DATA PRODUCT SPECIFICATION FOR GRADIENT TEST

Version 1-00
Document Control Number 1341-10010
2012-07-17

Consortium for Ocean Leadership
1201 New York Ave NW, 4th Floor, Washington DC 20005
www.OceanLeadership.org

in Cooperation with

University of California, San Diego
University of Washington
Woods Hole Oceanographic Institution
Oregon State University
Scripps Institution of Oceanography
Rutgers University

Document Control Sheet

Version	Date	Description	Author
0-01	2012-04-27	Initial draft	M. Lankhorst
0-02	2012-05-01	Complete example tables and code	M. Lankhorst
0-03	2012-06-13	Addressed comments from focused review	M. Lankhorst
0-04	2012-07-06	Addressed comments from formal review	M. Lankhorst
1-00	2012-07-17	Initial Release	E. Chapman

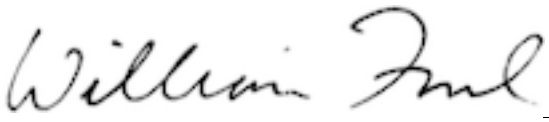
Signature Page

This document has been reviewed and approved for release to Configuration Management.

OOI Chief Systems Engineer: 

Date:2012-07-17

This document has been reviewed and meets the needs of the OOI Cyberinfrastructure for the purpose of coding and implementation.

OOI CI Signing Authority: 

Date: 2012-07-17

Table of Contents

1 Abstract 1
2 Introduction 1
2.1 Author Contact Information 1
2.2 Metadata Information 1
2.3 Instruments 2
2.4 Literature and Reference Documents 2
2.5 Terminology 2
3 Theory 2
3.1 Description 2
3.2 Mathematical Theory 3
3.3 Known Theoretical Limitations 3
3.4 Revision History 3
4 Implementation 3
4.1 Overview 3
4.2 Inputs 3
4.3 Processing Flow 3
4.4 Outputs 4
4.5 Computational and Numerical Considerations 4
4.6 Code Verification and Test Data Sets 5
Appendix A Example Code 1

1 Abstract

This document describes the OOI Gradient Test, which is an automated quality control algorithm used on various OOI data products. This automated algorithm generates flags for data points according to whether changes between successive points are within a pre-determined range.

2 Introduction

2.1 Author Contact Information

Please contact Matthias Lankhorst (mlankhorst@ucsd.edu) or the Data Product Specification lead (DPS@lists.oceanobservatories.org) for more information concerning the algorithm and other items in this document.

2.2 Metadata Information

2.2.1 Data Product Name

The parameter name for the quantity resulting from this computation is composed of the OOI Core Data Product Name that is being described, with "_GRADTST_QC" attached:

- [*DATAPRODUCT*]_GRADTST_QC

The descriptive name is:

- Gradient Test QC Flags

2.2.2 Data Product Abstract (for Metadata)

Quality control flags from gradient test.

2.2.3 Computation Name

The name for this quality control algorithm is: GRADTST

2.2.4 Computation Abstract (for Metadata)

The OOI Gradient Test quality control algorithm generates QC flags indicating whether changes between successive data points fall within a given range.

2.2.5 Instrument-Specific Metadata

n/a

2.2.6 Synonyms

n/a

2.2.7 Similar Algorithms

The OOI Spike Test (SPKETST, OOI document no. 1341-10006) algorithm identifies individual outlier points among a small group of data points, which is equivalent to an excessive gradient between the outlier and the surrounding values. However, while the spike test will only detect errors if they consist of single points, the gradient test will detect if multiple successive points are remote from a baseline of presumably good data points.
The OOI Trend Test (TRNDTST, OOI document no. 1341-10007) algorithm identifies if a data set as a whole, consisting of many data points, contains a trend. In contrast, the gradient test examines changes between individual data points.

2.3 Instruments

This algorithm is applied to OOI data products as per the separate processing flowcharts documents. The algorithm itself is not instrument-specific.

