# MEDD test specification

Author: D.DOBLER (IFREMER)

Date: 2019/11/20

Version: 1.2 (add precision on units of inputs)

The MEDD test is composed of three functions:

* The computation of the 2d relative distance algorithm : ***relative\_2D\_distance***
* The computation of the median and of the thresholds at distance d : ***QTRT\_Spike\_test\_MEDD***
* The main call to the computation with the **configuration** assuring 98% of robustness:

***QTRT\_Spike\_test\_MEDD\_main***

Here after are the specifications for these three functions.

## relative\_2D\_distance function

Inputs:

* **xa** and **ya** vectors (already adimensionalised). In practice, xa stands for the median in salinity or temperature or density and ya for depth or pressure,
* d distance (adimensionalised between 0 and 1)

output:

* **xmoins** and **xplus** vectors, having same size as x vector and being at, at least, distance d from all {xa,ya} points

### STEP 1: Curviline interpolation

To ensure there is less than the distance d between two points of the curve, the first step is to execute an interpolation.

*Initialisations*

Save original vectors before interpolation:

For all {xa,ya} points except the first one :

1.i. Test verticality and compute the slopes a(n) of the segments

1.ii. Compute the intercepts b(n) of the segments

1.iii. Compute the segment length:

1.iv. Compute the number of points to add that ensures a maximum distance d between segments (with a margin)

1.v. Compute the corresponding coordinates

The arrays xi\_n and yi\_n are of length n\_toadd and for the simplicity of writing, I have written them in a matlab fashion.

1.vi. Add them to the list of points to add

vii. Reconstruct xa, ya with the interpolated points

y

sort ya vector

apply the sort order to xa as well

Now that we have segments with less than distance d between points, we can compute the points at distance d from these points.

### STEP 2: Compute thresholds at distance d

*Initialisations*

Construct xbplus and xbmoins vectors of size xa\_ori.

*Computation*

For all points or in other words:

For all levels, find at which {xbm,xbp} values, the circles of radius **d** and centered on each crosses the horizontal line . The final two {xbmoins,xbplus} values that are at least at distance d from all {xa,ya} points are the minimum and the maximum of all the found {xbm,xbp} values.

for m=1 to the length of ya

if then (the circle can cross the line )

end

end of loop on m

## QTRT\_spike\_check\_medd function

For the simplicity of writing, many steps here will be written in a matrix fashion.

Inputs

* **PARAM** = 'DENS' or 'TEMP' or 'PSAL'. This is used for specific limits and for the nondimensionalization of the axes in the step that computes the distance d to median curve.
* **PROF** is an array with the values of the vertical profile to test (could be PSAL, TEMP, or their ADJUSTED counterpart)
* **PRES** is an array with the corresponding pressures in dbar for vertical location
* defines 4 vertical zones [Z1; Z2; Z3 ; Z4] used to customize sliding windows. e.g: =[150;500;1000;1500]
* is an array of the same size as Z\_levels used to define the sliding window for median computation in [dbar], depending on vertical zones. For example =[110; 180; 400; 500] => +/- 110 dbar above Z1; +/- 110 dbar at Z1; +/- 180 dbar at Z2; +/- 400 dbar at Z3; +/- 500 dbar at Z4; +/- 500 dbar below Z4

then the sliding windows are **linearly interpolated** on segment [Z1, Z2], [Z2, Z3] and [Z3, Z4].

* is an array of 5 values used to define the Z levels used for vertical derivative thresholds definition (dpdz\_thr and ddpdz\_thr)
* is an array of 6 values used to define the maximum allowed vertical derivative values for the parameter PARAM
* is an array of 6 values used to define the maximum allowed vertical derivative values when the sign changes for the parameter PARAM
* is an array of 4 values defining the relative 2D distance to median curve as a percentage of the windows. The values change on the vertical with limits defined by array.

N.B.: There are 6 parameters more in the function that are not set as inputs of the function. They are not prone to change and to prevent from having a very long list on inputs, I have kept them as constants inside the code itself (step 5.ii (2 parameters), 6.ii.g (1), 6.ii.h (1) and 7.iii (2)). But this could be changed. I have highlighted them in yellow in the specification.

