DATA PRODUCT SPECIFICATION FOR STUCK VALUE TEST

Version 1-01
Document Control Number 1341-10008
2012-05-23

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	2011-12-21	Initial draft	M. Lankhorst, S.H. Nam
0-02	2012-02-16	Modified to match DPS Outline.	S. Webster
0-03	2012-03-28	Added info about appropriate metadata to Output and info about irregularly spaced data to Known Limitations.	S. Webster
0-04	2012-04-16	Incorporated comments from focused review.	M. Lankhorst
0-05	2012-04-20	Updated exemplar QC look-up table and moved to Test Data section 4.6	S. Webster
1-00	2012-05-22	Initial Release	E. Chapman
1-01	2012-05-23	Formatting, copy edits	E. Griffin

Signature Page

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

OOI Chief Systems Engineer: 

Date:2012-05-22

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: 2012-05-22

Table of Contents

1 Abstract

This document describes the OOI Stuck Value Test (STUCKVL) quality control algorithm used to test time series for “stuck values,” i.e., repeated occurrences of one value. The purpose of this document is to serve as a reference in order to document which processing steps have been applied to a data product.

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

n/a

2.2.2 Data Product Abstract (for Metadata)

n/a

2.2.3 Computation Name

The name for this quality control algorithm is

- Stuck Value Test (STUCKVL)

2.2.4 Computation Abstract (for Metadata)

The OOI Stuck Value Test quality control algorithm generates a flag for repeated occurrence of one value in a time series.

2.2.5 Instrument-Specific Metadata

n/a

2.2.6 Synonyms

n/a

2.2.7 Similar Computations

n/a

2.3 Instruments

n/a

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

Stuck values	Values repeated more than specified times with specified difference in a single time series
Resolution	Criterion of difference that repeated values have for being considered as stuck values
Minimum Number	Criterion of the number of times the values are repeated for being considered as stuck values

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

The following variables and symbols are defined here for use throughout this document.

x_in(t)	input time series vector
x_out(t)	output Boolean vector
t	time
R	resolution
N	minimum number

3 Theory

3.1 Description

Differences between neighboring values in a time series are checked whether they are less than the resolution. In case that this stuck happens more than the minimum number, the values are taken as stuck values.

3.2 Mathematical Theory

If
| x_in(t) – x_in(t+i) | < R for i = 1 to N
then
X_out(t), x_out(t+1), …, x_out(t+N) = 0 %stuck values
else
X_out(t), x_out(t+1), …, x_out(t+N) = 1 %not stuck values

3.3 Known Theoretical Limitations

The theory behind this algorithm assumes that the data are collected at constant time intervals. However, OOI will run this algorithm independent of whether this is actually the case, i.e. run over differently spaced data points if there are data gaps or changes in sampling scheme. The amount of uncertain output caused by this is expected to be negligible.

3.4 Revision History

n/a

4 Implementation

4.1 Overview

This code returns “1” for presumably good data and “0” for stuck values. It is implemented using the MatLab function “`dataqc_stuckvaluetest`”, which itself calls MatLab functions “isnumeric”, “isvector”, “isscalar”, and “isreal” to check the inputs prior to applying the algorithm.

4.2 Inputs

`x_in` input time series vector
`R` resolution, repeat values less than `R` apart will be considered "stuck values".
`N` minimum number of successive values within `R` of each other that will trigger the "stuck value". This is optional and defaults to 10 if omitted or empty.

R and N are taken from the Stuck Value look-up table.

4.3 Processing Flow

Call MatLab function “`dataqc_stuckvaluetest`”:

```
x_out = dataqc_stuckvaluetest(x_in,R,N)
```

See code in Appendix A.

4.4 Outputs

`x_out` Boolean output: 0 where stuck values are found, 1 elsewhere.
`R,N` Cf. input parameters; to be available as metadata in the output as well.

4.5 Computation 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.

Input parameters:

```
X_in = [4.83  1.40  3.33  3.33  3.33  3.33  4.09  2.97  2.85  3.67]
R = 0.001
N = 4
```

Output:

```
x_out = 1     1     0     0     0     0     1     1     1     1
```

Include R and N as output metadata.