2.4 Literature and Reference Documents

DCN 1342-000xx **Instrument-specific Processing Flow documents** contain flow diagrams detailing all of the specific algorithms (product, QA and calibration, QC) necessary to compute all data products from the instrument at all levels of QA and QC and the order that the algorithms must be applied

2.5 Terminology

2.5.1 Definitions

n/a

2.5.2 Acronyms, Abbreviations and Notations

General OOI acronyms, abbreviations and notations are contained in the Level 2 Reference Module in the OOI requirements database (DOORS). There are no other acronyms, abbreviations, or notations for this document.

2.5.3 Variables and Symbols

DAT	input data (vector of numeric values)
X	input coordinate axis on which DAT is given (vector of numeric values, strictly increasing)
DDATDX	valid range of gradient (two-element vector)
MINDX	minimum threshold separation of successive X values (scalar)
STARTDAT	alternate start value for DAT in case the first data value is not considered good
TOLDAT	tolerance in DAT for returning from bad to good data, see algorithm description below
N	the number of points in DAT (and X)
OUTDAT	a copy of DAT that contains only the values that were actually considered by the algorithm
OUTX	a copy of X that contains only the values that were actually considered by the algorithm
OUTQC	output QC flags denoting the quality of OUTDAT (0 bad, 1 good) as judged by this algorithm

3 Theory

3.1 Description

The algorithm scans the data points in DAT successively, one-by-one. Starting from a data point assumed to be good, if the change from this point to the next, divided by the respective advance in X, is within the range defined by DDATDX, the next point will be assumed good, else bad. Once a data point is identified as bad, successive data points will be considered bad until a data point falls within TOLDAT of the last known good data point.

As a default starting point, the first data point is considered good. This default can be overridden by defining STARTDAT, in which case the first data point that falls within TOLDAT of STARTDAT will be considered good (and potentially earlier ones as bad).

X must be strictly increasing. In order to avoid small X steps, which can lead to exaggerated gradients because the X difference is used in the denominator, setting MINDX to a value greater than zero is an option to remove all data points DAT (and X) for which X is separated by MINDX or less.

3.2 Mathematical Theory

[included in description and exemplar code]

3.3 Known Theoretical Limitations

From an excessive gradient between two neighboring data points alone, it is not obvious which of the two points is wrong. This ambiguity needs to be addressed either by using STARTDAT to force the algorithm towards an absolute, pre-determined value, or by relying on the default assumption that the first value is always good.

If data points are obtained very closely together in X (meaning X differences are small), the resulting gradients naturally can become very large due to noise in DAT and the small difference in X being in the denominator. This can preclude the usefulness of the algorithm. To avoid flagging random noise on small time scales as bad, MINDAT can be used to remove such points.

Once an excessive gradient has triggered data points to be flagged as bad, one could imagine different criteria for later values returning to good. The algorithm here requires that the later data return to within TOLDAT of the last known good value. However, this approach might not be appropriate for all situations: e.g. if there is a natural slow trend during the episode of bad data points, a return to good values might not occur. Such situations might lead to erroneous flagging of the output data.

The MINDAT option of removing data points that are too close together can lead to some data points not being analyzed at all, hence upsetting the one-to-one correspondence of input data DAT and availability of output values OUTQC.

3.4 Revision History

n/a

4 Implementation

4.1 Overview

The implementation in the exemplar code follows the “Description” text: based on inputs STARTDAT and TOLDAT, it is determined whether the first data point is good or bad. Then, successive points are scanned one-by-one as outlined above.

4.2 Inputs

- DAT, X: the input data set
- DDATDX, MINDAT, STARTDAT, TOLDAT: from lookup table

4.3 Processing Flow

For general information, see exemplar code given in Appendix A.

Case-specific pre-processing: Some use cases require the input data DAT (and corresponding X) to be assembled from multiple data sets (e.g. density from multiple CTDMO instruments on a single mooring) or from specific subsets of data (e.g. one single profile from a mooring profiler or glider). This is called out in the lookup table for the input parameters.

Post-processing: In cases where OUTDAT and OUTX are not the same as DAT and X, the missing values in OUTQC need to be filled with dummies.