Outputs

* **is\_spike** is an array of the same size as PARAM and PRES arrays. It is set to one when PROF is out of [medm medp] window
* is the sliding median of the profile with many customizations
* and are computed as a 2D relative distance d from median
* **BO** equal 1 if the number of finite points is less or equal 5

### STEP 1. Test number of levels with finite values for PROF

; e

### STEP 2. Compute vertical derivative of PARAM

if the two consecutive points are too close to each other, then set the vertical derivative to zero, else, compute it.

Then duplicate the derivative of the second point into the first point:

### STEP 3. Compute sliding windows

Note here that it was chosen to construct such things inside this function but this also could have been done in the calling procedure and would have been defined there and mentioned as an input parameter of this function

### STEP 4. Compute arrays of parameters and

Same remark as above.

### STEP 5. Record doubtful points to discard them in the median computation

Initialize an array having the same length as array and with 0 as default value.

5.i. First discard large vertical derivative with a change of sign (thresholds)

Nan values are not considered (which gives this particular piece of code to handle), and supplementary arrays are built so that index n relates to the level below or above as follows:

Let be the indices where PROF and PRES are not NaN (already computed during STEP 1)

then, we can construct:

Let’s do the same on the maximum values allowed for derivative when there is a change in sign below and above (built from input parameter)

Discard points for which

And declare for these points

This can be done as follows:

At this level correspond to indices inside array, original levels have to be retrieved:

Finally:

Duplicate the second index into the first index (the same as is done for dpdz parameter):

5.ii. Discard fast increase in temperature or decrease in density

Most commonly, on the levels that are at stake, temperature generally decreases (except at high latitudes) and density increases. Thus troublesome points where the contrary is seen are not taken into account in the median computation:

N.B.: As already mentioned in the input part of the description of this function, I have not put the corresponding thresholds of 0.5 °C/dbar and 0.001 kg/m3/dbar as inputs of the function but this can be discussed. Note also that these points are only not taken into account in the median computation, they are not directly flagged as potential spikes.

5.iii. Discard large vertical derivative (thresholds)

**=> Here we are, in the variable are gathered all points to discard when computing the sliding median**

### STEP 6. Compute sliding median

6.i. Initialization of a last correct index array

This begins with a trick to record the correct index just above for points. This will be used later to assign a median value for points.

Before looping on z levels, this array is initialized, taken into account that first points can already be “out”.

At the end of the treatment on each level, this array will be updated.

First record which levels can be considered as “correct”

Then Initialize *array* with the same size as PROF array and the default value .

6.ii. Loop on all z-levels where and

6.ii.a Select coherent indices to compute median

The indices should be inside SW/2 distance and selected among . The indices are sorted to secure next steps.

) +1

6.ii.b Keep same number of points below and above

This step is important. This allows the median to be as close as possible of the current value if profile shape is smooth. If possible (i.e. there at least one point on each side), the same number of points are kept above and below. Another step handles the case when there is no point below or no point above.

N.B.: special warning, “below” and “above” refer to vertical sense: below is closest to bottom and above is closest to the surface. The corresponding indices are reversed as profile is read for surface to bottom: thus are the smallest indices and are the largest indices.

6.ii.c Treat special case when there is finally no more coherent values

6.ii.d Keep the number of values uneven for the median computation

With an even number of points, the median computed is the average of the two central values. This is unwanted, so the first value is duplicated to make it uneven artificially. In other words, the median is forced to equal the first of the two central values.

**6.ii.e Compute Median**

6.ii.f Adjust median when there is no points or just one point below in

This step handles the first points when a proper median cannot be estimated as there is no or almost no coherent point just below. This is the case for bottom point for instance. The baseline then is to check that the vertical derivative is in or out maximum value allowed for the current level and the level just above. If it is inside, then the median will be forced to be set equal to the value itself.

6.ii.g Adjust median when there is no points or just one point above in

The same is done for levels where there is no or almost no coherent point just above. This is the case for surface points for instance. There is a supplementary precaution taken on dpdz maximum value to apply this step on temperature profiles.

= 0

6.ii.h Adjust median for deep pressure values

The purpose of this step is to trap spiky points for hedgehog profiles. This should be trapped by Pressure increasing test normally. It has been checked that it did not degrade for deep Argo but definitely, there is a need to better define for pressure values deeper than 2000 dbar.

First record the level of last acceptable computed median above 1950 dbar and compute the gradient between current and levels above 1950 dbar. Two arrays of the same size as PROF should be initialized with NaN values for and zeros for .

