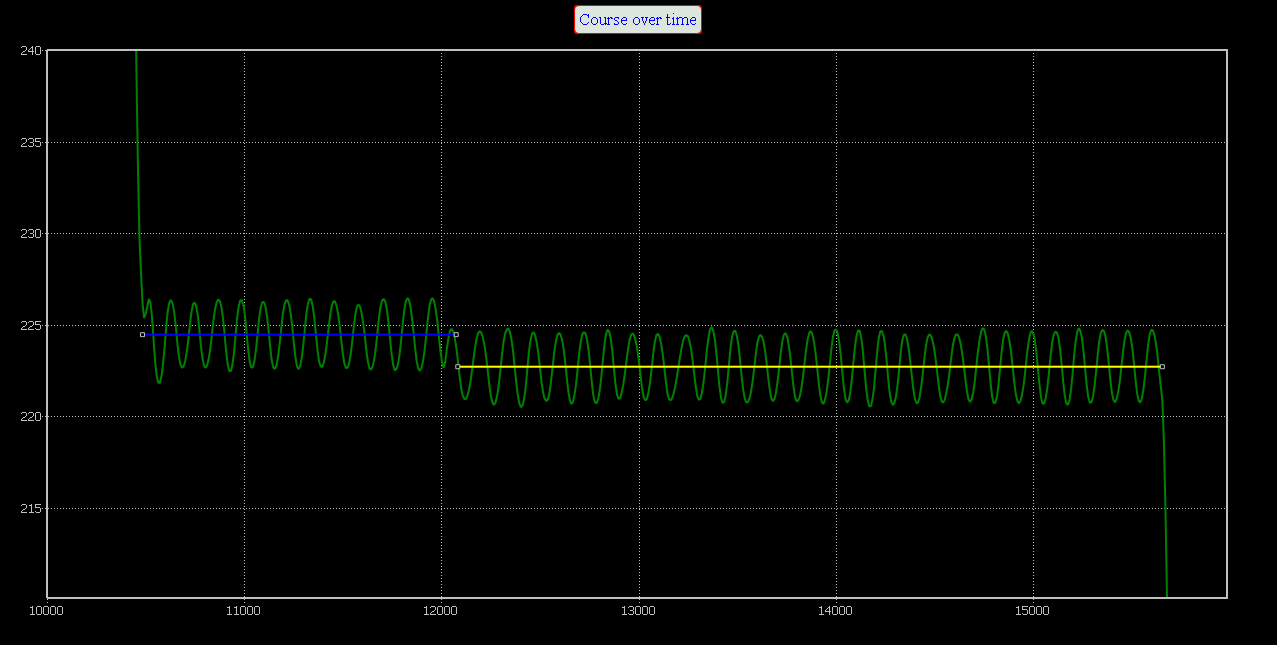


Figure 4 - after AdjustDevTouchingIntervals

Adjust touching intervals function/algorithm is called after sieve-algorithm i.e after having a list of good (but non-optimized) intervals. The basic principle again is - the solution with minimal standard deviation is the best one. However, we don't have to calculate standard deviations. It is enough just to calculate and compare inter-results - the sum of squares of deviations.

So, we have the following procedure with this algorithm:

1) We are iterating through the list of good intervals. Whenever we find two neighbouring intervals to be *touching*, in the same time - we are performing optimization.

2) The so called *LeftMovingEdge* point is initially determined. It is the closest point to the beginning of the left interval which satisfied the condition of minimal elapsed time. For example, if the touching intervals are [*i*, *j*] and [*j*, *k*], we are searching for the *x*-point belonging to the left interval [*i*, *j*] which satisfies conditions: time(*x*)-time(*i*) >= *mintime* and time(*x*-1)-time(*i*) < *mintime*. Practically, we start iterating from i+1, i+2,... The first point *x* which satisfies the condition time(*x*)-time(*i*) >= *mintime* is the initially determined *LeftMovingEdge*.

3) In the same manner, the so called *RightMovingEdge* point is initially determined. We are searching inside the right interval [*j*, *k*]; we start iterating backward from *k*-1, *k*-2... The first point *x* which satisfies the condition time(*k*)-time(*x*) >= *mintime* is the initially determined *RightMovingEdge*.

4) Then we are analyzing all possible touching points between *LeftMovingEdge* and *RightMovingEdge* (inclusive).

4a - we are testing first are both left and right intervals are good in regards to *areDeviationsInAllowedLimits* function.

4b - If 4a is satisfied, then both left and right intervals are tested by the regression analysis to see whether the regression lines are horizontal.

4c - if both (a) and (b) are satisfied, we can go to the 5th step. Otherwise, we are going to the next iteration (next loop; next touch-point).

5) The sum of squares of deviations is calculated.

7) The touching point (*x*) which results with minimal sum of squares of standard deviation is practically - the best touching point. The resulting intervals will be [*i*, *x*] and [*x*, *k*].

In such a way, the optimization of the touching point is performed.