4.4 Outputs

- OUTQC: Quality control flags. Note: If the computation did not evaluate all input data DAT (i.e. if OUTDAT and DAT are different), an additional step is needed to fill the missing values in OUTQC with dummies. This additional step is not included in the example code below.

The metadata that must be included with the output are

- Input parameters (or links to where they were obtained)
- An identifier/link that relates the output flags to the data product that the flags are meant to describe

4.5 Computational and Numerical Considerations

n/a

4.6 Code Verification and Test Data Sets

The algorithm code will be verified using the test data set provided, which contains inputs and their associated correct outputs. CI will verify that the algorithm code is correct by checking that the algorithm output, generated using the test data inputs, is identical to the test data output.

Table 1: Test Data Sets

Test Case 1:	
DAT	[3 5 98 99 4]
X	[1 2 3 4 5]
DDATDX	[-50 50]
MINDX	[]
STARTDAT	[]
TOLDAT	5
OUTDAT	[3 5 98 99 4]
OUTX	[1 2 3 4 5]
OUTQC	[1 1 0 0 1]
Test Case 2:	
DAT	[3 5 98 99 4]
X	[1 2 3 4 5]
DDATDX	[-50 50]
MINDX	[]
STARTDAT	100
TOLDAT	5
OUTDAT	[3 5 98 99 4]
OUTX	[1 2 3 4 5]
OUTQC	[0 0 1 1 0]
Test Case 3:	
DAT	[3 5 98 99 4]
X	[1 2 3 3.1 4]
DDATDX	[-50 50]
MINDX	0.2
STARTDAT	[]
TOLDAT	5
OUTDAT	[3 5 98 4]
OUTX	[1 2 3 4]
OUTQC	[1 1 0 1]

Table 2: Example Lookup Table

Data product used as input parameter DAT	Data Product used as input parameter X	Units of DAT	Units of X	DDATDX	MINDX	STARTDAT	TOLDAT
Temperature of sea water TEMPWAT from CTDMO instruments	Elapsed time	°C	s	[-0.01 0.01]	30	<empty>	0.1
Density of sea water from glider (extract one single dive)	Pressure	kg m^{-3}	dbar	[0 0.5]	0.1	<empty>	0.1

Note that this table is for example purposes only and some/all values may not be correct. The official QC lookup tables are kept separately from the DPS and, at the time of writing, do not exist in their final form.