If this is too steep below 1950 dbar: replace by the last acceptable median computed above 1950dbar.

6.ii.i Record last correct value to adjust median for levels

This last step records for the current level what the previous correct level is when the current level is part of . The aim is to apply the median computed for the last point just above that was considered as correct.

6.iii. Finalize Median computation

First set to profile value the median for eligible levels (steps 6.ii.f , 6.ii.g and 6.ii.h):

Second, set to last correct median for levels (step 5, step 6.i and step 6.ii.i)

Now we have a array that is as smooth as possible, in the way that was defined by the algorithm. The following step computes the thresholds to this median.

### STEP 7. Compute thresholds using 2D relative distance to median

7.i. Retrieve indices for the vertical zones

Four vertical zones are defined with a distinct d distance each:

7.ii Initialize arrays

Final thresholds are 2 arrays and of the same size as and filled by default with NaN values.

Intermediate thresholds are 8 arrays and with j going from 1 to 4, of the same size as and filled by default with NaN values.

7.iii. Nondimensionalization of axes

The z-axis was first nondimensionalized by a fixed maximal extension of 2000 dbar but it was visually more adapted with a smaller value. Finally 700 dbar was kept:

The x-axis is nondimensionalized by the extension of the x-values but this extension is bounded to avoid too large or too small windows.

Here, I’ve made something not straight to get directly the output of 2D relative distance function in the dimension of PROF. Instead of nondimensionalizing PROF and PRES, apply the function on them and put dimension back the output, I have chosen to nondimensionalize PRES and dimensionalize d.

Here is the proof:

with  **and**

7.iv - Call relative 2D distance function

Then call the first function relative\_2D\_distance for each with j = 1 to 4:

### STEP 8. Construct is\_spike output

## Configuration and main call (function QTRT\_spike\_check\_MEDD\_main)

Inputs

* **PRES:** pressure values in [dbar]
* **TEMP:** in-situtemperature values [°C - ITS90]
* **PSAL:** practical salinity values [PSS-78] (should be NaN(size(PRES)) if not available)
* **DENS:** potential density values [kg/m3] (should be NaN(size(PRES)) if not available)
* **LAT:** latitude of the profiles in [degrees]

Outputs

* **SPIKE\_T:** array of size(PRES) with a 1 on temperature spikes, It is set to NaN if the computation could not be made
* **SPIKE\_S:** array of size(PRES) with a 1 on salinity spikes, It is set to NaN if the computation could not be made
* **BO\_T:** breakout on temperature evaluation, set to 1 if the number of finite measures is less than 5, set to 0 otherwise
* **BO\_S:** breakout on salinity evaluation, set to 1 if the number of finite measures is less than 5, set to 0 otherwise
* The other outputs **PROF\_med, PROF\_medm and PROF\_medp with PROF = {'PSAL','TEMP','DENS'}** that are present in the function on thegithub repository are only used for display purpose for robustness tests and can be skipped

This last part is dedicated to the configuration, the call of the QTRT\_spike\_check\_MEDD function on each physical parameter and the construction of the final Spike estimation.

Some input parameters are common to all variables (TEMP, PSAL, DENS)



The other parameters depend on the physical variable:

For ‘TEMP’:

* if density is available
* if density is not available and abs(LAT) > 10
* if density is not available and abs(LAT) < 10

For ‘PSAL’



For ‘DENS’



Apply QTRT\_spike\_check\_medd function to TEMP, and if possible to PSAL and DENS profiles :

Finally, it is considered as spiky if there is a spike on PSAL or TEMP profile and (there is a spike on DENS profile or DENS could not be computed)

N.B.: This coupling between density and profile spikes was chosen to ensure a high robustness of the method to run it in automatic but this could also be applied unitary as well if not run in automatic.

## Reference papers and script

Here is where I find some help to construct my ideas:

* R.Wedd, M.Springer & H. Haines (2015) *Argo real-time quality control intercomparison, Journal of operational Oceanography*, 8:2, 108-122,DOI: <https://doi.org/10.1080/1755876X.2015.108186>
* G.P. Castelão (2016) *A flexible system for automatic quality control of oceanographic data.* <https://arxiv.org/abs/1503.02714v2>
* Routine STD\_spike\_check.m from ISAS v6.2 (F.Gaillard and R.Charraudeau)