In addition to the output (qcflag), the metadata from the QC Lookup table must be included with the output.

Table 1: Example Stuck Value Lookup Table

Data Product	Data Product Description	Resolution (R)	Number of stuck values (N) within resolution R
CONDWAT	Conductivity	0.000001 S/m	10
TEMPWAT	Temperature	0.00001 °C	10

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

This Appendix contains all Matlab subroutines necessary for performing the Stuck Value Test quality control algorithm as described herein.

A.1 Stuck Value Test Quality Control Algorithm

```
% DATAQC_STUCKVALUETEST   Data quality control algorithm testing a
%     time series for "stuck values", i.e. repeated occurences of
%     one value. Returns 1 for presumably good data and 0 for data
%     presumed bad.
%
% Time-stamp: <2011-10-31 11:20:23 mlankhorst>
%
% USAGE:   OUT=dataqc_stuckvaluetest(X,RESO,NUM);
%
%       OUT:  Boolean output: 0 where stuck values are found,
%             1 elsewhere.
%       X:    Input time series (vector, numeric).
%       RESO: Resolution; repeat values less than RESO apart will
%             be considered "stuck values".
%       NUM:  Minimum number of successive values within RESO of
%             each other that will trigger the "stuck value". NUM
%             is optional and defaults to 10 if omitted or empty.
%
% EXAMPLE:
%
% >> x=[4.83  1.40  3.33  3.33  3.33  3.33  4.09  2.97  2.85  3.67];
%
% >> dataqc_stuckvaluetest(x,.001,4)
%
% ans =
%
%       1     1     0     0     0     0     1     1     1     1
%


function out=dataqc_stuckvaluetest(varargin);

  error(nargchk(2,3,nargin,'struct'))

  x=varargin{1};
  reso=varargin{2};
  num=10;

  switch nargin
   case 3,
    if ~isempty(varargin{3})
      num=varargin{3};
    end
  end

  if ~isnumeric(x)
    error('X must be numeric.')
  end
  if ~isvector(x)
```

```
  error('X must be a vector.')
end

if ~isnumeric(reso)
  error('RESO must be numeric.')
end
if ~isscalar(reso)
  error('RESO must be a scalar.')
end
if ~isreal(reso)
  error('RESO must be real.')
end
reso=abs(reso);

if ~isnumeric(num)
  error('NUM must be numeric.')
end
if ~isscalar(num)
  error('NUM must be a scalar.')
end
if ~isreal(num)
  error('NUM must be real.')
end
num=abs(num);

ll=length(x);
out=zeros(size(x));
out=logical(out);

if ll<num
  warning('NUM is greater than length(X). Returning zeros.')
else
  out=ones(size(x));
  iimax=ll-num+1;
  for ii=1:iimax
    ind=[ii:ii+num-1];
    tmp=abs(x(ii)-x(ind));
    if all(tmp<reso)
          out(ind)=0;
    end
  end
end
out=logical(out);
```

A.2 isnumeric – Determine whether input is numeric array

Syntax

```
tf = isnumeric(A)
```

Description

`tf = isnumeric(A)` returns logical `1` (`true`) if `A` is a numeric array and logical `0` (`false`) otherwise. For example, sparse arrays and double-precision arrays are numeric, while strings, cell arrays, and structure arrays and logicals are not.

Examples

Given the following cell array,

```
C{1,1} = pi;                  % double
C{1,2} = 'John Doe';          % char array
C{1,3} = 2 + 4i;              % complex double
C{1,4} = ispc;                % logical
C{1,5} = magic(3)             % double array

C =
   [3.1416] 'John Doe' [2.0000+ 4.0000i] [1][3x3 double]
```

`isnumeric` shows that all but `C{1,2}` and `C{1,4}` are numeric arrays.

```
for k = 1:5
x(k) = isnumeric(C{1,k});
end

x
x =
     1     0     1     0     1
```

A.3 isvector – Determine whether input is vector

Syntax

```
isvector(A)
```

Description

`isvector(A)` returns logical `1` (`true`) if size(`A`) returns `[1 n]` or `[n 1]` with a nonnegative integer value `n`, and logical `0` (`false`) otherwise.