Appendix A Example Code

The following routine is example code run under MatLab:

```
% DATAQC_GRADIENTTEST   Data quality control algorithm testing if
%    changes between successive data points fall within a certain
%    range.
%
% Time-stamp: <2012-04-26 13:47:12 mlankhorst>
%
% Input data DAT are given as a function of coordinate X. The
% algorithm will flag DAT values as bad if the change
% deltaDAT/deltaX between successive DAT values exceeds thresholds
% given in DDATDX. Once the threshold is exceeded, following DAT
% are considered bad until a DAT value returns to within TOLDAT of
% the last known good value.
%
% It is possible to remove data points that are too close together
% in X coordinates (use MINDX).
%
% By default, the first value of DAT is considered good. To change
% this, use STARTDAT and TOLDAT to set as the first good data point
% the first one that comes within TOLDAT of STARTDAT.
%
% USAGE: [OUTDAT,OUTX,OUTQC]= ...
%            dataqc_gradienttest(DAT,X,DDATDX,MINDX,STARTDAT,TOLDAT);
%
%    DAT: Input dataset, a numeric real vector.
%    X:   Coordinate (e.g. time, distance) along which DAT is
%         given. Must be of the same size as DAT and strictly
%         increasing.
%    DDATDX: Two-element vector defining the valid range of
%         deltaDAT/deltaX from one point to the next.
%    MINDX: Scalar. Minimum deltaX for which this test will
%         be applied (data that are less than MINDX apart will be
%         deleted). Defaults to zero if NaN/empty.
%    STARTDAT: Start value (scalar) of DAT that is presumed
%         good. Defaults to first non-NaN value of DAT if NaN/empty.
%    TOLDAT: Tolerance value (scalar) for DAT; threshold to within
%         which DAT must return to be counted as good, after
%         exceeding a DDATDX threshold detected bad data.
%
%    OUTDAT: Same as DAT except that NaNs and values not meeting
%         MINDX are removed.
%    OUTX: Same as X except that NaNs and values not meeting
%         MINDX are removed.
%    OUTQC: Output quality control flags for OUTDAT. 0 means bad
%         data, 1 means good data.
%
%
% EXAMPLES:
%
% Ordinary use, default MINDX and STARTDAT:
%
% [outdat,outx,outqc]= ...
%      dataqc_gradienttest([3 5 98 99 4],[1:5],[-50 50],[],[],5)
%   outdat =      3     5    98    99     4
%   outx   =      1     2     3     4     5
%   outqc  =      1     1     0     0     1
%
%
```

```
% Alternate STARTDAT to swap good/bad segments:
%
%   [outdat,outx,outqc]= ...
%      dataqc_gradienttest([3 5 98 99 4],[1:5],[-50 50],[],100,5)
%   outdat =      3     5    98    99     4
%   outx   =      1     2     3     4     5
%   outqc  =      0     0     1     1     0
%
%
% Alternate MINDX to remove certain X and DAT:
%
%   [outdat,outx,outqc]= ...
%      dataqc_gradienttest([3 5 98 99 4],[1 2 3 3.1 4], ...
%                          [-50 50],0.2,[],5)
%   outdat =      3     5    98     4
%   outx   =      1     2     3     4
%   outqc  =      1     1     0     1
%

function [outdat,outx,outqc]= ...
      dataqc_gradienttest(dat,x,ddatdx,mindx,startdat,toldat);


  % Sanity checks on DAT and X:

  if ((~isvector(dat))|(~isvector(x)))
    error('DAT and X must be vectors.')
  end
  if (length(dat))~=(length(x))
    error('DAT and X must be of equal length.')
  end
  if ~all((diff(x))>0)
    error('X must be strictly monotonically increasing.')
  end

  ff=find((~isnan(dat))&(~isnan(x)));
  dat=dat(ff);
  x=x(ff);

  dat=dat(:)';
  x=x(:)';


  % Check & set MINDX

  if isempty(mindx)
    mindx=nan;
  end
  if isnan(mindx)
    mindx=0;
  end
  if ~isscalar(mindx)
    error('MINDX must be scalar, NaN, or empty.')
  end


  % Apply MINDX

  dx=diff(x);
  ff=find(dx>mindx);
  gg=[1 ff+1];
  dat=dat(gg);
  x=x(gg);
```

```
% Confirm that there are still data points left, else abort:

outqc=zeros(size(dat));
ll=length(dat);
if ll<=1
  warning(['DAT and X contain too few points for meaningful' ...
          ' analysis.'])
  outdat=dat;
  outx=x;
  return;
end


% Check & set STARTDAT, including output for data point 1:

if isempty(startdat)
  startdat=nan;
end
if isnan(startdat)
  startdat=dat(1);
  outqc(1)=1;
else
  if abs(startdat-dat(1))<=toldat
    startdat=dat(1);
    outqc(1)=1;
  else
    outqc(1)=0;
  end
end
if ~isscalar(startdat)
  error('STARTDAT must be scalar, NaN, or empty.')
end



% Main loop, checking for data points 2 through ll:

ii=2;

while (ii<=ll)

  if outqc(ii-1)==0

    if abs(dat(ii)-startdat)<=toldat
     outqc(ii)=1;
     startdat=dat(ii);
    else
     outqc(ii)=0;
    end

  else

    tmp=(dat(ii)-dat(ii-1))/(x(ii)-x(ii-1));
    if (tmp<ddatdx(1))|(tmp>ddatdx(2))
     outqc(ii)=0;
    else
     outqc(ii)=1;
     startdat=dat(ii);
    end

  end
```

```
  ii=ii+1;
end

outqc=logical(outqc);
outdat=dat;
outx=x;
```