Examples

Test matrix `A` and its row and column vectors:

```
A = rand(5);

isvector(A)
ans =
     0

isvector(A(3, :))
ans =
     1

isvector(A(:, 2))
ans =
     1
```

A.4 isscalar – Determine whether input is scalar

Syntax

```
isscalar(A)
```

Description

`isscalar(A)` returns logical `1` (`true`) if `size`(`A`) returns [1 1], and logical `0` (`false`) otherwise.

Examples

Test matrix `A` and one element of the matrix:

```
A = rand(5);

isscalar(A)
ans =
     0

isscalar(A(3,2))
ans =
     1
```

A.5 isreal – Check if input is real array

Syntax

```
TF = isreal(A)
```

Description

`TF = isreal(A)` returns logical `1` (`true`) if `A` does not have an imaginary part. It returns logical `0` (`false`) otherwise. If `A` has a stored imaginary part of value `0`, `isreal(A)` returns logical `0` (`false`).

> **Note** For `logical` and `char` data classes, `isreal` always returns `true`. For numeric data types, if `A` does not have an imaginary part `isreal` returns `true`; if `A` does have an imaginary part `isreal` returns `false`. For `cell`, `struct`, `function_handle`, and `object` data types, `isreal` always returns `false`.

`~isreal(x)` returns `true` for arrays that have at least one element with an imaginary component. The value of that component can be `0`.

Tips

If `A` is real, `complex(A)` returns a complex number whose imaginary component is `0`, and `isreal(complex(A))` returns `false`. In contrast, the addition `A + 0i` returns the real value `A`, and `isreal(A + 0i)` returns `true`.

If `B` is real and `A = complex(B)`, then `A` is a complex matrix and `isreal(A)` returns `false`, while `A(m:n)` returns a real matrix and `isreal(A(m:n))` returns `true`.

Because MATLAB software supports complex arithmetic, certain of its functions can introduce significant imaginary components during the course of calculations that appear to be limited to real numbers. Thus, you should use `isreal` with discretion.

Example 1

If a computation results in a zero-value imaginary component, `isreal` returns `true`.

```
x=3+4i;
y=5-4i;
isreal(x+y)

ans =

     1
```

Example 2

These examples use `isreal` to detect the presence or absence of imaginary numbers in an array. Let

```
x = magic(3);
y = complex(x);
```

`isreal(x)` returns `true` because no element of `x` has an imaginary component.

```
isreal(x)
ans =
     1
```

`isreal(y)` returns `false`, because every element of `x` has an imaginary component, even though the value of the imaginary components is `0`.

```
isreal(y)
ans =
     0
```

This expression detects strictly real arrays, i.e., elements with `0`-valued imaginary components are treated as real.

```
~any(imag(y(:)))
ans =
     1
```

Example 3

Given the following cell array,

```
C{1} = pi;                  % double
C{2} = 'John Doe';          % char array
C{3} = 2 + 4i;              % complex double
C{4} = ispc;                % logical
C{5} = magic(3);            % double array
C{6} = complex(5,0)         % complex double

C =
  [3.1416]  'John Doe'  [2.0000+ 4.0000i]  [1]  [3x3 double]  [5]
```

`isreal` shows that all but `C{1,3}` and `C{1,6}` are real arrays.

```
for k = 1:6
x(k) = isreal(C{k});
end

x
x =
     1     1     0     1     1     0
```

A.6 isempty – Test if array is empty

Syntax

```
tf = isempty(A)
```

Description

`tf = isempty(A)` returns logical true (`1`) if `A` is an empty array and logical false (`0`) otherwise. An empty array has at least one dimension of size zero, for example, 0-by-0 or 0-by-5.

Examples

```
B = rand(2,2,2);
B(:,:,:) = [];

isempty(B)

ans =
     1
```